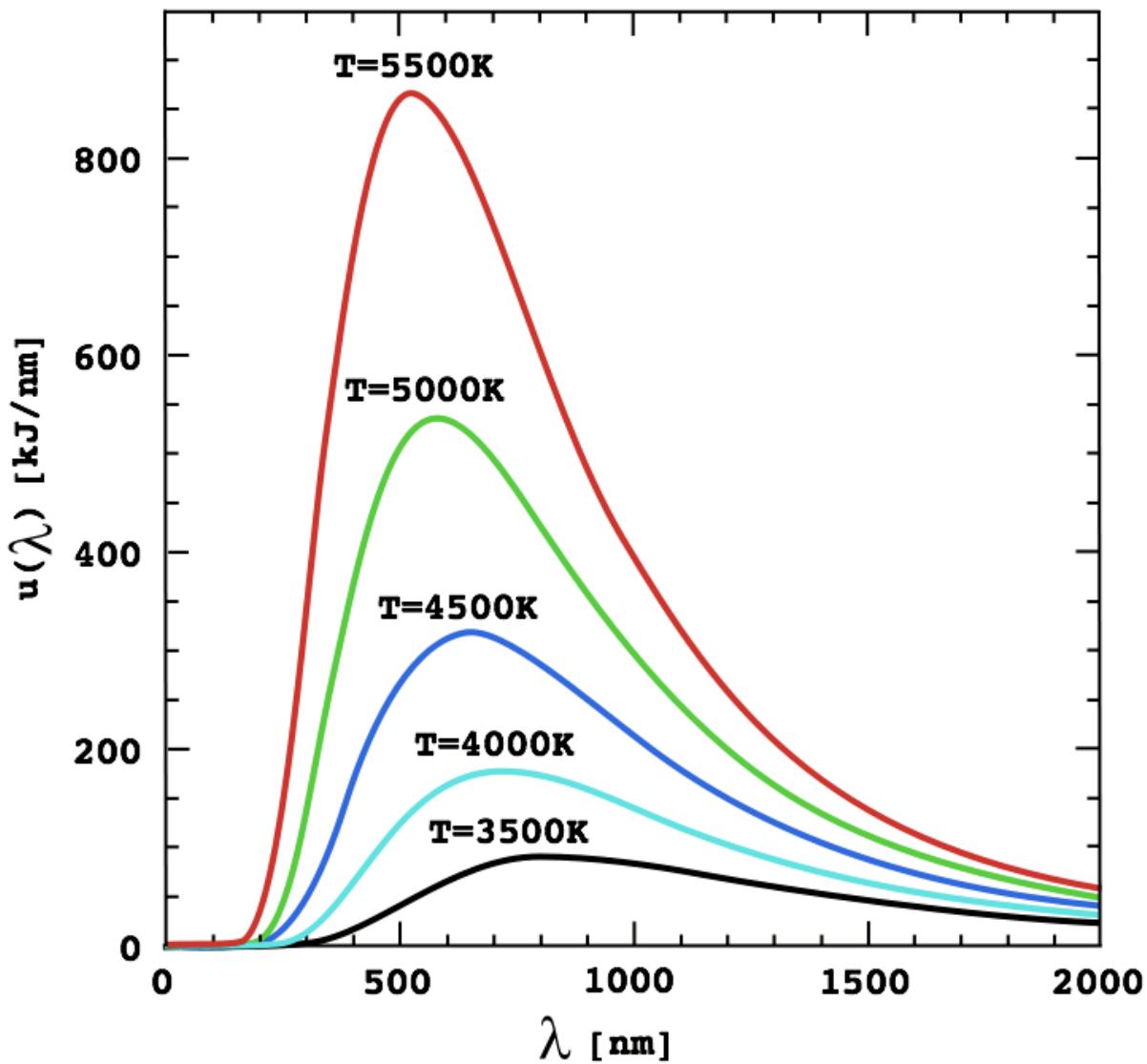




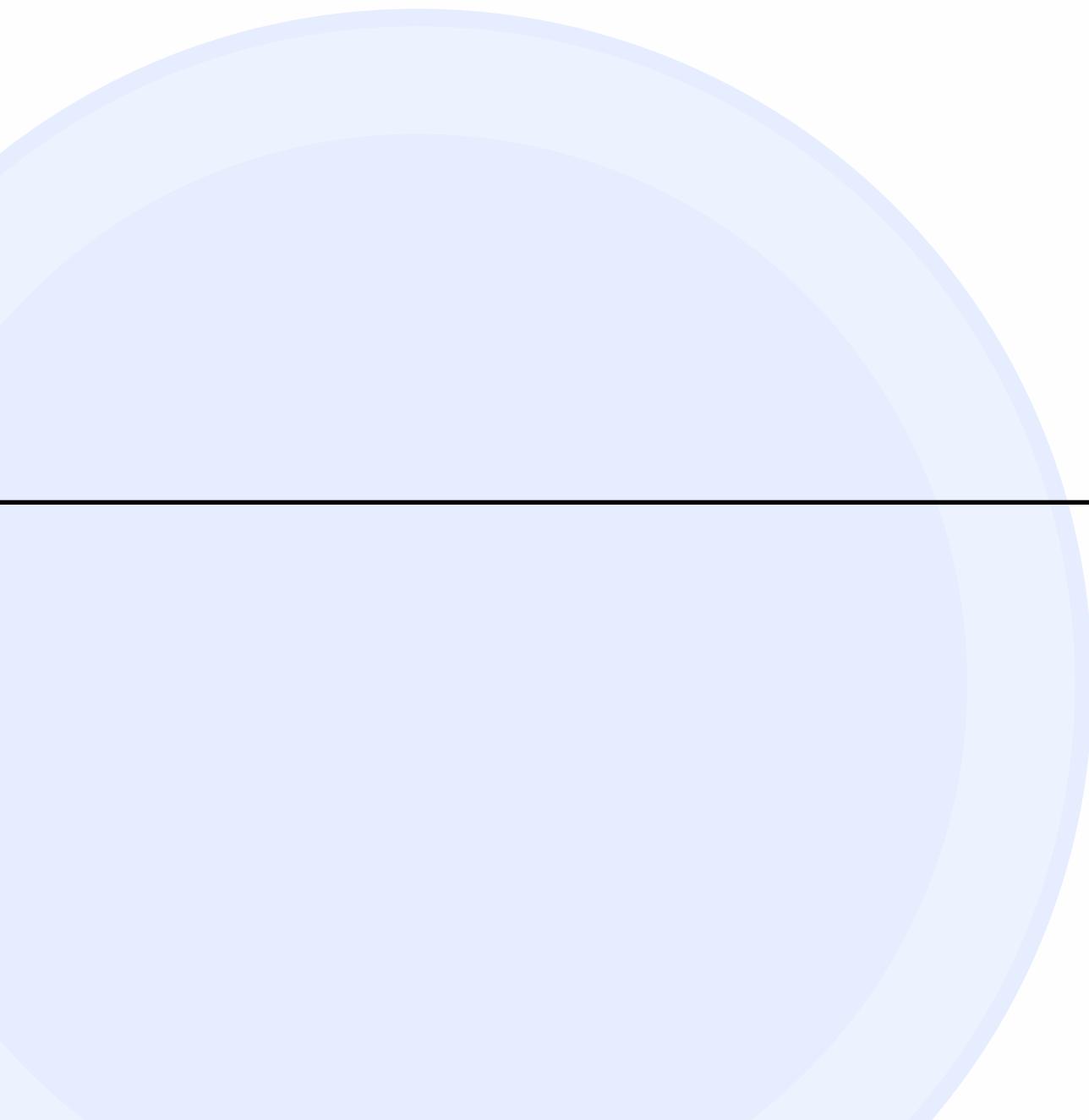
Andrei Zmievski
Chief Architect
Outspark, Inc

PHP::\$unicode → i18n()



$$I(\lambda, T) = \frac{2hc^3}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1}$$







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?

