

PHP 6 and Unicode

The Tower of Babel: Next Generation

Andrei Zmievski
Yahoo! Inc

New York PHP Conference ~ June 16, 2006



List: php-general
Subject: Re: [PHP3] Suggestions for improvement
From: Andrey Zmievski <zmievski () ispi ! net>
Date: 1998-09-30 20:18:29

[long discussion on encrypting scripts]

I had an idea for this that could solve a couple of problems. What if when the php script is first accessed it gets parsed and compiled into a sort of intermediate P-code and that code is what Apache module executes. Then for each access it checks to see if the compiled file exists. If no, then it checks the timestamp on the source and if the timestamp on source file is newer than the one on the compiled file, it recompiles it and then executes it. If it does exist, then it just executes the compiled file. This would cut out the parsing step except for when the file changes, and it would also prevent people from looking at the source code of the script since you can just distribute the compiled file with your applications.

-Andrey

List: php-general
Subject: Re: [PHP3] Suggestions for improvement
From: Zeev Suraski <bourbon () netvision ! net ! il>
Date: 1998-09-30 23:52:23

That's great, only it happens to mean rewriting all of the core of PHP. The timestamp and just in time compiling are the trivial parts here. This might happen in PHP 4.x, but then again, as things look now, PHP 4.x isn't likely to happen within a foreseeable time range, if at all.

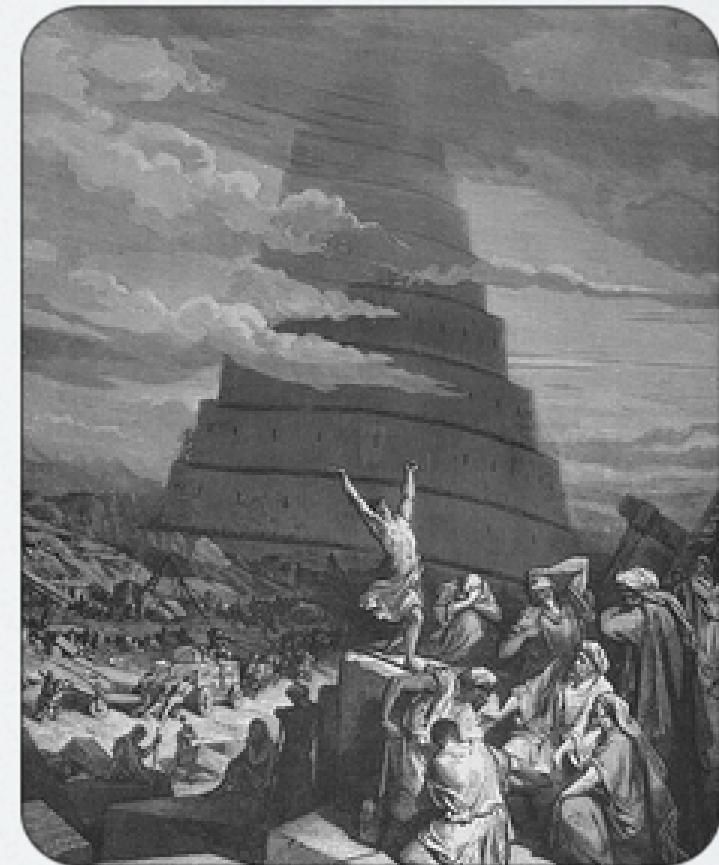
Zeev

Hello!

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Tower of Babel

“Come, let us descend and confuse their language, so that one will not understand the language of his companion”. — Genesis 11:6



Tower of Babel

- ✓ The Babel story rings true
- ✓ Dealing with multiple encodings is a pain
- ✓ Requires different algorithms, conversion, detection, validation, processing...
- ✓ Dealing with multiple languages is a pain too
- ✓ But cannot be avoided in this day and age

Challenges

- ✓ Need to implement applications for multiple languages and cultures
- ✓ Perform language and encoding appropriate searching, sorting, word breaking, etc.
- ✓ Support date, time, number, currency, and more esoteric formatting in the specific locale
- ✓ And much more

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

PHP in the Past

- ✓ PHP has always been a binary processor
- ✓ The string type is byte-oriented and is used for everything from text to images
- ✓ Core language knows little to nothing about encodings and processing multilingual data
- ✓ `iconv` and `mbstring` extensions are not sufficient

