Andrei Zmievski
Chief Architect
Outspark, Inc

PHP::$unicode→i18n()

AFUP Forum 2007, Paris

PHP 6 = PHP 5 + Unicode

PHP 5 = PHP 6 – Unicode

Unicode = PHP 6 – PHP 5

What is PHP?

Ha. Ha.

What is Unicode?

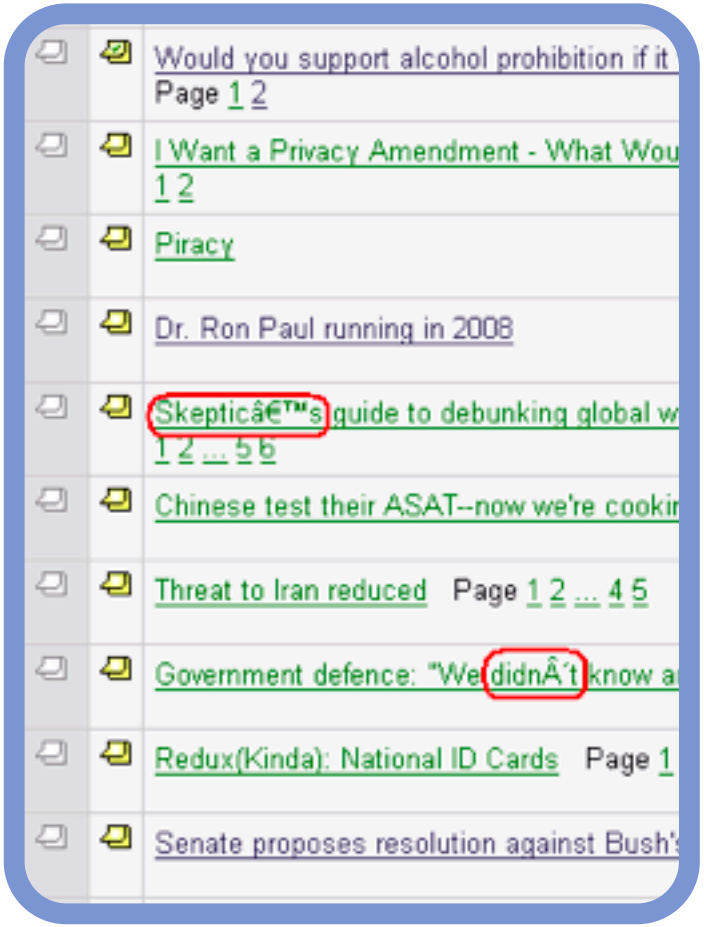
and why do I need?

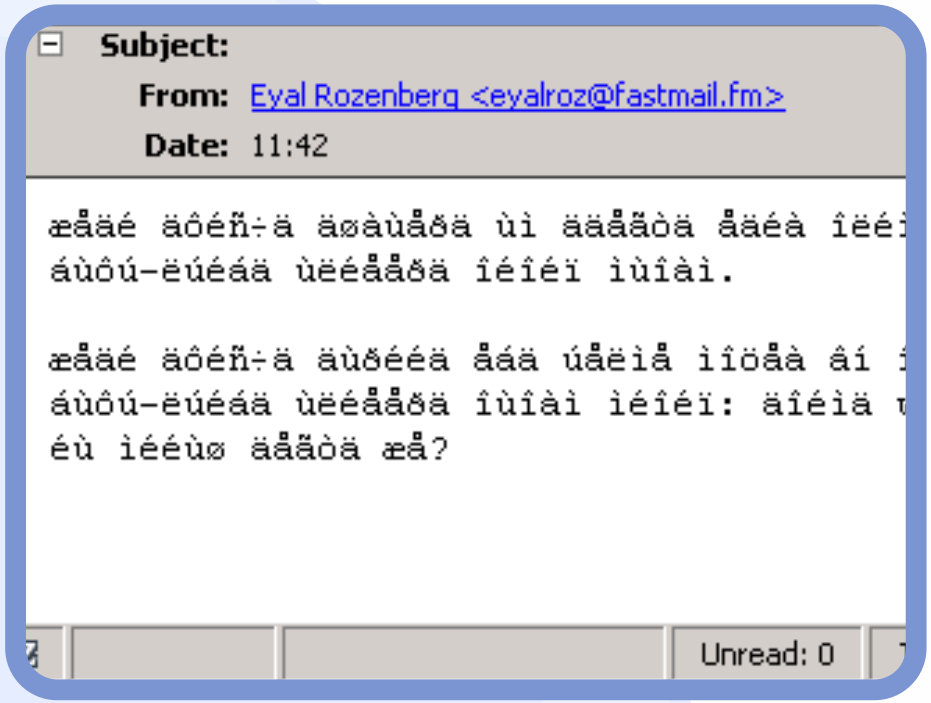
mojibake

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




Subject:
From: Eyal Rozenberg <eyalroz@fastmail.fm>
Date: 11:42
æåäé äôéñ÷ä äøàùåðä ùì äääåãòä åäéà îëéì
áùôú-ëúéáä ùëéååðä îéîéï ìùîàì.
æåäé äôéñ÷ä äùðééä åáä úåëìå ìîöåà âí ì
áùôú-ëúéáä ùëéååðä îùîàì ìéîéï: äîéìä ú
éù ìééùø äåãòä æå?
Unread: 0

