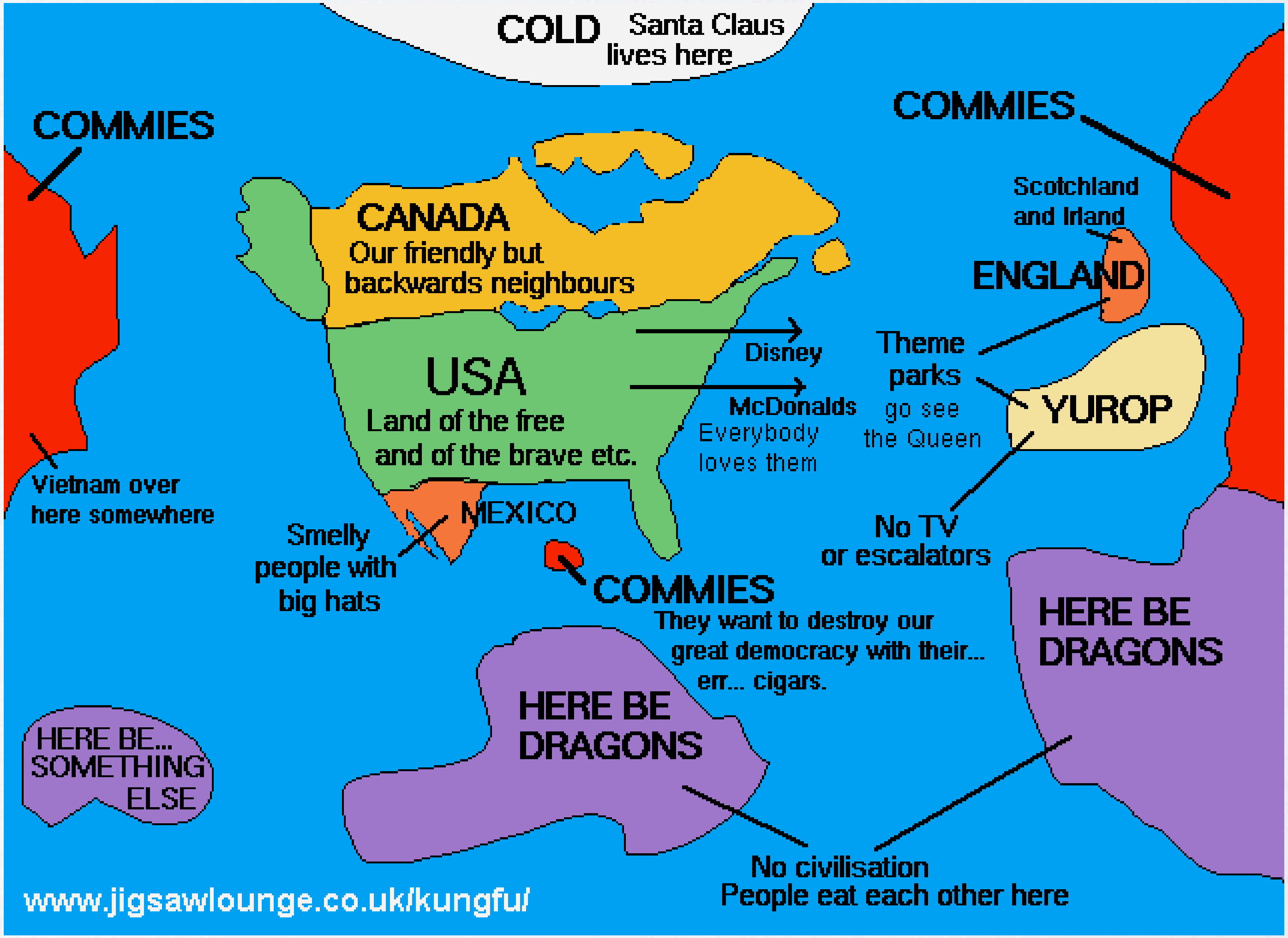
PHP and Unicode: A Love at Fifth Sight

Andrei Zmievski
Yahoo! Inc.

