Charsets & Encodings

Kore Nordmann <kore@php.net>, Manuel Pichler
<mapi@pdepend.org>

November 18, 2009

About us

- Kore Nordmann, <kore@php.net>
  - Long time PHP developer
  - Regular speaker, author, etc.
  - Active open source developer: eZ Components (Graph, WebDav, Document), Arbit, PHPUnit, Torii, *PHPillow*, KaForkL, Image 3D, WCV, ...
- Manuel Pichler, <mapi@pdepend.org>
  - Born 1978
  - Diplom in Computer Science
  - Active open source developer: PHP_Depend, phpUnderControl, PHPMD

Outline

The terms

abc

?

.:?!

The terms

- Character versus byte

The terms

- Character versus byte
  - UTF-8 byte sequence: 0xE2 0x98 0x83
  - UTF-16 byte sequence: 0x26 0x03

The terms

- Character versus byte
  - UTF-8 byte sequence: 0xE2 0x98 0x83
  - UTF-16 byte sequence: 0x26 0x03
- Character set = Set of characters (Unicode)

The terms

- Character versus byte
  - UTF-8 byte sequence: 0xE2 0x98 0x83
  - UTF-16 byte sequence: 0x26 0x03
- Character set = Set of characters (Unicode)
- Encoding = Mapping of characters to bytes

The terms

- Character versus byte
  - UTF-8 byte sequence: 0xE2 0x98 0x83
  - UTF-16 byte sequence: 0x26 0x03
- Character set = Set of characters (Unicode)
- Encoding = Mapping of characters to bytes
  - Or byte sequences, for multi-byte encodings (UTF-8, UTF-16, UCS2, . . .)

The terms

- Character versus byte
  - UTF-8 byte sequence: 0xE2 0x98 0x83
  - UTF-16 byte sequence: 0x26 0x03
- Character set = Set of characters (Unicode)
- Encoding = Mapping of characters to bytes
  - Or byte sequences, for multi-byte encodings (UTF-8, UTF-16, UCS2, . . .)
- Collation

Unicode

- Unicode is character set which is intended to contain each character from any language

Unicode

- Unicode is character set which is intended to contain each character from any language
  - Removes the encodings hassles in multi language environments

Unicode

- Unicode is character set which is intended to contain each character from any language
  - Removes the encodings hassles in multi language environments
- There are different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible

Unicode

- Unicode is character set which is intended to contain each character from any language
  - Removes the encodings hassles in multi language environments
- There are different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible
  - UTF-16, does not encode full unicode, always uses 2 bytes