mojibake

mojibake

Computers deal with numbers

Encoding soup

1. provinciality
2. computer limitations
3. inertia

Unicode

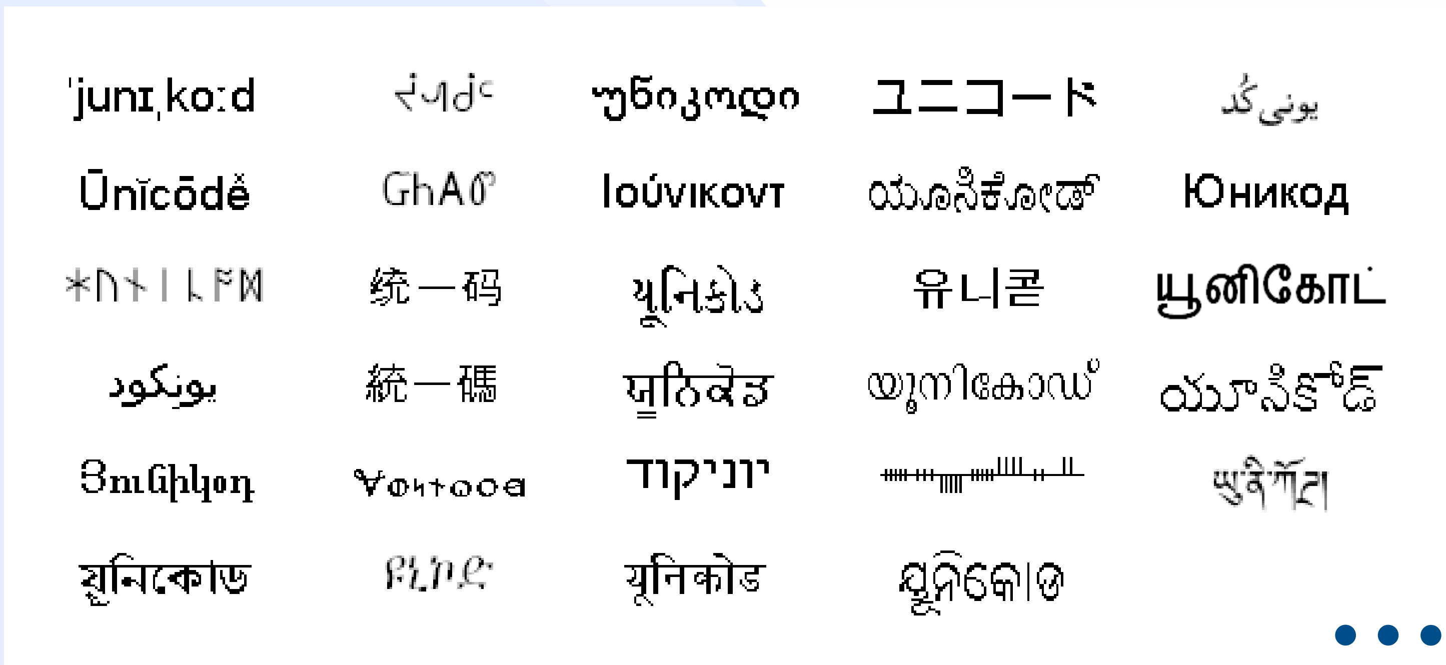
provides a unique number
for every character:

no matter what the platform,

no matter what the program,

no matter what the language.