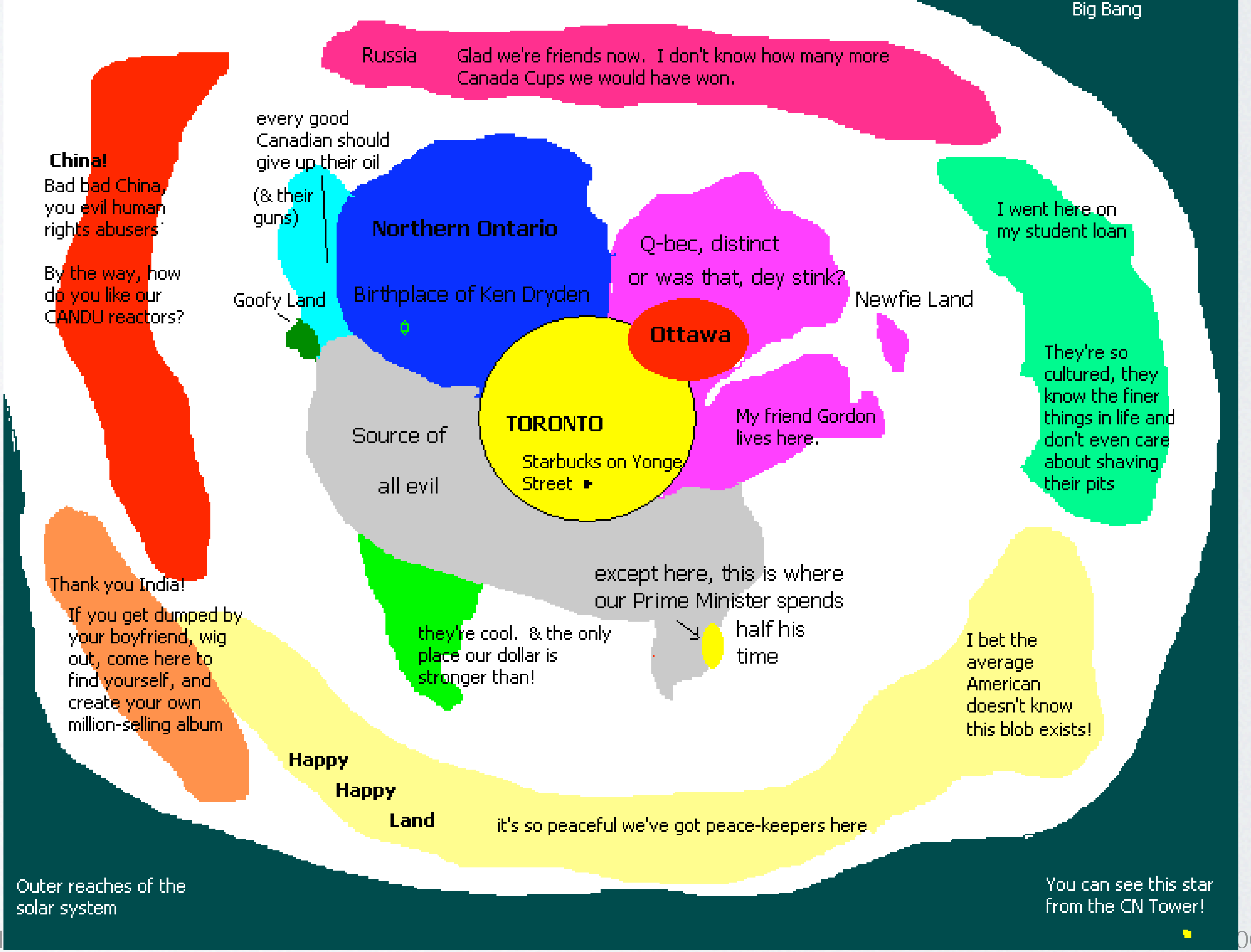
Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

