



Andrei Zmievski  
Chief Architect  
Outspark, Inc

PHP::\$unicode → i18n()

ZendCon 2007 San Francisco (aka Burlingame)

**Im in ur endginn**

**playin wif ur stringz**





**PHP 6 = PHP 5 + Unicode**



**PHP 5 = PHP 6 – Unicode**



**Unicode = PHP 6 – PHP 5**



# What is PHP?

---

Ha. Ha.



# What is Unicode?

---

and why do I need?



mojibake

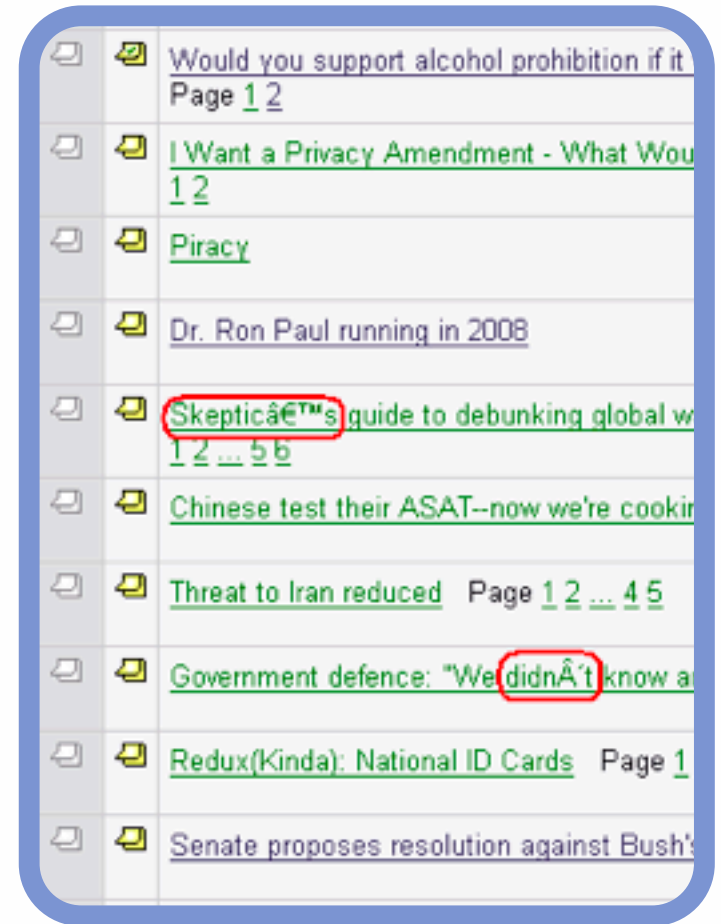
---

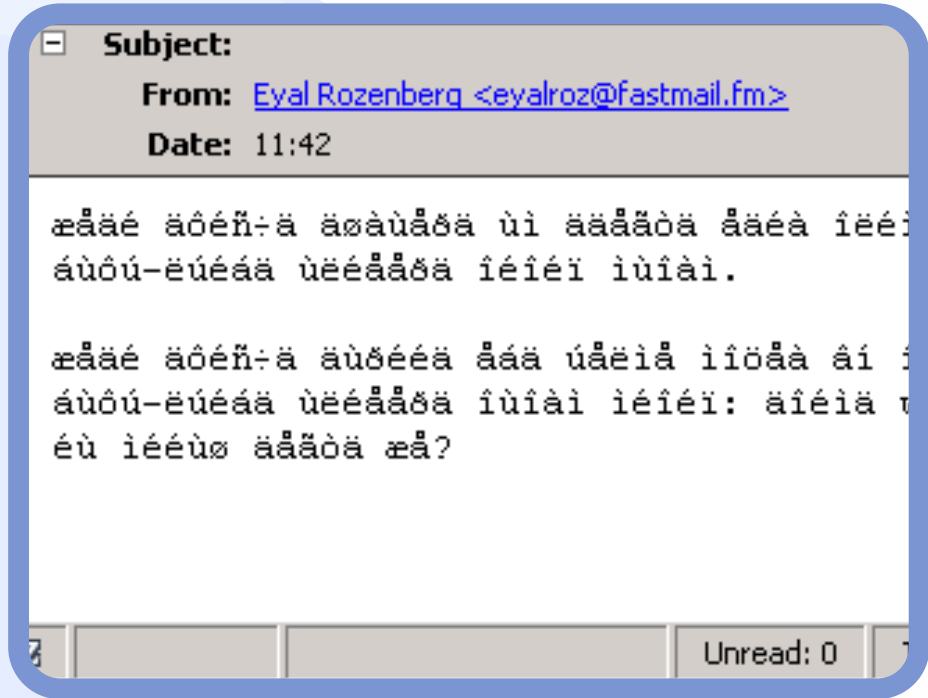
もじばけ



# mojibake

phenomenon of incorrect, unreadable characters shown when computer software fails to render a text correctly according to its associated character encoding





mojibake



The background features two overlapping, semi-transparent light blue circles on the left side of the slide. The larger circle is in the foreground, and a smaller one is behind it, both partially cut off by the left edge of the frame.

**Computers deal with numbers**

## Encoding soup

1. provinciality
2. computer limitations
3. inertia

# Unicode

provides a unique number  
for every character:

no matter what the platform,  
no matter what the program,  
no matter what the language.

# Unicode

'juːnɪˌkoʊd	ꠘꠞꠣꠘ	യൂണികോഡ്	ユニコード	يونى كُد
Ūnīcōdĕ	GhAᳵ	Ιούνικοντ	ಯೂನಿಕೋಡ್	Юникод
*ᲛᲠᲗᲚᲘ	统一码	യൂനികോഡ്	유니콘	யூனிகோட்
يونكود	統一碼	यूनिक्वड	ಯೂನಿಕೋಡ್	ಯೂನಿಕೋಡ್
Ἐνιϰὴλὸν	Уникод	ᲛᲠᲗᲚᲘ	ꠘꠞꠣꠘ	यूनिक्वड
यूनिकोड	ꠘꠞꠣꠘ	यूनिकोड	यूनिकोड	ꠘꠞꠣꠘ



# ○ ● ● business challenges

---

- Supporting languages needed for business
  - here or abroad
- Adding new languages (customers), easily



# ○ ● ● business challenges

---