PHP in the Past

- ✓ POSIX-based locale support
- ✓ Reliance on the system locale data
- ✓ Disparate i18n functions

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Unicode

- ✓ Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language is

'juni,kō:d	ජුනිකොද	ශ්‍රීලංකා ජුනිකොද	ජුනිකොද	ජුනිකොද
Unicōdē	GhA®	ලොංවිකොව්	යෝනිකේලැය්	ඇනිකොඩ
*统一码	统一码	යුනිකොය	유니콘	යුනිකොට
يونيكود	統一碼	සුතිකොත	ಅුනිකොය	යුනිකොಡ
ආක්ශීල්පි	Unicode	ආක්ශීල්පි	ආක්ශීල්පි	ආක්ශීල්පි
සුනිකොඩ	සුනිකොඩ	සුනිකොඩ	සුනිකොඩ	සුනිකොඩ

Unicode

- ✓ Developed by the Unicode Consortium
- ✓ Designed to allow text and symbols from all languages to be consistently represented and manipulated
- ✓ Covers all major living scripts
- ✓ Version 4.1 has 97,000+ characters
- ✓ Capacity for 1 million+ characters
- ✓ Unicode Character Set = ISO 10646

Unicode Character Set

The primary scripts currently supported by Unicode 4.0 are:

- Arabic
 - Armenian
 - Bengali
 - Bopomofo
 - Buhid
 - Canadian Syllabics
 - Cherokee
 - Cypriot
 - Cyrillic
 - Deseret
 - Devanagari
 - Ethiopic
 - Georgian
 - Gothic
 - Greek
 - Gujarati
 - Han
 - Hangul
 - Hanunóo
 - Hebrew
 - Hiragana
 - Kannada
 - Katakana
 - Khmer
 - Lao
 - Latin
 - Limbu
 - Linear B
 - Malayalam
 - Mongolian
 - Myanmar
 - Ogham
 - Gurmukhi
 - Old Italic (Etruscan)
 - Osmanya
 - Oriya
 - Runic
 - Shavian
 - Sinhala
 - Syriac
 - Tagalog
 - Tagbanwa
 - Tai Le
 - Tamil
 - Telugu
 - Thaana
 - Thai
 - Tibetan
 - Ugaritic
 - Yi

Organized by scripts into blocks

Example Unicode Characters

Unicode Terminology

- ✓ This **abstract character** is U+233B4 不
- ✓ It's **code point** value is 233B4, independent of UTF encoding
- ✓ In UTF-32 the **code unit** is 233B4
- ✓ In UTF-16 it has two 16-bit **code units**:
 - ✓ D84C, DFB4 (high and low surrogates)
- ✓ In UTF-8, it has four 8-bit **code units**:
 - ✓ F0, A3, 8E, B4

Unicode is Generative

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

A + ° = Å

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

a + ^ + . = á

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

a + ć + ˇ = ǎ

U+0061 + U+0322 + U+030C

Unicode is the Future

- ✓ Multilingual
- ✓ Rich and reliable set of character properties
- ✓ Standard encodings: UTF-8, UTF-16, UTF-32
- ✓ Algorithm specifications provide interoperability
- ✓ But Unicode != i18n

Definitions

Internationalization

I18n

Designing and developing an application:

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

Localization

L10n

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

Locales

- ✓ I18N and L10N rely on consistent and correct locale data
- ✓ Locale doesn't refer to data like in POSIX
- ✓ Locale = identifier referring to linguistic and cultural preferences
 - ✓ `en_US`, `en_GB`, `ja_JP`
- ✓ These preferences can change over time due to cultural and political reasons
 - ✓ Introduction of new currencies, like the Euro
 - ✓ Standard sorting of Spanish changes

Types of Locale Data

- ✓ Dates/time formats
- ✓ Number/Currency formats
- ✓ Measurement System
- ✓ Collation Specification
 - ✓ sorting
 - ✓ searching
 - ✓ matching
- ✓ Translated names for language, territory, script, timezones, currencies,...
- ✓ Script and characters used by a language

Common Locale Data Repository

- ✓ Hosted by Unicode Consortium
 - ✓ <http://www.unicode.org/cldr/>
- ✓ Goals:
 - ✓ Common, necessary software locale data for all world languages
 - ✓ Collect and maintain locale data
 - ✓ XML format for effective interchange
 - ✓ Freely available
- ✓ Latest release: June 2, 2005 (CLDR 1.3)
- ✓ 296 locales: 96 languages and 130 territories