The World According to US


COLD
Santa Claus lives here
COMMIES
COMMIES
Scotchland and Irland
ENGLAND
CANADA
Our friendly but backwards neighbours
USA
Land of the free and of the brave etc.
Disney
McDonalds
Everybody loves them
Theme parks
go see the Queen
YUROP
No TV or escalators
Vietnam over here somewhere
Smelly people with big hats
MEXICO
COMMIES
They want to destroy our great democracy with their... err... cigars.
HERE BE DRAGONS
HERE BE DRAGONS
HERE BE... SOMETHING ELSE
No civilisation
People eat each other here
www.jigsawlounge.co.uk/kungfu/

How a typical (central) Canadian sees the world


Remnants of the Big Bang
Russia
Glad we're friends now. I don't know how many more Canada Cups we would have won.
China!
Bad bad China, you evil human rights abusers
By the way, how do you like our CANDU reactors?
every good Canadian should give up their oil
(& their guns)
Goofy Land
Northern Ontario
Birthplace of Ken Dryden
Q-bec, distinct
or was that, dey stink?
Newfie Land
Ottawa
I went here on my student loan
They're so cultured, they know the finer things in life and don't even care about shaving their pits
TORONTO
Starbucks on Yonge Street
My friend Gordon lives here.
Source of all evil
except here, this is where our Prime Minister spends half his time
they're cool. & the only place our dollar is stronger than!
Thank you India!
If you get dumped by your boyfriend, wig out, come here to find yourself, and create your own million-selling album
I bet the average American doesn't know this blob exists!
Happy Happy Land
it's so peaceful we've got peace-keepers here
Outer reaches of the solar system
You can see this star from the CN Tower!

Multi-i18n-what?

- There is more than one country in the world
- They don't all speak English!
- Some of them even speak French

Multi-i18n-what?

- ✓ **Don't they all use the same alphabet?**

  No.

- ✓ **Well, then each language has a specific digital representation?**

  Guess again.

- ✓ **So it's a big mess?**

  You can't even begin to approximately imagine.

Definitions

Character Set

A collection of abstract characters or graphemes used in a certain domain

…А Б В Г Д Е Ё Ж З И…

Definitions

Character Encoding Form

Representation of a character set using a number of integer codes (code values)

KOI8-R: А = 225, И= 234

CP-1252: А = 192, И = 201

Unicode: А = 410, И = 418

Definitions

Character Encoding Sequence

Representation of code values as bit sequences, with attention given to things like platform-dependent byte order issues

KOI8-R: А = E1, И= EA

CP-1252: А = C0, И = C9

UTF-8: А = D0 90, И = D0 98

UTF-16BE: А = 04 10, И = 04 18

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

Multi-i18n-what?

- ✓ Dealing with multiple encodings is a pain
- ✓ Different algorithms, conversion, detection, validation, processing... understanding
- ✓ 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

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

Can't PHP do it now?

- PHP is a binary processor
- The string type is byte-oriented
- Encoding? What encoding?
- **But isn't it sweet that string vars can contain images?**
- Not if you are trying to do real work!

Ah, I can use *iconv()*

- ✓ Helps with encoding conversion
- ✓ And not much else!
- ✓ You're still stuck with a binary processor
- ✓ And the rest of the baggage: POSIX locales, machine-dependent locale data, inability to mix character sets

Well, *mbstring* sounds good..

- ✓ Automates certain aspects of handling encoding issues (for a subset of them)
- ✓ Tailored to CJK market
- ✓ Not really integrated into the language runtime
- ✓ Fixes a dozen string functions, but..
- ✓ You're <u>still</u> stuck with a binary processor!
- ✓ Lacks collation, search, and other i18n features

Anything else?

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

Hmm, what if there were only one character set, and a couple of sane encodings for all the languages, and well-defined algorithms, and stuff that actually works..

There is.

Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

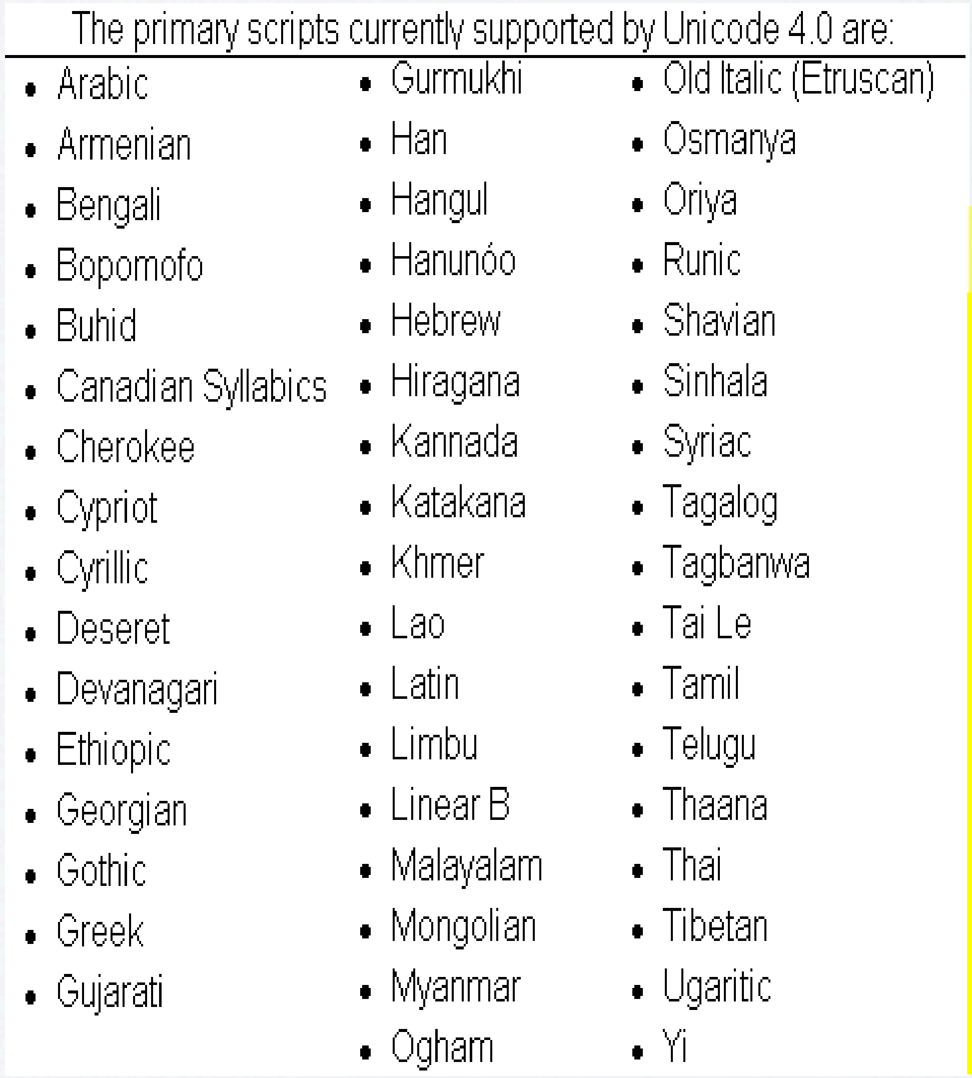
Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

Unicode Overview

- ✓ Developed by the Unicode Consortium
- ✓ Covers all major living scripts
- ✓ Version 4.0 has 96,000+ characters
- ✓ Capacity for 1 million+ characters
- ✓ Unicode Character Set = ISO 10646