- Increasingly, users are not satisfied with incorrect spellings, or restrictions to write their names, addresses, and other information in ASCII or incompatible encodings

# ○ ● ● technical challenges

---

- Differences in character encodings
- Require different algorithms
- Imply different code in each market
- High error rate and poor quality

# ○ ● ● unicode benefits

---

- allows for multilingual text using any or all the languages you desire
- invoice or ticketing applications can print customer information in their native languages from a single database

# ○ ● ● unicode benefits

---

- **one way to process text**
- **one version of the product can be used worldwide**

# ○ ● ● unicode benefits

---

- support of Unicode by modern technologies extends code life and broadens integration possibilities
- easier to take advantage of new technologies and integrate with other applications

# ○ ● ● unicode benefits

---

● internet-ready

○ XML, JavaScript, Firefox, Java, and now PHP, all Unicode-based

# ○ ● ● unicode standard

---

- Developed by the Unicode Consortium
- Covers all major living scripts
- Version 5.0 has 99,000+ characters
- Capacity for 1 million+ characters

# ○ ● ● unicode standard

---

- One character set for worldwide use
- Standard encodings: UTF-8, UTF-16, UTF-32
- International Standard – ISO 10646
- Precisely defined
- Widely supported by standards & industry



# ○ ● ● generative

- Composition can create “new” characters
- Base + non-spacing (combining) character(s)

A + ° = Å

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

a + ^ + . = â

U+0061 + U+0302 + U+0323 = U+1EAD

a + ˇ + ˇ = ă

U+0061 + U+0322 + U+030C = ?

