

Welcome!

Share your information

About Me

- Dutchie living in Norway
- eZ systems A.S.
- eZ components project lead
- PHP development
- mcrypt, input_filter, date/time support, unicode
- QA

What is so important about Unicode anyway?

- There is more than one country in the world
- Ce n'est pas tout le monde qui parle anglais
- Tjueseks karakterer holder ikke mål
- Нот эврибади из юзин зэ сэйм скрипт ивэн
- 它变得更加复杂的与汉语语言

i18n challenges

- Support for multiple encodings: conversion, detection, processing...
- Support for multiple language in different encodings and scripts

Sorting Strings

How would you sort: côté (side), côte (coast), cote (dimension), coté (with dimensions)?

Logical is: cote, coté, côte, côté

But the french do it like: cote, côte, coté, côté

The french are not the only ones with "weird" sorting!

- In Lithuanian, y is sorted between i and k.
- In traditional Spanish ch is treated as a single letter, and sorted between c and d.
- In Swedish v and w are considered variant forms of the same letter.
- In German dictionaries, öf would come before of. In phone books the situation is the exact opposite.

Conversion Between ISO-8859 Character Sets

- Each set has only 256 positions
- Impossible to convert everything
- Conversion will result in broken text
- <http://www.eki.ee/letter/>

ISO 8859-1	ISO-8859-2	ISO 8859-1	ISO-8859-2	Description
Hex	Char	Hex	Char	
E0	à	--	-	LATIN SMALL LETTER A WITH GRAVE
E1	á	E1	á	LATIN SMALL LETTER A WITH ACUTE
E2	â	E2	â	LATIN SMALL LETTER A WITH CIRCUMFLEX
E3	ã	--	-	LATIN SMALL LETTER A WITH TILDE
E4	ä	E4	ä	LATIN SMALL LETTER A WITH DIAERESIS
E5	å	--	-	LATIN SMALL LETTER A WITH RING ABOVE
E6	æ	--	-	LATIN SMALL LETTER AE
E7	ç	E7	ç	LATIN SMALL LETTER C WITH CEDILLA
E8	è	--	-	LATIN SMALL LETTER E WITH GRAVE
E9	é	E9	é	LATIN SMALL LETTER E WITH ACUTE
EA	ê	--	-	LATIN SMALL LETTER E WITH CIRCUMFLEX
EB	ë	EB	ë	LATIN SMALL LETTER E WITH DIAERESIS
EC	ì	--	-	LATIN SMALL LETTER I WITH GRAVE
ED	í	ED	í	LATIN SMALL LETTER I WITH ACUTE
EE	î	EE	î	LATIN SMALL LETTER I WITH CIRCUMFLEX
EF	ï	--	-	LATIN SMALL LETTER I WITH DIAERESIS

Do We Need Something New?

- PHP only deals with bytes, not characters.
- PHP doesn't know anything about encodings.
- Having a binary image in a string is nice, but not if you need to deal with i18n

Iconv

- Is part of glibc
- BSD requires an external library
- Enabled by default in PHP 5
- Supports 100s of character sets

But it doesn't solve:

localization, sorting, searching, encoding detection and you still work with binary data only!

mbstring

- Handles certain encoding problems for you
- Updates string functions by "overloading" them

But it doesn't solve:

localization, sorting, searching and you still work with binary data only!

Anything Else That Sucks?

- Some of PHP's functions can make use of POSIX-locales
- But that's only a few of them
- Those locales are system dependent (different names, rules, etc...)
- They are not always available

Unicode and ISO-10646 (UCS)

- UCS uses 31 bits for character storage
- Contains all known characters and symbols
- First 128 bytes are the same as ASCII
- First 256 bytes are the same as ISO-8859-1
- Unicode 3.0 describes the BMP (Basic Multilingual Plane) (16 bits)
- Unicode 3.1 describes other planes (21 bits)
- Characters are ordered in language/script blocks: Basic latin, Cyrillic, Hebrew, Arabic, Gujarati, Runic, CJK etc.
- Encoding in numerous encodings: UCS-2, UCS-4, UTF-8, UTF-16 etc.

A Ъ ث X

Building New Characters

You can compose new characters from base characters with combining modifiers that use no "space".