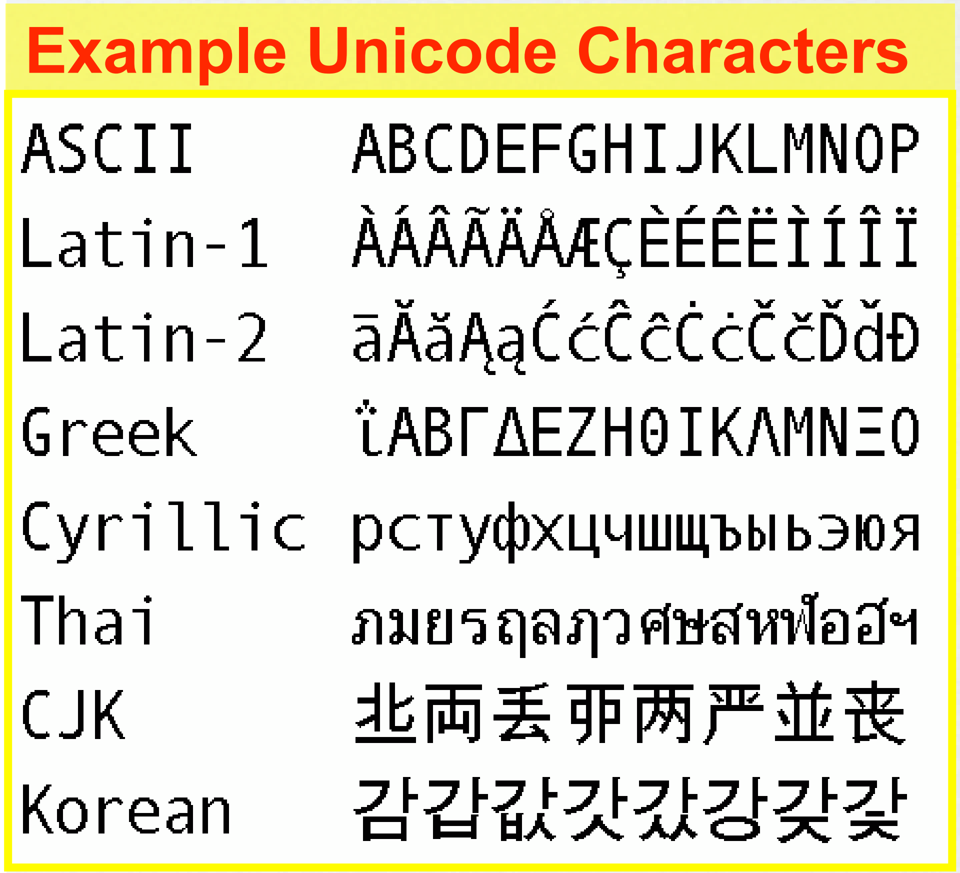
Unicode Character Set



The primary scripts currently supported by Unicode 4.0 are:

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

Organized by scripts into blocks



Example Unicode Characters

ASCII	ABCDEFGHIJKLMNOP
Latin-1	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
Latin-2	āĂăĄąĆćĈĉĊċČčĎďĐ
Greek	ΐΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟ
Cyrillic	рстуфхцчшщъыьэюя
Thai	ภมยรฤลฦวศษสหฬอฮฯ
CJK	丠両丟丣两严並丧
Korean	감갑값갓갔강갖갗

Unicode Character Set

- Code Points 0 to 10FFFF, (Maximum 21 Bits)
  - Unicode notation for code point is U+hhhh
  - 17 Planes of 64K (FFFF) code points
- Basic Multilingual Plane (BMP) U+0000-U+FFFF
  - Commonly used characters in living scripts
- 1st Supplementary Plane (U+10000-U+1FFFF)
  - archaic, fictional characters
- 2nd Supplementary Plane (U+20000-U+2FFFF)
  - Ideographs

Unicode is 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 + . + ˆ = ậ

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

Unicode is Cool

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

Agenda

- Multi-i18n-what?
- Can't PHP do it now?
- Unicode, quoi?
- How do we get it into PHP?
- When can I get my hands on it?

Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

Goals

- ✓ Native Unicode string type
- ✓ Distinct binary and native encoding string types
- ✓ 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

Fundamentals

- ✓ UTF-16 as internal encoding
- ✓ Operational unit is a code point
- ✓ Unicode characters in identifiers

Fundamentals

- ✓ Explicit, rather than implicit, i18n features
- ✓ Normalization form NFC is expected
- ✓ Unicode is a choice, not a requirement
- ✓ Extending language semantics in the Unicode mode is allowed

ICU

- IBM Components for Unicode
- Why not our own solution?
  - Lots of know-how is required
  - Reinventing the wheel
  - In the spirit of PHP: borrow when possible, invent when needed, but solve the problem

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-request configuration setting
- No changes to program behavior unless enabled
- Does not imply no Unicode at all when disabled!