Unicode

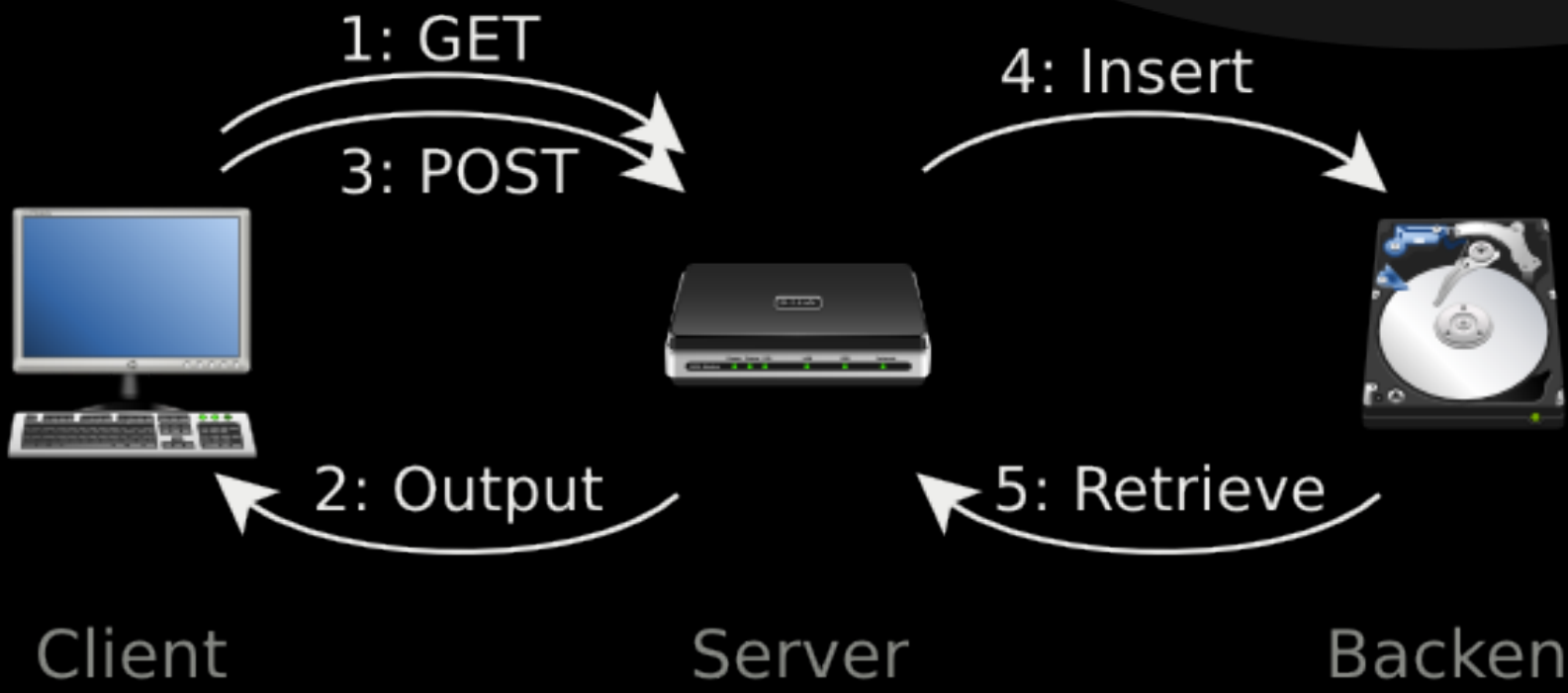
- Unicode is character set which is intended to contain each character from any language
  - Removes the encodings hassles in multi language environments
- There are different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible
  - UTF-16, does not encode full unicode, always uses 2 bytes
  - UTF-32, encodes full unicode, always uses 4 bytes