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ **PHP in the Future**
- ✓ Current Status

Goals

- ✓ Native Unicode string type
- ✓ Distinct binary string type
- ✓ Unicode string literals
- ✓ Updated language semantics
- ✓ Upgrade existing functions, rather than create new ones

Goals

- ✓ Backwards compatibility
- ✓ Making simple things easy and complex things possible
- ✓ Focus on functionality
- ✓ Parity with Java's Unicode and i18n support

ICU

- ✓ International Components for Unicode
- ✓ Why not our own solution?
 - ✓ Lots of know-how is required
 - ✓ Would be reinventing the wheel
 - ✓ In the spirit of PHP: borrow when possible, invent when needed

Why ICU?

- ✓ It exists
- ✓ Full-featured
- ✓ Robust
- ✓ Fast
- ✓ Proven
- ✓ Portable
- ✓ Extensible
- ✓ Open Source
- ✓ Supported and maintained

ICU Features

- ✓ 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

Major Milestones

- ✓ Retrofitting the engine to support Unicode
- ✓ Making existing extensions Unicode-aware
- ✓ Exposing ICU API

Let There Be Unicode!

- ✓ A control switch called `unicode_semantics`
- ✓ Per-virtual server configuration setting
- ✓ No changes to program behavior unless enabled
- ✓ Does not imply no Unicode at all when disabled!

String Types

- ✓ PHP 4/5 string types
 - ✓ only one, used for everything
- ✓ PHP 6 string types
 - ✓ Unicode: textual data in UTF-16 encoding
 - ✓ Binary: textual data in other encodings and true binary data

String Literals

- With `unicode_semantics=off`, string literals are old-fashioned 8-bit strings
- 1 character = 1 byte

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

```
$jp = "検索オプション"; // UTF-8 encoding
echo strlen($str);      // result is 21
```

Unicode String Literals

- With `unicode_semantics=on`, string literals are of Unicode type
- 1 character may be > 1 byte

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

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

- To obtain length in bytes one would use a separate function

Binary String Literals

- ✓ Binary string literals require new syntax
- ✓ The contents, which are the literal byte sequence inside the delimiters, depend on the encoding of the script

```
// assume script is written in UTF-8

$str = b'woof'; // 77 6F 6F 66

$str = b"q\x{a0}"; // 71 A0 71

$str = b<<<EOD
Ω\x{cf}\x{86}
EOD; // CE A9 CF 82 CF 86
```