String Types

- Existing string types: only overloaded one, used for everything
- New string types
  - Unicode: textual data (UTF-16 internally)
  - Binary: binary data and strings meant to be processed on the byte level
  - Native: for backwards compatibility and representing strings in a specific encoding

String Literals

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

```
$str = "hello world"; // ASCII string
echo strlen($str);    // result is 11


$jp = "検索オプション"; // UTF-8 string
echo strlen($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
$str = "hello world"; // Unicode
echo strlen($str);    // result is 11


$jp = "検索オプション"; // Unicode
echo strlen($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\xa0q"; // 71 A0 71

$str = b<<<EOD
   Ως\xcf\x86
EOD;                // CE A9 CF 82 CF 86
```

Escape Sequences

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

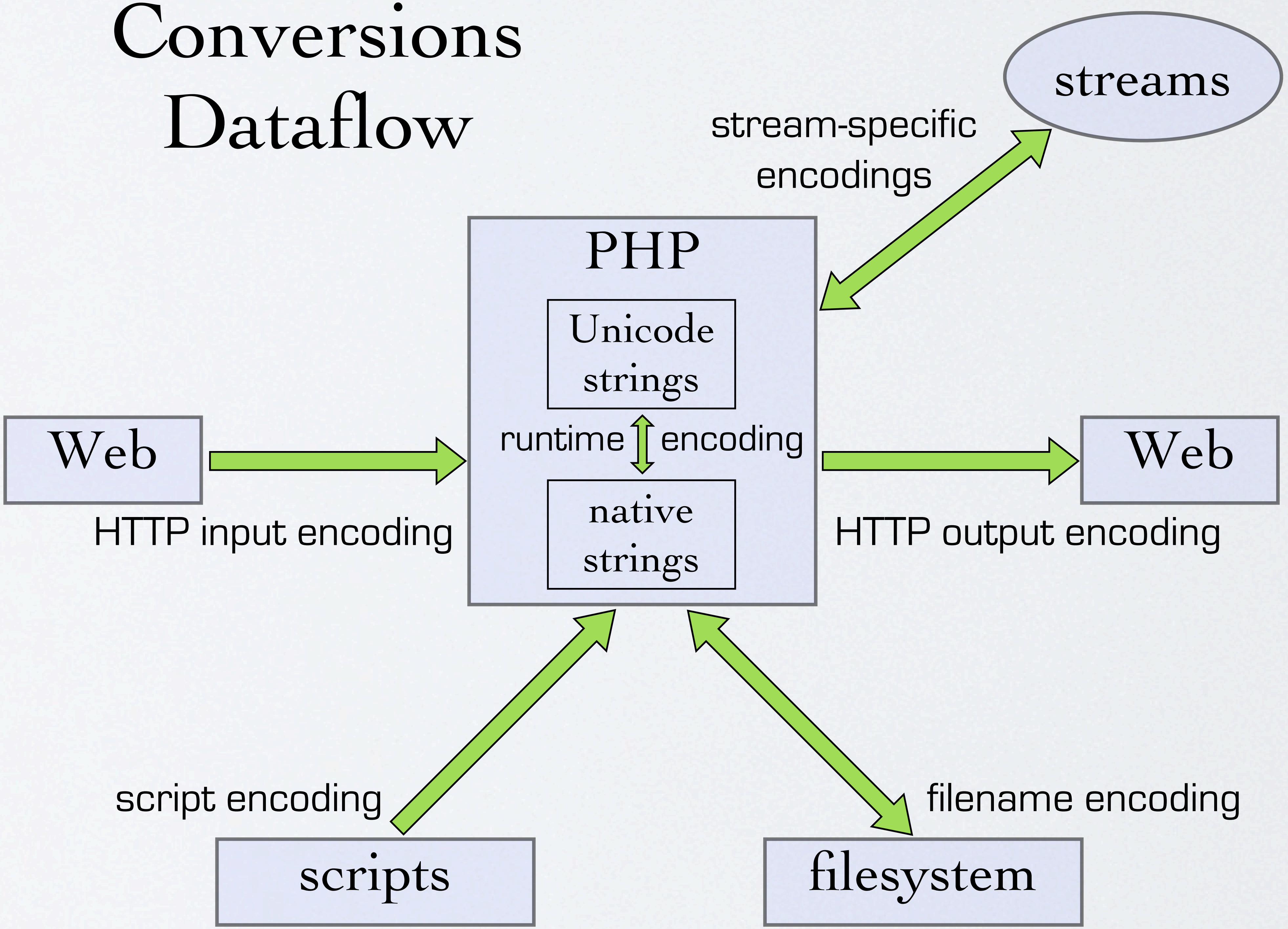
```
// these are equivalent
$str = "Hebrew letter alef: א";
$str = "Hebrew letter alef: \u05D0";

// so are these
$str = 'ideograph: 𠀋';
$str = 'ideograph: \U02000B';
```