Unicode


'juniˌkoːd
უნიკოდი
ユニコード
يونى كُد
Ūnĭcōdě
Ιούνικοντ
ಯೂನಿಕೊಡ್
Юникод
统一码
યૂનિકોડ
유니코드
யூனிகோட்
يونیکود
統一碼
ਯੂਨਿਕੋਡ
യൂനികോഡ്
యూనికోడ్
Յունիկօդ
יוניקוד
য়ুনিকোড
यूनिकोड
ୟୁନିକୋଡ

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
- one way to process text
- internet-ready
- extends code life and broadens integration possibilities

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 + ˆ + . = ậ

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

a + ˛ + ˇ = ą̌

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

unicode != i18n

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

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

CLDR

- 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 ∉ Unicode type

string types

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

strings

- All string literals are Unicode

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


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

strings

- 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 ᓱᓴᓐ ᐊᒡᓗᒃᑲᖅ   { ... }
    function சிவாஜி கணேசன்  { ... }
    function འབྲུག་ཡུལ།       { ... }
}

$プロバイダ = array();
$プロバイダ[‘רַעְיולוּחַ שָׁנָה’] = 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("ΣΕΛΛΆΣ");  // result is σελλάς
```

- strip_tags() works on complex text

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

- strrev() preserves combining sequences

```
$u = "Vie\u0302\u0323t Nam";   // Việt Nam
$str = strrev($u);             // result is maN tệiV,
                               // not maN t̂eiV
```

streams

- 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
```

sample

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

```php
$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

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
- Normalization

API

- 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_REVISED@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

- `getDisplayName($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)`

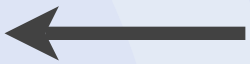
error handling

- `getErrorCode()`
  - returns last error code
- `getErrorMessage()`
  - returns last error message

comparing strings

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

```
$coll = new Collator("fr_CA");
if ($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");
$coll->setAttribute(Collator::CASE_FIRST,
                    Collator::UPPER_FIRST);
if ($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 = new Collator('en_US');
$coll->setAttribute(Collator::ALTERNATE_HANDLING,
                    Collator::SHIFTED);
$coll->setAttribute(Collator::STRENGTH,
                    Collator::TERTIARY);
$coll->sort($strings);
```

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

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

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

more

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

NumberFormatter

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

MessageFormatter

what it is

- produces concatenated messages in a language-neutral way
- operates on *patterns*, which contain *subformats*

what it is

- Need to get:

  Today is November 21, 2007.

- Normal PHP way: `date('F d, Y')`
- MessageFormat would use

  pattern: Today is {0,date}.

  arg: `array(time())`

format types

- time
- date
- number
- choice
- string

instantiating

- `new MessageFormatter($locale, $pattern)`
- `msgfmt_create($locale, $pattern = null)`

formatting

- **format($args)**

```
$pattern = “On {0,date} you have {1,number} meetings.”;
$args = array(time(), 2);
$fmt = new MessageFormatter(‘en_US’, $pattern);
echo $fmt->format($args);

// On November 22, 2007 you have 2 meetings.
```

formatting

- Can use various style modifiers

```
$pattern = “On {0,date,full} you received
            {1,number,#,##0.00} emails.”;
$args = array(time(), 184);
$fmt = new MessageFormatter(‘en_US’, $pattern);
echo $fmt->format($args);

// On Tuesday, November 22, 2007 you received 184.00
emails.
```

formatting

- Trying different locales

```
$fr_pattern = "Aujourd''hui, {0,date,dd MMMM},
               il y a {1,number} personnes sur {3}.";
$fr_args = array(time(), 6579844000.0, 3, "la Terre");

$msg = new MessageFormatter('fr_FR', $fr_pattern);
echo $msg->format($fr_args);

// Aujourd'hui, 22 novembre, il y a 6 579 844 000 personnes
sur la Terre.
```

formatting

Trying different locales

```
$kr_pattern = "오늘 {0,date,long} 행성 {2,number}에
               {1,number,integer}명의 사람 있는다.";
$kr_args = array(time(), 6579844000.0, 3);

$msg = new MessageFormatter('ko_KR', $kr_pattern);
$res = $msg->format($kr_args);
var_dump($res);

// 오늘 2007년 11월 22일 행성 3에6,579,844,000명의 사람 있는다.
```

parsing messages

- Can parse messages according to pattern to extract arguments

parse($message)

```
$pattern = "On {0,date} you have {1,number} meetings.";
$text = "On November 22, 2007 you have 33 meetings.";
$msg = new MessageFormatter('en_US', $pattern);
var_dump($fmt->parse($args));

array(2) {
  [0]=>
  int(1195686000)
  [1]=>
  int(33)
}
```

message resources

- Need management of messages for translation
- Tools:

```
sourceforge.net/projects/rthree

www.php.net/manual/en/ref.gettext.php

ResourceBundle in pecl/intl (upcoming)
```

rthree

- Manage templates and translations
- Sharing layout components
- Compile-time PHP execution

status

- Check out pecl/intl `HEAD` for PHP 6
- Check out pecl/intl `PHP_5_2` branch for PHP 5
- Requires ICU to be installed

docs

- Documentation for implemented classes is in CVS
- Not integrated into online docs yet

future

- Stay tuned, work on other areas continues

Thank You

http://gravitonic.com/talks