Escape Sequences

- Inside Unicode strings `\uXXXX` and `\UXXXXXXXX` escape sequences may be used to specify Unicode code points explicitly

```
// these are equivalent
$str = "Hebrew letter alef: א";
$str = "Hebrew letter alef: \u05D0";

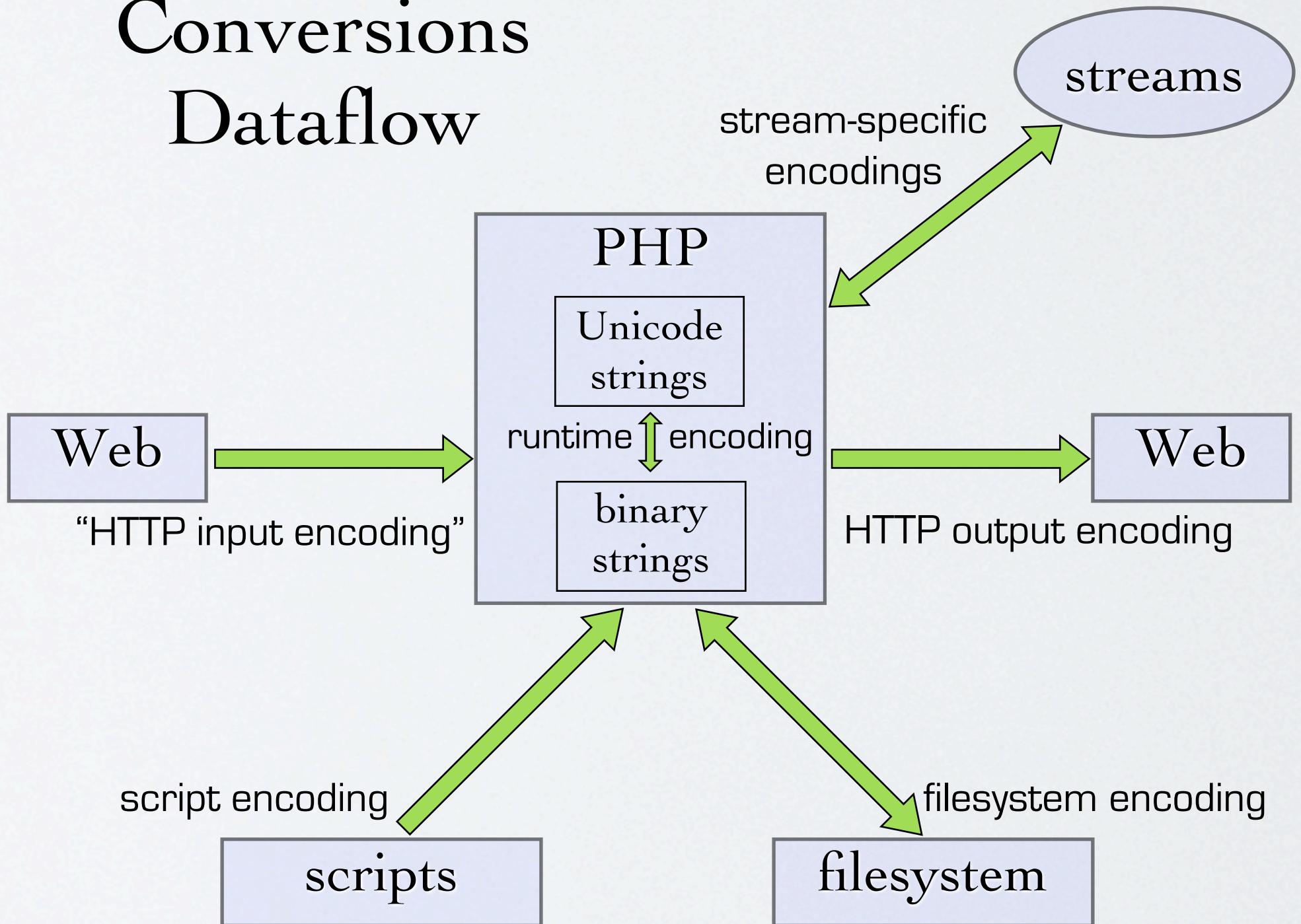
// so are these
$str = 'tetragram: 四';
$str = 'tetragram: \U01D328';
```

Escape Sequences

- Characters can also be specified by name, using the `\C{...}` escape sequence

```
// these are equivalent
$str = "Alef: \C{HEBREW LETTER ALEF}";
$str = "Alef: \u05D0";
```

Conversions Dataflow



Runtime Encoding

- Specifies what encoding to use when converting between Unicode and binary strings at runtime

```
// unicode.runtime_encoding = iso-8859-1

$uni = "Café";           // Unicode
$str = (binary)$str;    // binary ISO-8859-1 string
$uni = (unicode)$uni;   // back to Unicode
```

- Also used when interfacing with functions that do not yet support Unicode type

```
$str = uniqid(); // binary ISO-8859-1 string
```

Script/Source Encoding

- ✓ Currently, scripts may be written in a variety of encodings: ISO-8859-1, Shift-JIS, UTF-8, etc.
- ✓ The engine needs to know the encoding of a script in order to parse it correctly
- ✓ Encoding can be specified as an INI setting or with `declare()` pragma
- ✓ Affects how identifiers and string literals are interpreted

Script Encoding

- ✓ Whatever the encoding of the script, the resulting normal string literals are of Unicode type

```
// unicode.script_encoding = iso-8859-1
$uni = "øl"; // Unicode string
```

```
// unicode.script_encoding = utf-8
$uni = "øl"; // also Unicode string
```

- ✓ In both cases `$uni` is a Unicode string containing two codepoints: U+00F8(ø) and U+006C(l)

Script Encoding

- ✓ Encoding can be also changed with a pragma
- ✓ Has to be the very first statement in the script
- ✓ Does not propagate to included files

```
// unicode.script_encoding = utf-8

declare(encoding="iso-8859-1");
$uni = "øl"; // read as ISO-8859-1 string
              // since declare() overrides
              // INI setting

// the contents of file are read as UTF-8
include "myfile.php";
```