Equivalents:

å != å

U+00C5 != U+0041 + U+030A

Alternative Order:

a + ^ + = â

a + + ^ = â

Unicode Is More

- Unicode is a multi-language character set
- Standard encodings: UTF-8, UTF-16 and UTF-32
- Defines algorithms for plenty of issues (Collation, Bidi, Normalization)
- It defines properties for characters:

Å

```
00C5;LATIN CAPITAL LETTER A WITH RING ABOVE;Lu;0;L;0041 030A;;;
N;LATIN CAPITAL LETTER A RING;;;00E5;
```

ĎŽ

```
01C4;LATIN CAPITAL LETTER DZ WITH CARON;Lu;0;L;<compat> 0044 017D;;;
N;LATIN CAPITAL LETTER D Z HACEK;;;01C6;01C5
```

Å

```
212B;ANGSTROM SIGN;Lu;0;L;00C5;;;
N;ANGSTROM UNIT;;;00E5;
```

What Do We Want for PHP?

- Native Unicode strings
- A clear separation between Binary / Native (Encoded) Strings and Unicode Strings
- Unicode string literals
- Updated language semantics
- Where possible, upgrade the existing functions
- Backwards compability
- PHP should do what most people will expect
- Make complex things possible, without making using strings in PHP complex
- Must be as good as Java's support

How is It Going To Work?

- UTF-16 as internal encoding
- All functions and operators work on Normalized Composed Characters (NFC)
- All identifiers can contain Unicode characters
- Internationalization is explicit, not implicit
- You can turn off Unicode semantics if you don't need it

UTF 16 Surrogates

As UTF-16 is supposed to encode the full Unicode character set.

- UTF-16 uses a double two-byte sequence for characters outside the BMP
- Special ranges in the Unicode range are used for this

byte 1 = 0xd800 - 0xdbff

byte 2 = 0xdc00 - 0xffff

U+10418 DESERET CAPITAL LETTER GAY

0xd810 0xdc18

- Code point: a character
- Code unit: a two-byte sequence with UTF-16

How Will It Be Implemented?

ICU: International Components for Unicode

- Unicode is extremely complex, with ICU we don't have to implement it ourselves
- ICU has a lot of features, is fast, stable, portable, extensible, Open Source and well maintained and supported

ICU Features

- Character, String and Text processing
- Text Transformations
- Encoding Conversions
- Collation
- Localization: date, time, number, currency formatting

Roadmap

- Support Unicode in the Engine
- Upgrade existing functions
- Add new functions for explicit i18n/l10n support
- Expose ICU's features

How Do We Turn It On?

- With an INI setting: `unicode_semantics`
- It can be turned on "per-vhost"
- (Almost) no behavioral changes when it's not enabled
- The setting does not mean you won't have any Unicode strings

String Types

- **string:** Used to represent binary data, for example the contents of a JPEG file or a "native" string.

```
. P N G . . . . . I H D R  
89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52
```

```
B l å    b æ    r ø    l  
62 6C C3 A5 62 C3 A6 72 C3 A8 6C
```

- **unicode:** Strings, internally encoded in UTF-16.

```
B    l    å    b    æ    r    ø    l  
62 00 6C 00 E5 00 62 00 E6 00 72 00 F8 00 6C 00
```

Unicode Semantics are off:

```
<?php // script is encoded in UTF-8
$str = "hallo daar!";
echo gettype($str), ': ', strlen($str), "\n";

$str = "привет!";
echo gettype($str), ': ', strlen($str), "\n";
?>
```

outputs:

```
string: 11
string: 13
```

Unicode Semantics are on:

```
<?php // script is encoded in UTF-8
$str = "hallo daar!";
echo gettype($str), ': ', strlen($str), "\n";

$str = "привет!";
echo gettype($str), ': ', strlen($str), "\n";
?>
```

outputs:

```
unicode: 11
unicode: 7
```

Interpreting an iso-8859-1 script as UTF-8:

```
<?php
    declare(encoding="iso-8859-1");
    $str = "blå = 青";
    var_inspect($str);
?>
```

Interpreting an UTF-8 script as UTF-8:

```
<?php
    declare(encoding="utf-8");
    $str = "blå = 青";
    var_inspect($str);
?>
```

Characters, not Bytes!