Escape Sequences

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

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

Conversions Dataflow



Runtime Encoding

- Specifies what encoding to attach to native strings generated at runtime

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

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

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

```
$str = long2ip(20747599); // $str is ISO-8859-1
```

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
- ✓ 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 string value is of Unicode type

```
// script_encoding = iso-8859-1
$uni = "øl"; // script bytes are F8 6C

// script_encoding = utf-8
$uni = "øl"; // script bytes are C3 B8 6C
```

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

Script Encoding

- ✓ Encoding can be also changed with a pragma
- ✓ Pragma does not propagate to included files

```
// script_encoding = utf-8

declare(encoding="iso-8859-1");
$uni = "øl"; // bytes are F8 6C

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

Output Encoding

- ✓ Specifies the encoding for the standard output stream
- ✓ The script output is transcoded on the fly
- ✓ Does not affect binary strings

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

$uni = "øl"; // input bytes are  F8 6C
echo $uni;   // output bytes are C3 B8 6C

echo b"øl";  // output bytes are F8 6C
```

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
- If the incoming encoding is not found, PHP can use the `http_input_encoding` setting to decode the data

HTTP Input Encoding

- Frequently incoming data is in the same encoding as the page it was submitted from
- Applications can ask for incoming data to be decoded again using a different encoding

Filename Encoding

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

```
// filename_encoding = utf-8

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

Fallback Encoding

- The 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-8859-2 data:

```
fallback_encoding = iso-8859-2
```

Type Conversions

from \ to	Native	Unicode	Binary

Native	—	implicit=yes
		explicit=yes
			implicit=no
			explicit=yes

Unicode	implicit=no
	explicit=yes
		—	implicit=no
			explicit=yes

Binary	implicit=no
	explicit=no
		implicit=no
		explicit=no
			—

implicit = concatenation, e.g.
explicit = casting

Conversion Issues

- ✓ Not all characters can be converted between Unicode and legacy encodings
- ✓ PHP will always attempt to convert as much of the data as possible
- ✓ The severity of the error issued by PHP depends on the type of the encountered problem
- ✓ The conversion error behavior is customizable

Operator Support

- Concatenating a native string with a Unicode one requires up-converting it to Unicode

```
$str = foo();          // foo() returns a native string
$uni = "def";          // Unicode string
$res = $str . $uni; // result is Unicode
```

- Binary type cannot be concatenated with other types

```
$res = b"abc" . "新着情報";           // runtime error!