Unicode

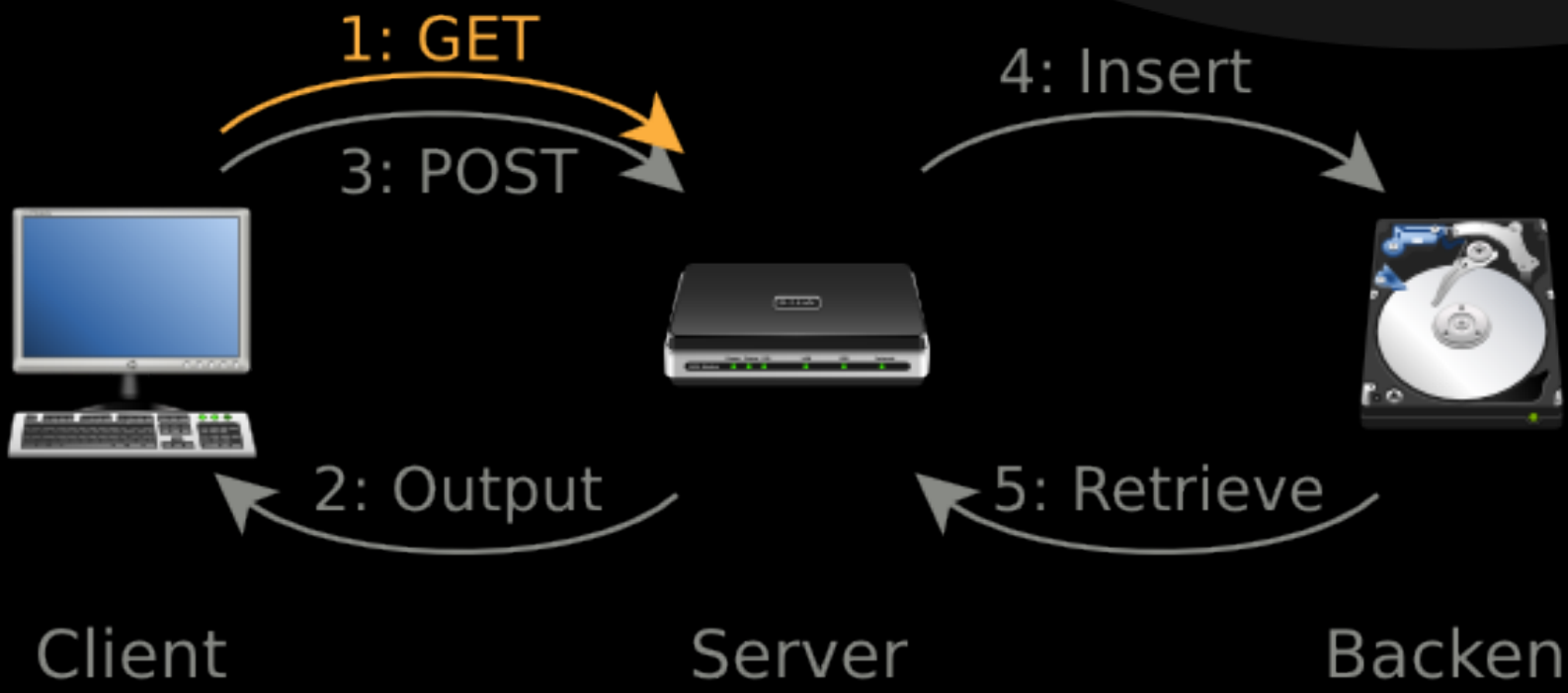
- Unicode is character set which is intended to contain each character from any language
  - Removes the encodings hassles in multi language environments
- There are different Unicode encodings
  - UTF-8, encodes full unicode, uses something between 1 and 4 bytes, ASCII compatible
  - UTF-16, does not encode full unicode, always uses 2 bytes
  - UTF-32, encodes full unicode, always uses 4 bytes
  - UTF-7, UCS2, UCS4, ...

Outline

Common simplified request cycle



Common simplified request cycle



The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The browser tells us about the encodings it understands

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The browser tells us about the encodings it understands
  - ... in the `Accept-Charset` header

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The browser tells us about the encodings it understands
  - ... in the `Accept-Charset` header
  - Most claim to understand *all* encodings: `*;q=0.7`

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The browser tells us about the encodings it understands
  - ... in the `Accept-Charset` header
  - Most claim to understand *all* encodings: `*;q=0.7`
  - Nearly all understand UTF-8 today