Output Encoding

- ✓ Specifies the encoding for the standard output stream
 - ✓ `echo`, `print`, `var_dump()`, output buffering, etc
- ✓ The script output is transcoded on the fly
- ✓ Does not affect binary strings

```
// unicode.output_encoding = utf-8
// unicode.script_encoding = iso-8859-1

$uni = "øl"; // Unicode string (from ISO-8859-1)
echo $uni; // converts $uni to UTF-8

echo b"øl"; // no conversion, raw contents
```

“HTTP Input Encoding”

- ✓ With Unicode semantics switch enabled, we need to convert HTTP input to Unicode
- ✓ GET requests have no encoding at all and POST ones rarely come marked with the encoding
- ✓ Encoding detection is not reliable
- ✓ Correctly decoding HTTP input is somewhat of an unsolved problem

“HTTP Input Encoding”

- ✓ Browsers are supposed to submit the data in the same encoding as the page the form was on
- ✓ PHP will attempt to decode based on `unicode.output_encoding` setting
- ✓ Could also look at `Accept-Charset` header
- ✓ If the decoding fails, PHP will not populate request arrays at all
- ✓ Applications can ask for incoming data to be decoded again using a different encoding

Filesystem Encoding

- ✓ Specifies the encoding of the file and directory names on the filesystem
- ✓ Filesystem-related functions will do the conversion when accepting and returning filenames

```
// unicode.filename_encoding = utf-8

$dh = opendir("/tmp/musique_de_noël");
while (false !== ($file = readdir($dh))) {
    echo $file, "\n";
}
```

Fallback Encoding

- ✓ This encoding is used when the other encodings do not have assigned values
- ✓ Easy, one-stop configuration
- ✓ Defaults to UTF-8 if not set
- ✓ If the app works only with ISO-8859-1 data:

```
unicode.fallback_encoding = iso-8859-1
```

Type Conversions

- ✓ Unicode and binary string types can be converted to one another explicitly or implicitly
- ✓ Conversions use `unicode.runtime_encoding`
- ✓ Explicit conversions = casting
 - ✓ `(binary)` casts to binary string type
 - ✓ `(unicode)` casts to Unicode string type
 - ✓ `(string)` casts to Unicode type if `unicode_semantics` is on and to binary otherwise

Type Conversions

- ✓ Implicit conversions = concatenation, comparison, parameter passing
- ✓ Prefer to convert to Unicode: more precision

```
// unicode.runtime_encoding = iso-8859-1

$uni = "Côte d'Azur";      // Unicode string
$str = (binary)$uni;       // binary ISO-8859-1 string

$sum = $str . "est là";   // $str is converted to Unicode
                          // and concatenated with literal

if ($str < "soleil")     // $str is converted to Unicode
                          // and compared with the literal

// assume str_word_count() has not been upgraded
$n = str_word_count($sum); // $sum is converted to binary
                           // ISO-8859-1 string
```

Conversion Issues

- ✓ Unicode is a superset of legacy character sets
- ✓ Many Unicode characters cannot be represented in legacy encodings
 - ✓ Hebrew letter **ו** (U+05D8) is not part of ISO-8859-1 character set
- ✓ Strings may also contain corrupt data or irregular byte sequences

Conversion Error Behavior

- ✓ You can customize what PHP should do when it runs into a conversion error
- ✓ Global settings apply to all conversions by default
- ✓ Can be changed with:

```
unicode_set_error_mode(direction, mode)  
unicode_set_subst_char(character)
```

Conversion Error Modes

- ✓ `direction = FROM_UNICODE / TO_UNICODE`
- ✓ Modes can be stop, skip, substitute, or escape (represented by constants)
- ✓ Mode can be OR'ed with a flag specifying that you want PHP to throw an exception on error

```
// unicode.runtime_encoding = iso-8859-1
unicode_set_error_mode(FROM_UNICODE,
    U_CONV_ERROR_SUBST | U_CONV_ERROR_EXCEPTION);
unicode_set_subst_char('*');
$uni = "< 力 >";

try { $str = (binary) $uni; }
catch (UnicodeConversionException $e) { .. }
```

Unicode → Binary Error Modes

```
// unicode.runtime_encoding = iso-8859-1
$str = (binary) "< 力 >";
```