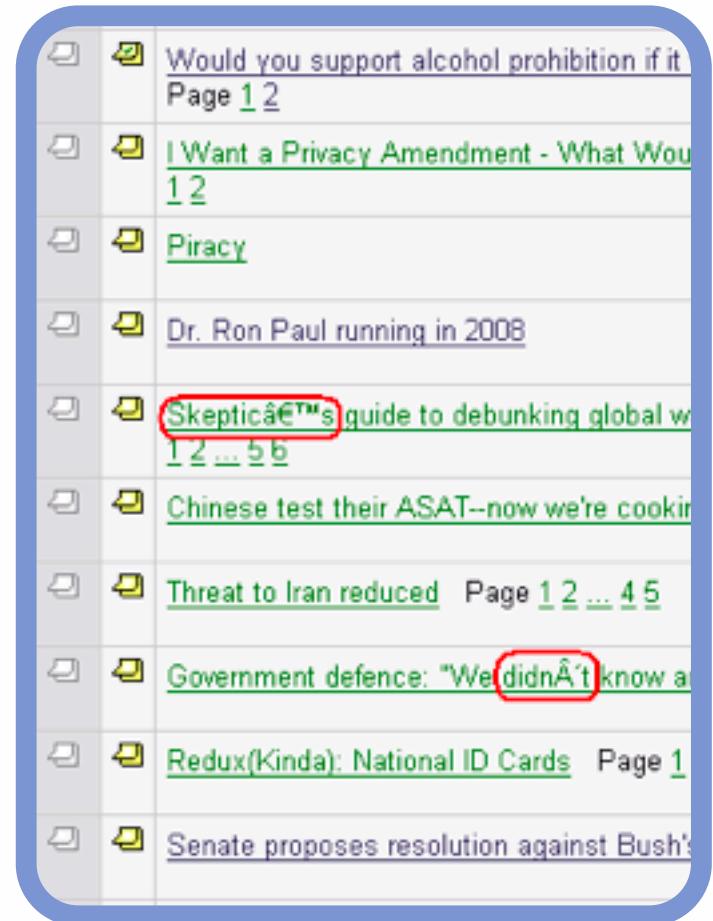


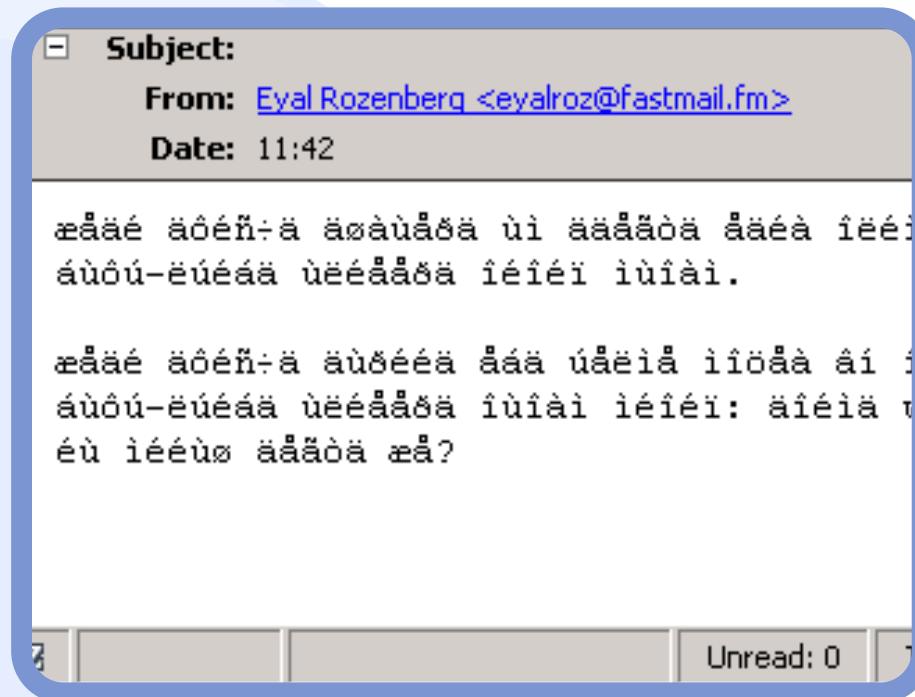
mojibake

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

mojibake



ウィキペディア
フリー百科事典

475-476, 478-479, 480-481

Science

2009-06

a ajsafjaʃl̩ a, aʃfa, =aʃ>aʃ aʃʃa, a, aʃ—a, aʃʃaf² a, a, aʃaʃʃaʃl̩ aʃʃa, aʃʃa— eiʃaʃtai @əsəc ʃe əeə aʃʃaʃʃaʃl̩ aʃʃa, aʃʃa—

ææ-†

äfzäf%äf

ç·é·t

ပုဂ္ဂိုလ်

संस्कृतानि अस्ति एव विद्यन्

æ-tå-åCE-ã]

å†°å...; äf•äfåäf½ç™¾ç§•ä°å...ä€žä,!ä,£ä,äfšäf†ä,£ä,çí¼Wikipediaï¼‰ä€

āl l è̄ cœ̄āl •ālœ̄āl ə̄l „çl ¾è̄±āl ®āl "āl "āl

ç, œ, ë, è, ö

1 ä, »äi« äZÄYä

2 é-téeeé ... c>@

ä »ä¶ äåžŸå»

[C-éšt]

ā, Vāf•āf^ā, lāSā, fā...āf] āf Vāf%ā, lāSā, fāl @āf^āf@āf-āf«āe] æ-þā—ā, ^āf Vāf%
ē] æ Vāl @é] •āl -āl ^āl @á] (EáZÝá) ál -ál ^á] oá cāe

è|| æ ½ã|| ®é|| •ã|| ..

- ç°åll æ,æ-tå—å,åf½åf%å,å,åll oåtåçl tç»åll ®é—"åll §æ-tå—åf tåf½å,å,å,å,åll —

Unicode

provides a unique number
for every character:

no matter what the platform,

no matter what the program,

no matter what the language.



unicode standard

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

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

time formats



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

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

International Components for Unicode

- ✓ Unicode Character Properties
- ✓ Unicode String Class & text processing
- ✓ Text transformations (normalization, upper/lowercase, etc)
- ✓ Text Boundary Analysis (Character/Word/Sentence Break Iterators)
- ✓ Encoding Conversions for 500+ legacy encodings
- ✓ Language-sensitive collation (sorting) and searching
- ✓ Unicode regular expressions
- ✓ Thread-safe
- ✓ Formatting: Date/Time/Numbers/Currency
- ✓ Cultural Calendars & Time Zones
- ✓ (230+) Locale handling
- ✓ Resource Bundles
- ✓ Transliterations (50+ script pairs)
- ✓ Complex Text Layout for Arabic, Hebrew, Indic & Thai
- ✓ International Domain Names and Web addresses
- ✓ Java model for locale-hierarchical resource bundles. Multiple locales can be used at a time