$res = b"abc" . b"新着情報";          // OK

$res = b"abc" . (binary)"新着情報"; // OK, but different result
```

Operator Support

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

```
$str = "大学";    // bytes are e5 a4 a7  e5 ad a6
echo $str{1};    // result is 学
$str{0} = 'サ'; // string is now サ学
                 // bytes are e3 82 b5  e5 ad a6
```

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

Arrays

- All three string types can be used as keys
- The `unicode_semantics` switch affects how lookup is done
  - With `unicode_semantics=on`, native “`abc`” and Unicode “`abc`” are equivalent for hash lookup purposes
  - With `unicode_semantics=off`, they are distinct

Inline HTML

- PHP scripts are very frequently interspersed with HTML blocks
- These blocks should be in the same encoding as the PHP blocks
- Transcode them to output encoding as necessary

Functions

- Default distribution of PHP has a few thousand functions
- Most of them use parameter parsing API that accepts typed parameters
- The upgrade process can be alleviated by adjusting this API to perform automatic conversions

Functions

- The upgrade will be a continuous process that will require involvement from extension authors
- All functions should be analyzed to determine their semantics as applied to Unicode strings
- A set of guidelines is essential

Guidelines

- ✓ No drastic changes to behavior of existing functions
- ✓ Search/comparison functions work in binary mode
- ✓ Case-insensitive functions use simple case mapping

Guidelines

- ✓ Combining sequences do not influence matching
- ✓ Formatting functions do not use ICU API

Example

- By default, compare on a codepoint level using simple case mapping

```
if (strcasecmp($a, $b) == 0) {
   ...
}
```

- If proper collation is desired, use ICU API

```
$coll = new Collator("fr_FR@collation=phonebook", ...);
$coll->setAttribute(UCOL_STRENGTH, UCOL_SECONDARY);
if ($coll->compare($a, $b) == 0) {
  ...
}
```

Stream IO

- PHP has a streams-based I/O system
- Generalized file, network, data compression, and other operations
- Streams will be in binary mode by default

Stream IO

- Applications can manage Unicode conversion explicitly

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

- Or apply a conversion filter to the stream

```
$fp = fopen($file, 'r');
stream_filter_append($fp, 'unicode.from.euc-jp');
// reads EUC-JP data and converts to Unicode
$data = fread($fp, 1024);
```

Stream IO

- Bad Unicode write! Bad!

```
$fp = fopen('somefile.txt', 'w');
fwrite($fp, "\u0123foo bar baz\u0456");
```

- Good Unicode writes! Good! ☺

```
$fp = fopen('somefile.txt', 'w');
stream_filter_append($fp, 'unicode.to.utf8');
fwrite($fp, "\u0123foo bar baz\u0456");
```

```
$fp = fopen('somefile.txt', 'wt');
fwrite($fp, "\u0123foo bar baz\u0456");
```

Stream IO

- Overriding default output encoding for streams

```
$ctx = stream_context_get_default();
stream_context_set_params(array('output_encoding'=>'latin1'));
$fp = fopen('somefile.txt', 'wt');
fwrite($fp, "\u0123foo bar baz\u0456");
```

Unicode Identifiers

- PHP will allow Unicode characters in identifiers
- Can have ideographic characters in addition to accented ones

```
class コンポーネント {
    function コミット { ... }
}

$プロバイダ = array();
$プロバイダ['רַעְיולוּחַ שָׁנָה'] = new コンポーネント();
```

Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

Agenda

- ✓ Multi-i18n-what?
- ✓ Can't PHP do it now?
- ✓ Unicode, quoi?
- ✓ How do we get it into PHP?
- ✓ When can I get my hands on it?

When can I have it?

- ✓ In a while
- ✓ In a longish while
- ✓ 90% of described functionality is done
- ✓ Merge into public tree imminent

When can I have it?

- ✓ Document new API and migration guidelines
- ✓ Upgrade core extensions to support Unicode
- ✓ Expose ICU services
- ✓ Optimize performance
- ✓ Educate, educate, educate

Thank You!

LIFE ENGINE

Download the slides at:
http://www.gravitonic.com/talks

Functions

- We can ease the transition for extension authors
- If a Unicode string is passed to a function expecting a legacy string, the engine will attempt to convert it to the runtime encoding
- The inverse happens for functions that are passed a legacy string when they require a Unicode one

Functions

- Many Unicode operations may require additional context
- Upgraded functions will use the most common mode of operation, and leave the edge cases to ICU API
- Consider `strcasecmp()`