- All functions and operators work on Code Points (characters) and not Code Units (bytes)
- Backward compatible if you only used single byte encodings before
- This does create overhead though, as we need to scan through a whole string

String Indexes:

```
<?php
    $string = "网页搜索";
    // bytes are: E7 BD 91 E9 A1 B5 E6 90 9C E7 B4 A2

    echo $string[1];
?>
```

ICU Locales

- ICU comes with it's own Locale information
- PHP currently uses POSIX locales for some functions only
- Those functions need to be modified

```
<?php
    locale_set_default("nl");
    echo strtotitle("het ijsselmeer (ijsselmeer) is ßaf"), "<br/>\n";

    locale_set_default("tr");
    echo strtotitle("het ijsselmeer (ijsselmeer) is ßaf"), "\n";
?>
```

Comparing strings:

```
<?php
$coll = new Collator("fr_CA");
if ($coll->compare("côte", "coté") < 0) {
    echo "less\n";
} else {
    echo "greater\n";
}
?>
```

Ignore case and accents:

```
<?php
$coll = new Collator("fr_CA");
$coll->setStrength(Collator::PRIMARY);
if ($coll->compare("côte", "cOTÉ") == 0) {
    echo "same\n";
} else {
    echo "different\n";
}
?>
```

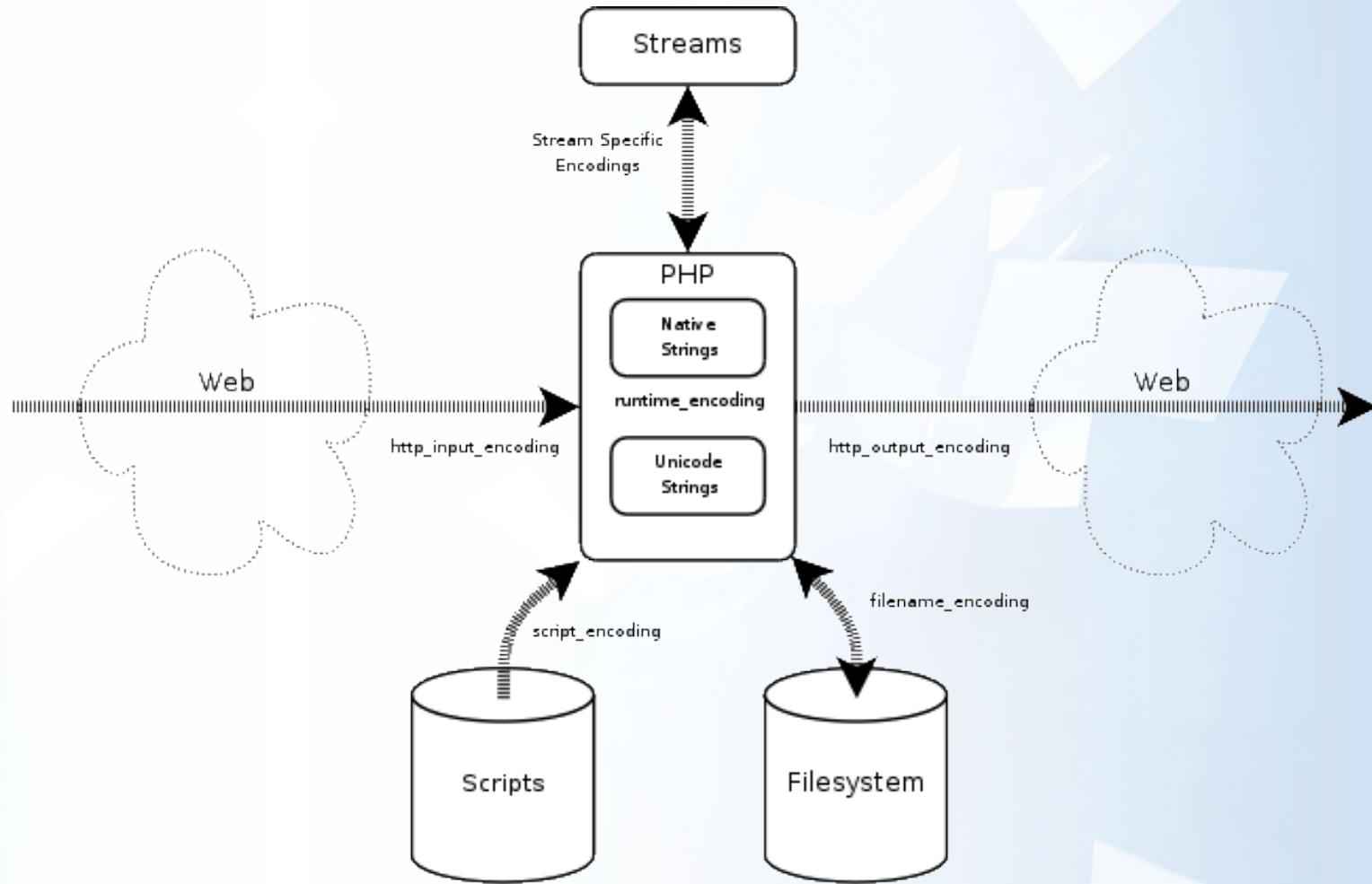
```
<pre># orig norm loc trad
-----
<?php
    $d = $c = $b = $a = array('mapa', 'kilo', 'libro', 'llave', 'loca');
    sort($b);
    locale_set_default('es_VE');
    sort($c, SORT_LOCALE_STRING);
    locale_set_default('es_VE@collation=traditional');
    sort($d, SORT_LOCALE_STRING);

    for ($i = 0; $i < 5; ++$i) {
        echo sprintf('%d. %-5s %-5s %-5s %-5s<br/>',
            $i + 1, $a[$i], $b[$i], $c[$i], $d[$i]);
    }
?>
```