string types



Unicode

- text

- default for literals, etc

Binary

- bytes

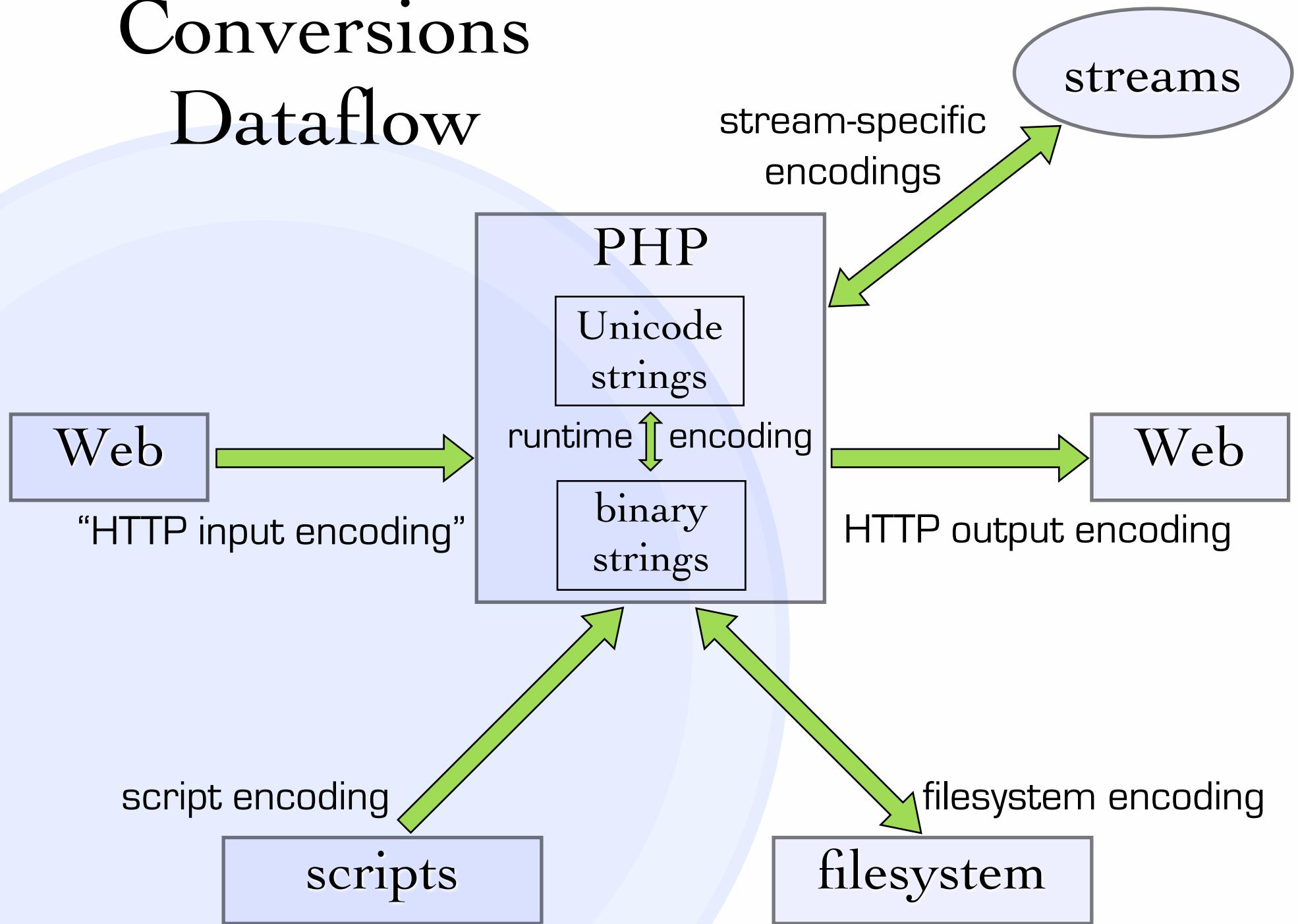
- everything \notin Unicode type

string types



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

Conversions Dataflow



● 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
```

● 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 ફુંક્શન્સ { ... }  
    function சிவாஜி கணேசன் { ... }  
    function ପ୍ରାଣ୍ୟମା { ... }  
}  
  
$provider = array();  
$provider['רַעֲיוֹלָה שְׁנָה'] = 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 = "Viء\u0302\u0323t Nam"; // Việt Nam
$str = strrev($u);           // result is maN t̄eiV,
                            // not maN ūeiV
```

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
```

streams

- If most of the files use an encoding other than UTF-8, change default context

```
stream_default_encoding('Shift-JIS');
$data = file_get_contents('somefile.txt', FILE_TEXT);
// ... work on $data ...
file_put_contents('somefile.txt', $data, FILE_TEXT);
```

- Or create a custom context and use it instead