Mode	Result
stop	" "
skip	"< >"
substitute	"< * >"
escape (Unicode)	"< {U+30AB} >"
escape (ICU)	"< %U30AB >"
escape (Java)	"< \u30AB >"
escape (XML decimal)	"< カ >"
escape (XML hex)	"< カ >"

Binary → Unicode Error Modes

```
// unicode.runtime_encoding = utf-8
$str = (unicode)b"< \xe9\xfe >"; // illegal sequence
```

Mode	Result
stop	""
skip	""
substitute	""
escape (Unicode)	"< %XE9%XFE >"
escape (ICU)	"< %XE9%XFE >"
escape (Java)	"< \xE9\xFE >"
escape (XML decimal)	"< éþ >"
escape (XML hex)	"< éþ >"

Conversion Error Modes

- ✓ PHP will always stop on corrupt or irregular data, unless you use one of the escape modes
- ✓ The message you receive will be something like:

```
Warning: Could not convert Unicode string to binary string  
(converter ISO-8859-1 failed on character {U+DC00} at offset 2)
```

- ✓ Or, if going from binary to Unicode:

```
Warning: Could not convert binary string to Unicode string  
(converter UTF-8 failed on bytes (0xE9) at offset 2)
```

Generic Conversions

- ✓ Casting is just a shortcut for converting using runtime encoding
- ✓ For all other encodings, use provided functions

```
$uni = "Yahoo! по-русски"; // means "Yahoo! in Russian"

// $str is binary KOI8-R string
$str = unicode_encode($uni, 'koi8-r', U_CONV_ERROR_SUBST);

// $res should be identical to $uni
$res = unicode_decode($uni, 'koi8-r');
```

Operator Support

- String offset operator works on code points, not bytes!

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

- No need to change existing code if you work only with single-byte encodings, like ASCII or ISO-8859-1

Unicode Identifiers

- ✓ PHP will allow Unicode characters in identifiers
- ✓ You may start with something quite simple and old-fashioned

```
class Component {  
    function process { ... }  
}  
  
$schema = array();  
$schema['part'] = new Component();
```

Unicode Identifiers

- Perhaps you feel that a few accented characters won't hurt

```
class Bâtiment {  
    function créer { ... }  
}  
  
$schématique = array();  
$schématique['premier'] = new Bâtiment();
```

Unicode Identifiers

- Then you learn a couple more languages...
- ...and the fun begins

```
class コンポーネント {  
    function ഫല ലൈബ്രേറി { ... }  
    function சிவாஜி கனேசன் { ... }  
    function దస్తుశాయితా { ... }  
}  
  
$provider = array();  
$provider[ 'ריעולות שגנה' ] = new コンポーネント();
```

Text Iterator

- ✓ Using offset operator `[]` for accessing character in a linear fashion is slow
- ✓ Get used to the idea of using `TextIterator`
- ✓ It is very fast and gives you access to various text units in a generic fashion
- ✓ You can iterate over code units, code points, combining sequences, characters, words, lines, and sentences forward and backward

Text Iterator

Ask for code points

```
$text = "la\u0300-bas"; // really is là-bas
foreach (new TextIterator($text) as $u) {
    var_inspect($u);
}
```

Result

```
unicode(1) "l" { 006c }
unicode(1) "a" { 0061 }
unicode(1) "`" { 0300 }
unicode(1) "-" { 002d }
unicode(1) "b" { 0062 }
unicode(1) "a" { 0061 }
unicode(1) "s" { 0073 }
```

Text Iterator

Ask for characters

```
$text = "la\u0300-bas"; // really is là-bas
foreach (new TextIterator($text,
    TextIterator::CHARACTER) as $u) {
    var_inspect($u);
}
```

Result

```
unicode(1) "l" { 006c }
unicode(2) "à" { 0061 0300 }
unicode(1) "-" { 002d }
unicode(1) "b" { 0062 }
unicode(1) "a" { 0061 }
unicode(1) "s" { 0073 }
```

Text Iterator

Ask for 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

Ask for line break boundaries

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

Result

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

Locales

- ✓ Forget about `setlocale()`
- ✓ Unicode support in PHP relies exclusively on ICU locales
- ✓ Default locale can be accessed with:
 - `locale_set_default()`
 - `locale_get_default()`

Locales

- ✓ ICU locale IDs have somewhat different format from POSIX locale IDS

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

- ✓ Example:

`sr_Latn_YU_REVISED@currency=USD`

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

Collation

- ✓ Collation is the process of ordering units of textual information
- ✓ Specific to a particular locale, language, and document

English:

ABC...RSTUVWXYZ

German:

AÄB...NOÖ...SßTUÜV...YZ

Swedish/Finnish: ABC...RSTUVWXYZÅÄÖ

Collation

- ✓ Languages may sort more than one way
 - ✓ German dictionary vs. phone book
 - ✓ Japanese stroke-radical vs. radical-stroke
 - ✓ Traditional vs. modern Spanish
- ✓ PHP comparison operators do not use collation

```
if ("côte" < "coté") {  
    echo "less\n";  
} else {  
    echo "greater\n"; ←—————  
}
```

Collation

- Let's try collation

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

- We can ignore accents if we want

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

Collation

- ✓ We can sort arrays with collators

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

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

Collation

- ✓ There is a default collator associated with the default locale
- ✓ Can be accessed with:
 - `collator_get_default()`
 - `collator_set_default()`
- ✓ When the default locale is changed, the default collator changes as well

Collation

- ✓ Full collation API is very flexible and customizable
- ✓ You can change collation strength, make it use numeric ordering, ignore or respect case level or punctuation, and much more
- ✓ You can depend on collation algorithm and data to be up-to-date with each version of Unicode

Functions

- ✓ Default distribution of PHP has a few thousand functions
- ✓ All of them need to be analyzed to see whether they need be upgraded to understand Unicode and if so, how
- ✓ Parameter parsing API will perform automatic conversions while upgrades are being done

Functions

- ✓ The upgrade is a continuous process that requires involvement from extension authors
- ✓ Current function upgrade stats are available here:
<http://www.php.net/~scoates/unicode/>

Examples

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

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

- ✓ `strip_tags()` understands Unicode

```
$str = strip_tags("雅<span>虎</span>通"); // result is 雅虎通
```

- ✓ `strrev()` preserves combining sequences

```
$u = "Vi\u0302\u0323t Nam"; // Việt Nam  
$str = strrev($u); // result is maN t\u00e9iV,  
// not maN \u0302teiV
```

Stream I/O

- ✓ PHP has a streams-based I/O system
- ✓ Generalized file, network, data compression, and other operations
- ✓ PHP cannot assume that data on the other end of the stream is in a particular encoding
- ✓ Need to apply encoding conversions

Stream I/O

- ✓ By default, a stream is in binary mode and no encoding conversion is done
- ✓ Applications can convert data explicitly

```
$data = file_get_contents('mydata.txt');
$unidata = unicode_decode($data, 'EUC-JP');
```

- ✓ But ... we're lazy so it's easier to let the streams do it

Stream I/O

- ✓ **t** mode - it's not just for Windows line endings anymore!
- ✓ Uses **encoding** setting in default context, which is UTF-8 unless changed
- ✓ Reading from UTF-8 text file:

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

- ✓ Writing to UTF-8 text file:

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

Stream I/O

- ✓ If you mainly work with files in 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);
```

Stream I/O

- ✓ If you have a stream that was opened in binary mode, you can also automate encoding handling

```
$fp = fopen('http://www.yahoo.com/', 'r');
stream_encoding($fp, 'utf-8');
$data = fgetss($fp, 1024); // reads 1024 Unicode chars
```

- ✓ `fopen()` can actually detect the encoding from the headers, if it's available
- ✓ Streams use global conversion error settings by default, but can be changed with context parameters

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 complex 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
";
$names = strtotitle(str_transliterate($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

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ **PHP in the Future**
- ✓ Current Status

Agenda

- ✓ Tower of Babel
- ✓ PHP in the Past
- ✓ Unicode and Locales
- ✓ PHP in the Future
- ✓ Current Status

Current Status

- ✓ The code is in the public CVS tree
- ✓ Download PHP 6 snapshots today
- ✓ Most of the described functionality has been implemented
- ✓ Development still underway
- ✓ Minor feature tweaks are possible

What's Left?

- ✓ Finish core functionality
- ✓ Upgrade bundled extensions
- ✓ Expose ICU services
- ✓ Update PHP manual
- ✓ Optimize performance
- ✓ Educate, educate, educate

Thank You!



<http://www.gravitonic.com/talks/>