# ○ ● ● unicode != i18n

---

- Unicode simplifies development
- Unicode does not fix all internationalization problems



**What is  $i18n$ ?**

---

# ○ ● ● definitions

---

## Internationalization

**I18n**

To design and develop an application:

- ✓ without built-in cultural assumptions
- ✓ that is **efficient** to localize

## Localization

**L10n**

To tailor an application to meet the needs of a particular region, market, or culture

# date formats ○ ● ●

- USA: 2/16/05
- France: 16.2.05 or 16-2-05
- Japan, China: 2005年2月16日

# calendars ○ ● ●

---

- Gregorian 2007
- Thailand: 2550 (Buddhist Year)
- Taiwan: 96 (1911-based)
- Hebrew: 5767
- Also Hijri (Islamic), Lunar (Asia) and many others

# time formats ○ ● ●

- USA: 4:00 P.M.
- France: 16.00
- Japan: 1600
- Don't forget to identify the time zone

# number formats

- England: 12,345.67
- Germany: 12.345,67
- Switzerland: 12'345,67
- Swiss money: 12'345.67
- France: 12 345,67
- India: 12,34,567.89



# currency ○ ● ●

- Symbol placement
- Symbol length (1-15)
- Number width
- Number precision:
  - Spain, Japan 0
  - Mexico, Brazil 2
  - Egypt, Iraq 3

US \$12.34

12.345,67 €

12\$34€

¥123

# sorting ○ ● ●

English: ABC...RSTUVWXYZ

German: AÄB...NOÖ...SßTUÜV...YZ

Swedish/Finnish: ABC...RSTUVWXYZÅÄÖ

- Languages may sort more than one way
  - traditional vs. modern Spanish
- Japanese stroke-radical vs. radical-stroke
- German dictionary vs. phone book

# sorting ○ ● ●

- Swedish:  $z < ö$
- German:  $ö < z$
- Dictionary:  $öf < of$
- Phonebook:  $of < öf$
- Upper-first:  $A < a$
- Lower-First:  $a < A$
- Contractions:  $H < Z$ , but  $CH > CZ$
- Expansions:  $OE < Œ < OF$

# locale data ○ ● ●

---

- I18N and L10N rely on consistent and correct locale data
- Problem with POSIX locales: not always consistent or correct

- Hosted by Unicode Consortium
- Goals:
  - Common, necessary software locale data for all world languages
  - Collect and maintain locale data
  - XML format for effective interchange
  - Freely available
- Latest release: July 2007 (CLDR 1.5)
- 394 locales, with 135 languages and 149 territories



# PHP 6

---

# ○ ● ● unicode support

---

## ● Everywhere:

- in the engine
- in the extensions
- in the API

# ○ ● ● unicode support

---

- Native and complete
  - no hacks
  - no mishmash of external libraries
  - no missing locales
  - no language bias



# string types

- Unicode

- text

- default for literals, etc

- Binary

- bytes

- everything  $\notin$  Unicode type

# string types ○ ● ●

---

- internal processing: Unicode
- interface to outside world: binary

- All string literals are Unicode

```
$str = "Hello, world!"; // Unicode string  
echo strlen($str);    // result is 13
```

```
$jpn = "検索オプション"; // Unicode string  
echo strlen($jpn);     // result is 7
```

- String offsets work on code points

```
$str = "大学"; // 2 code points  
echo $str[1]; // result is 学  
$str[0] = 'サ'; // full string is now サ学
```

# identifiers ○ ● ●

- Unicode identifiers are allowed

```
class コンポーネント {  
    function ັໂ ດໂ ດໂ ດໂ { ... }  
    function சிவாஜி கணேசன் { ... }  
    function අසුරා ආඥා { ... }  
}  
  
$プロバイダ = array();  
$プロバイダ['הַשָּׂרָא הַגִּיבּוֹר'] = new コンポーネント();
```

# functions ○ ● ●

## ● Functions understand Unicode text

- `strtoupper()` and friends do proper case mapping

```
$str = strtoupper("fußball"); // result is FUSSBALL  
$str = strtolower("ΣΕΛΛΑΣ"); // result is σελλάς
```

- `strip_tags()` works on complex text

```
$str = strip_tags("雅<span>είναι</span>通");
```