```
$ctx = stream_context_create(NULL,
                            array('encoding' => 'big5'));
$data = file_get_contents('somefile.txt', FILE_TEXT, $ctx);
// ... work on $data ...
file_put_contents('somefile.txt', $data, FILE_TEXT, $ctx);
```

text iterator



- Use it when iterating over text in a linear fashion (instead of [] operator)
- Iteration over code points, characters, graphemes, words, lines, and sentences forward and backward
- Also provides ICU's boundary analysis API

text iterator

Iterate over code points

```
$text = "nai\u0308ve"; // rendered as naïve
foreach (new TextIterator($text) as $u) {
    var_inspect($u);
}
```

Result

```
unicode(1) "n" { 006e }
unicode(1) "a" { 0061 }
unicode(1) "i" { 0069 }
unicode(1) "\u0308" { 0308 }
unicode(1) "v" { 0076 }
unicode(1) "e" { 0065 }
```

text iterator

Iterate over characters

```
$text = "nai\u308ve"; // rendered as naïve
foreach (new TextIterator($text,
    TextIterator::CHARACTER) as $u) {
    var_inspect($u);
}
```

Result

```
unicode(1) "n" { 006e }
unicode(1) "a" { 0061 }
unicode(2) "\u0308" { 0069 0308 }
unicode(1) "v" { 0076 }
unicode(1) "e" { 0065 }
```

text iterator

Iterate over words in reverse order

```
$text = "Pouvez-vous me dire quelle heure il est ? Merci.";  
foreach (new ReverseTextIterator($text,  
                                TextIterator::WORD) as $u) {  
    if ($u != " ") echo($u), "\n";  
}
```

Result

```
.  
Merci  
?  
est  
il  
heure  
quelle  
dire  
me  
vous  
-  
Pouvez
```

text iterator

Truncate text at a word boundary

```
$it = new TextIterator($text, TextIterator::WORD);
$offset = $it->preceding(40);
echo substr($text, 0, $offset), "...\\n";
```

Get the last 2 sentences of the text

```
$it = new TextIterator($text, TextIterator::SENTENCE);
[it->last();
$offset = $it->previous(2);
echo substr($text, $offset);
```

Get all the pieces delimited by boundaries

```
$it = new TextIterator($text, TextIterator::WORD);
$words = $it->getAll();
```

text transforms



- Powerful and flexible way to process Unicode text
 - script-to-script conversions
 - normalization
 - case mappings and full-/halfwidth conversions
 - accent removal
 - and more
- Allows chained transforms

[`:Latin:]`; `NFKD`; `Lower`; `Latin-Katakana`;

transliteration

```
$names = "  
    김, 국삼  
    김, 명희  
    たけだ, まさゆき  
    おおはら, まなぶ  
    Горбачев, Михаил  
    Козырев, Андрей  
    Καφετζόπουλος, Θεόφιλος  
    Θεοδωράτου, Ελένη  
";  
$r = strtotitle(str_transliterate($names, "Any", "Latin"));
```

```
Gim, Gugsam  
Gim, Myeonghyi  
Takeda, Masayuki  
Oohara, Manabu  
Gorbačev, Mihail  
Kozyrev, Andrej  
Kaphetzópoulos, Theóphilos  
Theodōrátou, Elénē
```

transliteration

- Here's how to get (a fairly reliable) Japanese pronunciation of your name

```
$k = str_transliterate('Britney Spears', 'Latin', 'Katakana');
echo($k), "\n";
$l = strtotitle(str_transliterate($k, 'Katakana', 'Latin')));
echo($l), "\n";
```

ブリテネイ スペアルス
Buritenei Supearusu

- 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

features



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

- 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)



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”



Collator

collation levels



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

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 < 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

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



NumberFormatter



what it is

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



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

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
```



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())`

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 personnes  
sur la Terre.
```

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)  
}
```

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



other things

- APC bundled
- PCRE default regex engine
- “taint” mode
- closures
- traits
- a couple of syntactic sugar treats
- 64-bit integer type
- general cleanup

Im in ur endginn

playin wif ur stringz



Thank You

<http://gravitonic.com/talks>