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

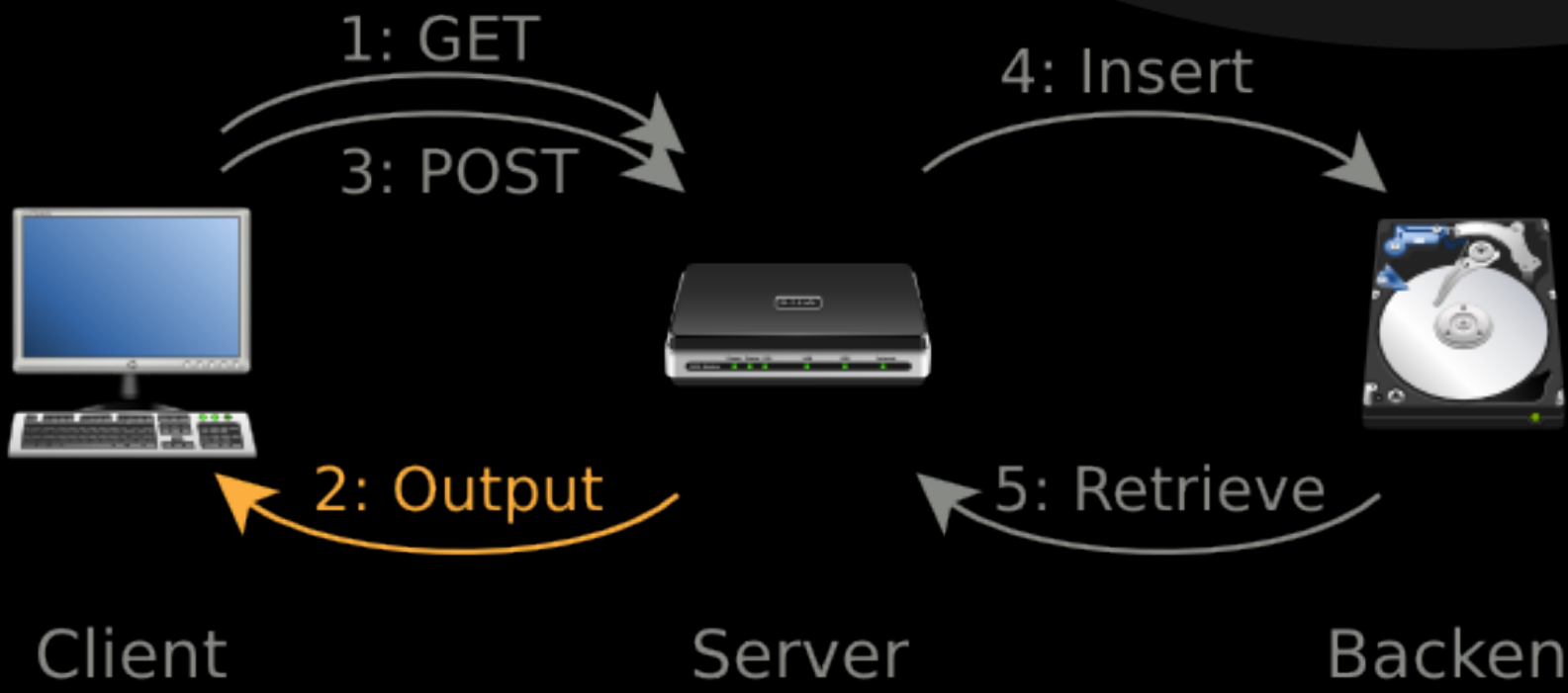
- The browser tells us about the encodings it understands
  - ... in the `Accept-Charset` header
  - Most claim to understand *all* encodings: `*;q=0.7`
  - Nearly all understand UTF-8 today
- Use the known encodings for displayed content

The browser request


```
1 GET /path HTTP/1.1
2 Host:            example.com
3 User-Agent:      Mozilla/5.0 (X11; U; Linux x86_64; en-
      US; rv:1.9.1.5) Gecko/20091109 Firefox/3.5.5
4 Accept:          text/html,application/xhtml+xml,
      application/xml;q=0.9,*/*;q=0.8
5 Accept-Encoding: gzip,deflate
6 Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The browser tells us about the encodings it understands
  - ... in the `Accept-Charset` header
  - Most claim to understand *all* encodings: `*;q=0.7`
  - Nearly all understand UTF-8 today
- Use the known encodings for displayed content
- Usage of charset vs. encoding: `http://kore-nordmann.de/blog/0082_charset_versus_encoding.html`

Common simplified request cycle



The output

- Depending on the client the output encoding is determined by:

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)
  - HTML-Meta-Section

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)
  - HTML-Meta-Section
- Always define both, and keep them consistent

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)
  - HTML-Meta-Section
- Always define both, and keep them consistent
  - Automatic detection can break *a lot*

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)
  - HTML-Meta-Section
- Always define both, and keep them consistent
  - Automatic detection can break *a lot*
- Use UTF-8 or an encoding, known by the client

The output

- Depending on the client the output encoding is determined by:
  - HTTP-Header (Content-Type)
  - HTML-Meta-Section
- Always define both, and keep them consistent
  - Automatic detection can break *a lot*
- Use UTF-8 or an encoding, known by the client
  - Always using UTF-8 minimizes usage of Entities and necessary conversions

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?
  - None, they are just arrays of bytes

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?
  - None, they are just arrays of bytes
  - You need to maintain the encoding information yourself

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?
  - None, they are just arrays of bytes
  - You need to maintain the encoding information yourself
  - This will probably change in PHP 6

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?
  - None, they are just arrays of bytes
  - You need to maintain the encoding information yourself
  - This will probably change in PHP 6
- This can make string processing harder in some situations

Strings in PHP

- Which encoding / charset do strings have in PHP anyways?
  - None, they are just arrays of bytes
  - You need to maintain the encoding information yourself
  - This will probably change in PHP 6
- This can make string processing harder in some situations
  - Length of a string:

```
1 $ php -r 'var_dump(
2       strlen( öäü"" ),
3       iconv_strlen( öäü"", "UTF-8" )
4 );'
5 int(6)
6 int(3)
```

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters
  - `ASCII//TRANSLIT`: Transliterate unknown characters

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters
  - `ASCII//TRANSLIT`: Transliterate unknown characters
  - Transliteration depends on the libC PHP is linked against, and the known encodings there.

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters
  - `ASCII//TRANSLIT`: Transliterate unknown characters
  - Transliteration depends on the libC PHP is linked against, and the known encodings there.
- There is no sense in using `utf8_(de|en)code()`

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters
  - `ASCII//TRANSLIT`: Transliterate unknown characters
  - Transliteration depends on the libC PHP is linked against, and the known encodings there.
- There is no sense in using `utf8_(de|en)code()`
  - Supports *only* ISO-8859-1 to UTF-8 conversions

Encoding conversions

- Very simple with the PHP core function `iconv()`

```
1 iconv( "UTF-8", "ASCII", "Umlauts: öäü" );
```

- The problem are charset conversions, two options:
  - `ASCII//IGNORE`: Ignore unknown characters
  - `ASCII//TRANSLIT`: Transliterate unknown characters
  - Transliteration depends on the libC PHP is linked against, and the known encodings there.
- There is no sense in using `utf8_(de|en)code()`
  - Supports *only* ISO-8859-1 to UTF-8 conversions
  - Does not handle transliteration.

Output escaping

- With Unicode-encodings there is no need for `htmlentities()`

Output escaping

- With Unicode-encodings there is no need for `htmlentities()`
- `htmlentities()`, improperly used, might even break your output

```
1 $ php -r 'var_dump( htmlentities( ü"" ) );'
2 string(16) "&Atilde;&frac14;"
```

Output escaping

- With Unicode-encodings there is no need for `htmlentities()`
- `htmlentities()`, improperly used, might even break your output

```
1 $ php -r 'var_dump( htmlentities( ü"" ) );'
2 string(16) "&Atilde;&frac14;"
```

- Always use the encoding parameter:

```
1 $ php -r 'var_dump( htmlspecialchars( ü"",
      ENT_QUOTES, "UTF-8" ) );'
2 string(2) ü""
```

Output escaping

- With Unicode-encodings there is no need for `htmlentities()`
- `htmlentities()`, improperly used, might even break your output

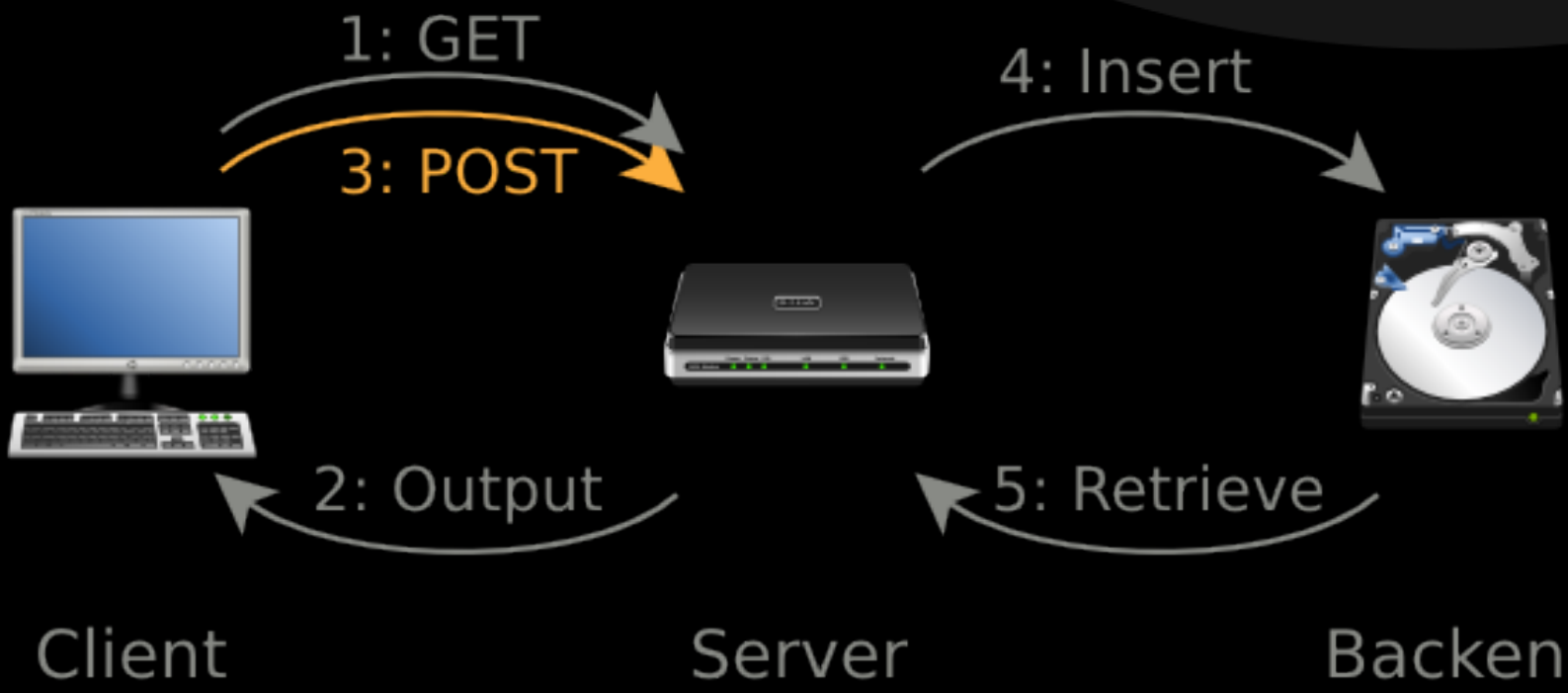
```
1 $ php -r 'var_dump( htmlentities( ü"" ) );'
2 string(16) "&Atilde;&frac14;"
```

- Always use the encoding parameter:

```
1 $ php -r 'var_dump( htmlspecialchars( ü"",
      ENT_QUOTES, "UTF-8" ) );'
2 string(2) ü""
```

- Should of course use the defined output encoding

Common simplified request cycle



Client data

- Which encoding does client data have?

Client data

- Which encoding does client data have?
  - If site encoding is consistently defined: The site encoding.

Client data

- Which encoding does client data have?
  - If site encoding is consistently defined: The site encoding.
  - Otherwise: Generally indeterministic, depends on the client implementation.

Client data

- Which encoding does client data have?
  - If site encoding is consistently defined: The site encoding.
  - Otherwise: Generally indeterministic, depends on the client implementation.
- `<form ... accept-charset="$encoding">` does *not* help.

Client data

- Which encoding does client data have?
  - If site encoding is consistently defined: The site encoding.
  - Otherwise: Generally indeterministic, depends on the client implementation.
- `<form ... accept-charset="$encoding">` does *not* help.
  - Won't hurt setting it to the site encoding value, though.

Client data

- Which encoding does client data have?
  - If site encoding is consistently defined: The site encoding.
  - Otherwise: Generally indeterministic, depends on the client implementation.
- `<form ... accept-charset="$encoding">` does *not* help.
  - Won't hurt setting it to the site encoding value, though.
  - Setting it to something different, might seriously fuck up your data, though

Client data

- Why not just “detect” the client encoding?

Client data

- Why not just “detect” the client encoding?
  - ... because it is impossible.

Client data

- Why not just “detect” the client encoding?
  - … because it is impossible.
- Detect if a byte sequence could be interpreted as UTF-8:

```
1  (^(?:
2       [\x00-\x7f] |
3       [\xc0-\xdf][\x80-\xff] |
4       [\xe0-\xef][\x80-\xff]{2} |
5       [\xf0-\xf7][\x80-\xff]{3}
6  )*$)x
```

Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.

Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.
- Simple check:

```
1 $input === iconv( 'utf-8', 'utf-8//IGNORE', $input
      )
```

Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.
- Simple check:

```
1 $input === iconv( 'utf-8', 'utf-8//IGNORE', $input
      )
```

- `iconv()` throws notices for invalid UTF-8 byte sequences

Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.
- Simple check:

```
1 $input === iconv( 'utf-8', 'utf-8//IGNORE', $input
      )
```

- `iconv()` throws notices for invalid UTF-8 byte sequences
  - Either strip invalid byte sequences, ...

Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.
- Simple check:

```
1 $input === iconv( 'utf-8', 'utf-8//IGNORE', $input
      )
```

- `iconv()` throws notices for invalid UTF-8 byte sequences
  - Either strip invalid byte sequences, ...
  - ... or error on invalid input, ...

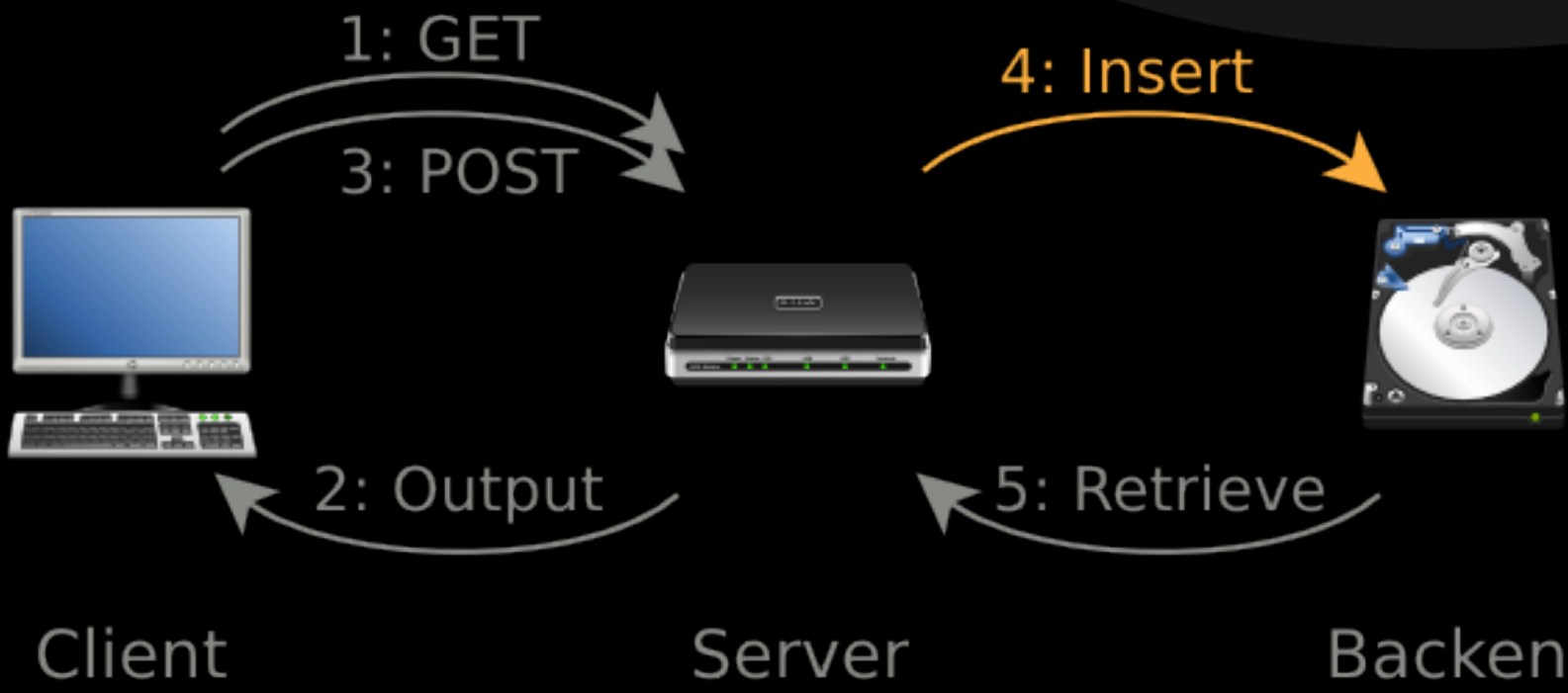
Client data

- You need to ensure client data encoding, or it will be *really* hard to debug encoding issues.
- Simple check:

```
1 $input === iconv( 'utf-8', 'utf-8//IGNORE', $input
      )
```

- `iconv()` throws notices for invalid UTF-8 byte sequences
  - Either strip invalid byte sequences, ...
  - ... or error on invalid input, ...
  - ... or flag the data for checks.

Common simplified request cycle



Backends

- There are numerous different backends, all are different:

Backends

- There are numerous different backends, all are different:
  - Databases (MySQL, PostgreSQL, CouchDB, . . .)

Backends

- There are numerous different backends, all are different:
  - Databases (MySQL, PostgreSQL, CouchDB, ...)
  - Web-Services (REST, XMLRPC, SOAP, ...)

- Most important setting is the connection encoding

MySQL

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)

MySQL

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)
  - Or use `SET NAMES "utf-8"` with prepared statements

MySQL

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)
  - Or use `SET NAMES "utf-8"` with prepared statements
- Table (column) encodings do not matter

MySQL

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)
  - Or use `SET NAMES "utf-8"` with prepared statements
- Table (column) encodings do not matter
  - Encoding does need to match the charset, though.

MySQL

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)
  - Or use `SET NAMES "utf-8"` with prepared statements
- Table (column) encodings do not matter
  - Encoding does need to match the charset, though.
  - Optimize to your needs, most columns do not need to be UTF-8

- Most important setting is the connection encoding
- MySQL will convert from and to table (column) encoding
  - Use C-API call, otherwise the connection will not know about changed encoding (`mysql_real_escape_string()`)
  - Or use `SET NAMES "utf-8"` with prepared statements
- Table (column) encodings do not matter
  - Encoding does need to match the charset, though.
  - Optimize to your needs, most columns do not need to be UTF-8
  - varchar($n) allocates $\$n * 3$ bytes

Webservices

- JSON “always” uses UTF-8

Webservices

- JSON “always” uses UTF-8
  - At least the PHP functions do

Webservices

- JSON “always” uses UTF-8
  - At least the PHP functions do
  - You may use UTF-16 or UTF-32, but those two are actually not differentiable

Webservices

- JSON “always” uses UTF-8
  - At least the PHP functions do
  - You may use UTF-16 or UTF-32, but those two are actually not differentiable
- XML declares the used encoding in the XML header `<?xml encoding="utf-8" ?>`

Outline

Conclusion

- What encoding should I use in my application?

Conclusion

- What encoding should I use in my application?
  - Use UTF-8 consistently, everywhere.

Conclusion

- What encoding should I use in my application?
  - Use UTF-8 consistently, everywhere.
    - Except for some storage optimization reasons in the backend

Conclusion

- What encoding should I use in my application?
  - Use UTF-8 consistently, everywhere.
    - Except for some storage optimization reasons in the backend
- Proper encoding handling is not hard. Just stay consistent.

The end

- Open questions?
- Further remarks?
- Contact
  - Mail: <kore@php.net>, <mapi@pdepend.org>
  - Twitter: `http://twitter.com/koredn` / `http://twitter.com/manuelp`
  - Web: `http://kore-nordmann.de/` (Slides will be available here soonish) / `http://manuel-pichler.de`
- More information
  - PHP Charset/Encding FAQ: `http://kore-nordmann.de/blog/php_charset_encoding_FAQ.html`