- `strrev()` preserves combining sequences

```
$u = "Việ\u0302\u0323t Nam"; // Việt Nam  
$str = strrev($u); // result is maN tậiV,  
// not maN ậiV
```

- Built-in support for converting between Unicode strings and other encodings on the fly
- Reading from a UTF-8 text file:

```
$fp = fopen('somefile.txt', 'rt');  
$str = fread($fp, 100); // returns 100 Unicode characters
```

- Writing to a UTF-8 text file:

```
$fp = fopen('somefile.txt', 'wt');  
fwrite($fp, $uni); // writes out data in UTF-8 encoding
```

- Grab first 5 titles from Reuters China feed, clean up, and send out as JSON

```
$xml = simplexml_load_file(
    'http://feeds.feedburner.com/reuters/CNTbusinessNews/');

$titles = array();
$i = 0;
foreach ($xml->channel->item as $item) {
    // each title looks like this: [台灣匯市] 台幣兌美元

    $title = preg_replace('!\p{Ps}.*\p{Pe}\s*!', '', $item->title);
    $titles[] = $title;
    if (++$i == 5) break;
}

echo json_encode($titles);
```





**pecl/intl**

---

- Builds on the ICU library
- Uses CLDR data
- Developed by Yahoo!, Zend, LiveNation (and me)

# features ○ ● ●

---

- *Locales*
- Collation
- Number and Currency Formatters
- Date and Time Formatters
- Time Zones
- Calendars
- Message Formatter
- Choice Formatter
- Resource Handler

- OO and procedural API
- Same underlying implementation

`collator_create() == new Collator()`

`collator_set_attribute() == Collator::setAttribute()`

`numfmt_format() == NumberFormatter::format()`

# versioning ○ ● ●

---

- Works under PHP 5 and 6
- Uses native strings in PHP 6
- Requires UTF-8 strings in PHP 5
- Can use mbstring, iconv



# Locale

---

# ○ ● ● locale format

---

- Locale = identifier referring to linguistic and cultural preferences of a user community
- pecl/intl relies exclusively on ICU locales
- ICU locale IDs have a somewhat different format from POSIX locale IDs

`<language>[_<script>]_<country>[_<variant>][@<keywords>]`

# ○ ● ● locale examples

---

- Example:

`en_GB`

English (Great Britain)

- Extended example:

`sr_Latn_YU_REVISIED@currency=USD`

Serbian (Latin, Yugoslavia, Revised Orthography, Currency=US Dollar)



# ○ ● ● default locale

---

- Do not use `setlocale()`
- Default locale can be accessed with:

```
Locale::set_default()
```

```
Locale::get_default()
```

# ○ ● ● locale pieces

---

- `getPrimaryLanguage($locale)`
- `getScript($locale)`
- `getRegion($locale)`
- `getVariant($locale)`
- `getKeywords($locale)`

# ○ ● ● localized pieces

---

- `getDisplayNames($locale, $in_locale = null)`
- `getDisplayLanguage($locale, $in_locale = null)`
- `getDisplayScript($locale, $in_locale = null)`
- `getDisplayRegion($locale, $in_locale = null)`

Example:

- `getDisplayScript(getScript("zh-Hant-TW"), "en-US")` returns "Traditional Chinese"

# ○ ● ● breaking and making

- **canonicalize()** - normalize locale according to RFC 4646
- **parseLocale()** - returns array composed of locale subtags
- **composeLocale()** - creates locale ID out of subtags

Examples:

- **parseLocale('sr-Latn-RS')** returns  
`array('language'=>'sr', 'script'=>'Latn', 'region'=>'RS')`
- **composeLocale(array('language'=>'sr', 'script'=>'Latn', 'region'=>'RS'))** returns  
`'sr-Latn-RS'`



# Collator

---

# sorting ○ ● ●

English: ABC...RSTUVWXYZ

German: AÄB...NOÖ...SßTUÜV...YZ

Swedish/Finnish: ABC...RSTUVWXYZÅÄÖ

- Languages may sort more than one way
  - traditional vs. modern Spanish
- Japanese stroke-radical vs. radical-stroke
- German dictionary vs. phone book