i18n Extensions

- ext/unicode: always enabled, functions to set the locale, the locale aware string functions. String searching and collation API.
- i18n_regexp: Unicode aware regular expressions
- i18n_translit: PECL/translit extension, integrated with ICU's transliteration
- i18n_...: Other specialized extensions, such as normalization, break iteration...

Encodings Overview



Script Encoding

- Is used by the parser to read in your script
- Determines how string literals and identifiers are handled
- Can be set with an ini setting (`script_encoding`) or with an inline "pragma"
- No matter what the script's encoding is, the resulting string is always a Unicode string (or identifier)

Interpreting an iso-8859-1 script as UTF-8:

```
<?php
    declare(encoding="iso-8859-1");
    $str = "blå = 青";
    var_inspect($str);
?>
```

Interpreting an UTF-8 script as UTF-8:

```
<?php
    declare(encoding="utf-8");
    $str = "blå = 青";
    var_inspect($str);
?>
```

Runtime Encoding

- Determines which encoding to attach to Native Strings
- Also used when functions are not upgraded to support Unicode yet

Encoding problems:

```
<?php
    ini_set('unicode.runtime_encoding', 'iso-8859-5');
    ini_set('unicode.from_error_subst_char', '2D');
    $food = 'blåbær != блябар';
    $food2 = (binary) $food;
    $food3 = (string) $food2;
    echo $food, '<br/>', $food2, '<br/>', $food3, "<br/>\n";
?>
```

HTTP Output Encoding

- Is used as encoding for the output of the script
- Script output is encoding on the fly
- Binary strings will never be automatically converted

On the fly encoding:

```
<?php
declare(encoding="iso-8859-1");
ini_set('unicode.output_encoding', 'utf-8');

$str = "rødbærsyltetøy<br/>";
$bstr = b"rødbærsyltetøy<br/>";

echo $str, $bstr;
?>
```

Fallback Encoding

- Used if any of the other encoding settings is not set
- Easy way of configuring all encoding settings
- Defaults to UTF-8 if it's not set

INI Settings Recap:

```
unicode.script_encoding = "UTF-8"  
    Source encoding for your script  
  
unicode.runtime_encoding = "iso-8859-15"  
    Internal encoding used for "native strings"  
  
unicode.from_error_subst_char = "2f"  
    Hex value of substitution character  
  
unicode.http_input_encoding = "UTF-8"  
    Default encoding for HTTP input variables  
  
unicode.output_encoding = "UTF-8"  
    Encoding used for script output  
  
unicode.fallback_encoding = "UTF-8"  
    Fallback encoding
```

When Can We Have This?

- When it is ready.
- Development version is in CVS.
- Release: hopefully at the end of 2006.

Resources

- PDM notes: <http://php.net/~derick/meeting-notes.html>
- This presentation: <http://derickrethans.nl/talks.php>
- Questions: dr@ez.no