# sorting ○ ● ●

- Lithuanian:  $i < y < k$
- Swedish:  $v = w$
- Swedish:  $z < ö$
- German:  $ö < z$
- Dictionary:  $öf < of$
- Phonebook:  $of < öf$
- Contractions:  $H < Z$ , but  $CH > CZ$
- Expansions:  $OE < Œ < OF$

# collation levels ○ ● ●

---



- Also called “strengths”
- Each locale has default level setting
- Differences in lower levels are ignored if higher levels have differences



# collation levels

---

## Primary (1)

-  used to denote differences between base characters
-  “strongest” level

**a < b < c < d < e**

# collation levels



---

- **Secondary (2)**
  - distinguishes accents in characters
  - other differences, depending on language

as < às < at

# collation levels

## Tertiary (3)

-  distinguishes case in characters
-  as well as variants of a base form of a letter

ao < Ao < aò

A < Ⓐ

# collation levels

---

- Quaternary (4)
  - used when ignoring punctuation is required (at higher levels)
  - used when processing Japanese text

$ab < a-b < aB$

# collation levels

---

- Identical (5)**
  - used as tiebreaker**
  - when all other levels are equal**

# instantiation ○ ● ●

---

- `new Collator($locale)`
- `collator_create($locale)`

# error handling ○ ● ●

---

- **getErrorCode()**

- returns last error code

- **getErrorMessage()**

- returns last error message

# comparing strings ○ ● ●

- `compare($str1, $str2) = 0, -1, 1`

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

**côte < coté**



# sorting strings ○ ● ●

- `sort($array, $flags)`
- `asort($array, $flags)`
- `sortWithSortKeys($array)`

```
$strings = array(
    "cote", "côte", "Côte", "coté",
    "Coté", "côté", "Côté", "coter");
$coll = new Collator("fr_CA");
$coll->sort($strings);
```

```
cote
côte
Côte
coté
Coté
côté
Côté
coter
```

# strength control ○ ● ●

- `setStrength($strength)`
- `getStrength()`

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

**côte = coté**

# other attributes ○ ● ●

- `setAttribute($attr, $value)`
- `getAttribute($attr)`

```
$coll = new Collator("en_US");  
$coll->setAttribute(Collator::CASE_LEVEL,  
                  Collator::UPPER_FIRST);  
if ($coll->compare("abc", "ABC") < 0) {  
    echo "less\n";  
} else {  
    echo "greater\n";  
}
```

**ABC < abc**

# numeric collation ○ ● ●

## ● Collator::NUMERIC\_COLLATION

```
$strings = array("10", "1", "2");  
$coll->setStrength(Collator::NUMERIC_COLLATION,  
                  Collator::ON);  
$coll = new Collator(null);  
$coll->sort($strings);
```

1 < 2 < 10

# ignoring punctuation ○ ● ●

## ● Collator::ALTERNATE\_HANDLING

```
$strings = array("U.S.A.", "USA", "u.s.a.", "usa");  
$coll->setAttribute(Collator::ALTERNATE_HANDLING,  
                  Collator::SHIFTED);  
$coll->setAttribute(Collator::STRENGTH,  
                  Collator::TERTIARY);  
$coll = new Collator('en_US');  
$coll->sort($strings);
```

not ignored: u.s.a < U.S.A. < usa < USA

ignored: usa < u.s.a < USA < U.S.A.

# case level

## ● Collator::CASE\_LEVEL

```
$strings = array("role", "rôle", "Role");  
$coll->setAttribute(Collator::CASE_LEVEL,  
                  Collator::ON);  
$coll->setAttribute(Collator::STRENGTH,  
                  Collator::PRIMARY);  
$coll = new Collator('en_US');  
$coll->sort($strings);
```

without case level: Role = rôle = role

with case level: role = rôle < Role

- ICU collation supports custom “tailoring” rules
- make Cyrillic sort before Latin, for example
- Not exposed in pecl/intl yet



# NumberFormatter

---

it formats Benjamins too



# ○ ● ● what it is

---

- allows to format numbers as strings according to the localized format or given pattern or set of rules
- and parse strings into numbers according to the above patterns
- replacement for `number_format()`

# instantiating

---

- `new NumberFormatter($locale, $style, $pattern = null)`
- `numfmt_create($locale, $style, $pattern = null)`

**`$style`** is one of:

NumberFormatter::PATTERN\_DECIMAL  
NumberFormatter::DECIMAL  
NumberFormatter::CURRENCY  
NumberFormatter::PERCENT

NumberFormatter::ORDINAL  
NumberFormatter::DURATION  
NumberFormatter::SCIENTIFIC  
NumberFormatter::SPELLOUT

# formatter styles

---

123456.789 in en\_US

- NumberFormatter::PATTERN\_DECIMAL

123456.79 (with ##.##)

- NumberFormatter::DECIMAL

123456.789

- NumberFormatter::CURRENCY

\$123,456.79

- NumberFormatter::PERCENT

12,345,679%

# formatter styles

---

123456.789 in en\_US

- NumberFormatter::SCIENTIFIC

1.23456789E5

- NumberFormatter::SPELLOUT

one hundred and twenty-three thousand, four hundred and fifty-six point seven eight nine

- NumberFormatter::ORDINAL

123,457th

- NumberFormatter::DURATION

34:17:37

# error handling ○ ● ●

---

- **getErrorCode()**

- returns last error code

- **getErrorMessage()**

- returns last error message

# ● ● ● formatting

## ● `format($number [, $type])`

```
$fmt = new NumberFormatter('en_US',  
                             NumberFormatter::DECIMAL);  
  
$fmt->format(1234);  
// result is 1,234  
  
$fmt = new NumberFormatter('de_CH',  
                             NumberFormatter::DECIMAL);  
  
$fmt->format(1234);  
// result is 1'234
```

# ● ● ● formatting currency

- using **CURRENCY** style for default rules
- using **formatCurrency()** method, with ISO 4217 currency codes

```
$fmt = new NumberFormatter('ta_IN',  
                             NumberFormatter::CURRENCY);  
$fmt->format(1234);  
// result is ₹ 1,234.00  
  
$fmt = new NumberFormatter('es_ES',  
                             NumberFormatter::CURRENCY);  
$fmt->formatCurrency(1234, 'JPY');  
// result is 1.234 ¥
```

# ○ ● ● parsing numbers

- `parse($num [, $type [, $pos]])`
- `$type` is `TYPE_DOUBLE` by default

```
$fmt = new NumberFormatter('de_DE',  
                             NumberFormatter::DECIMAL);  
$num = '1.234,567 min';  
$fmt->parse($num);  
// result is 1234.567  
  
$fmt->parse($num, NumberFormatter::TYPE_INT32, $pos);  
// result is 1234, $pos = 9
```



# ○ ● ● parsing currency

- `parseCurrency($num [, $value [, $currency]])`

```
$fmt = new NumberFormatter('es_ES',  
                             NumberFormatter::CURRENCY);  
  
$fmt->parseCurrency('1234 $', $value, $curr);  
// $value = 1234, $curr = 'USD'  
  
$fmt->parseCurrency('1.234 ¥', $value, $curr);  
// $value = 1234, $curr = 'JPY'
```

# ○ ● ● attributes and such

---

- NumberFormatter has many attributes
- Impossible to list all here
- *Plus, it's 2 am and I'm tired*

- Check out pecl/intl **HEAD** for PHP 6
- Check out pecl/intl **PHP\_5\_2** branch for PHP 5
- Requires ICU to be installed

- Documentation for Collator and NumberFormatter is in CVS
- Not integrated into online docs yet

- **Stay tuned, work on other areas continues**

<http://gravitonic.com/talks>

Outspark is hiring:  
engineers, sysadmins  
email [andrei@outspark.com](mailto:andrei@outspark.com)



**kthxbye**

---