PHP Wrap-up

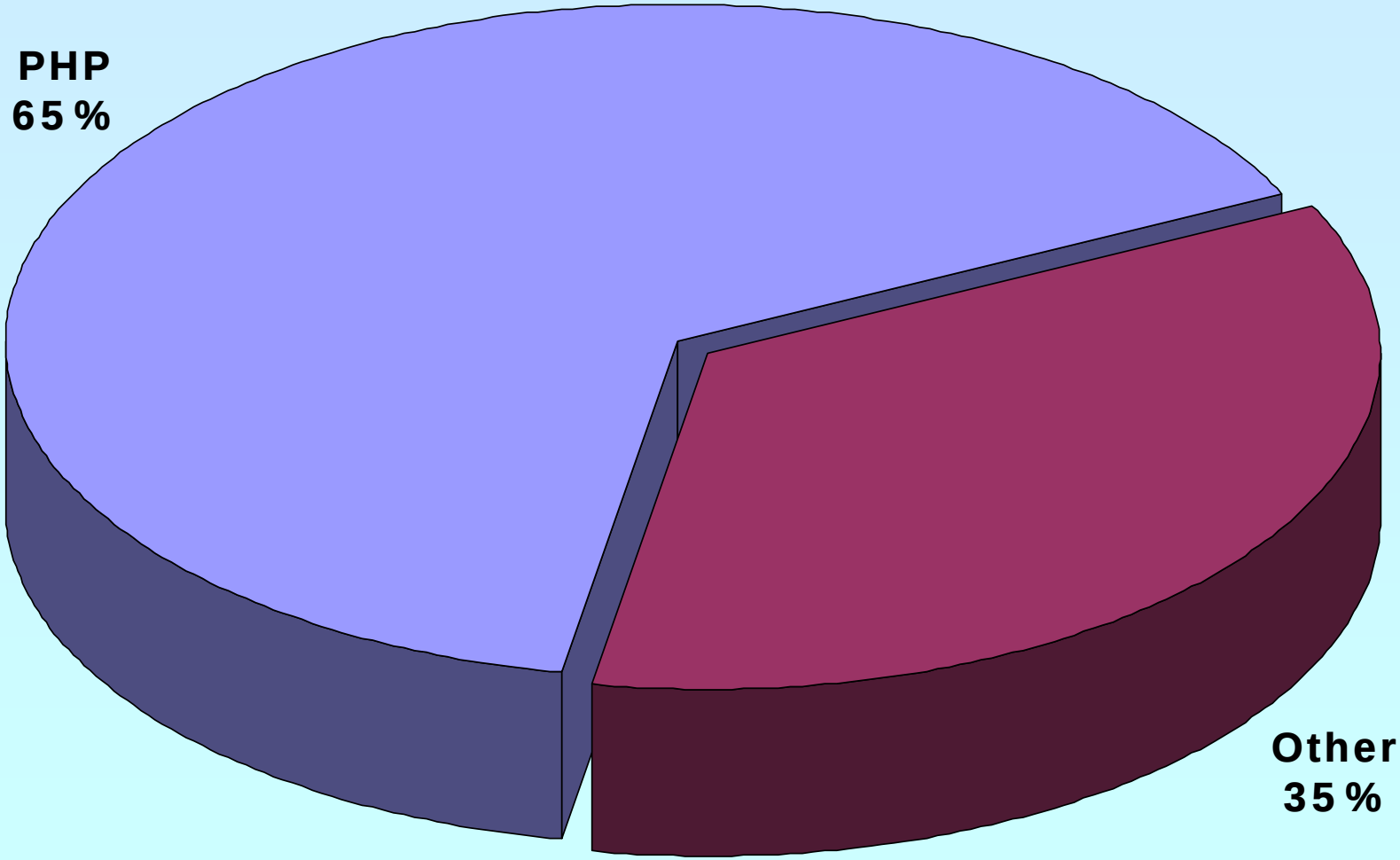
Marcus Börger

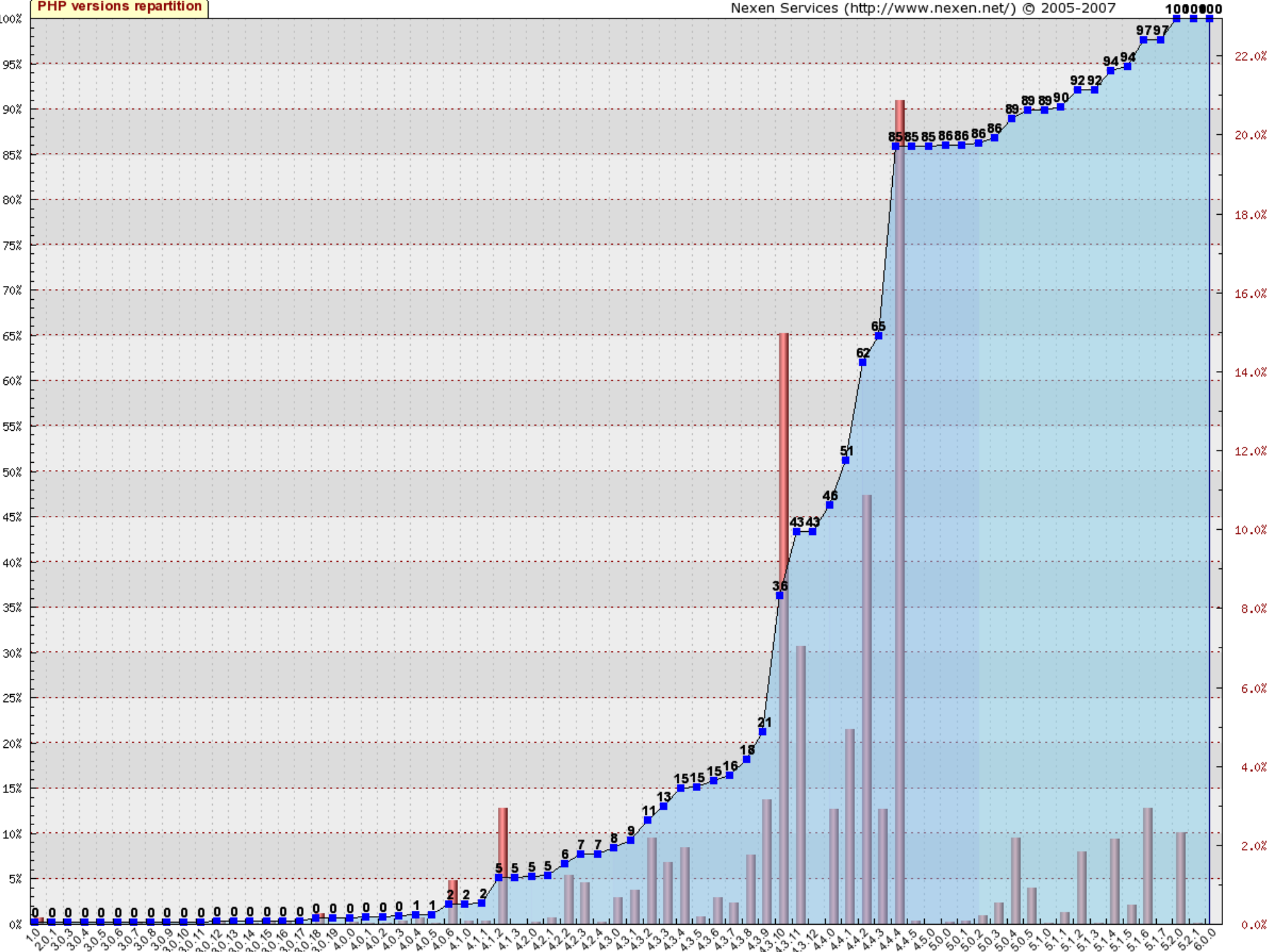
Mexico CONSOL 2007

Welcome

- ☑ Some statistics
- ☑ Revisiting PHP 4
- ☑ Looking at PHP 5.0, 5.1
- ☑ Upgrading to new PHP 5.2
- ☑ A glimpse at the future PHP 6

PHP usage by IP




PHP versions repartition
Nexen Services (http://www.nexen.net/) © 2005-2007

27 Dec 2002: PHP 4.3

- ☑ Do not use
  - ☑ Insecure
  - ☑ Unsupported
  - ☑ No longer continued

11 Jul 2005: PHP 4.4

- ☑ Fixes a memory corruption with references
- ☑ Only security fixes
- ☑ No new features at all
- ☑ Available for quite some time
- ☑ PHP 4.4.5 released 14th February 2007

- ☑ No PHP 4.5 will come out

PHP 4 and OOP ?

- Poor Object model
  - ☑ Methods
    - ☒ No visibility
    - ☒ No abstracts, No final
    - ☒ Static without declaration
  - ☑ Properties
    - ☒ No static properties
    - ☒ No constants
  - ☑ Inheritance
    - ☒ No abstract, final inheritance, no interfaces
    - ☒ No prototype checking, no types
  - ☑ Object handling
    - ☒ Copied by value
    - ☒ No destructors

13 Jul 2004: PHP 5.0

- ☑ Completely rewritten Zend Engine 2.0
- ☑ First version with the new object model
- ☑ Libxml2 based XML replaces Sablotron
- ☑ New extensions
  - ☑ DOM, MySQLi, PDO, SimpleXML, SPL
- ☑ Unbundled mysql client library

- ☑ Sometimes called the "alpha" version
- ☑ Discontinued

Objects referenced by ID

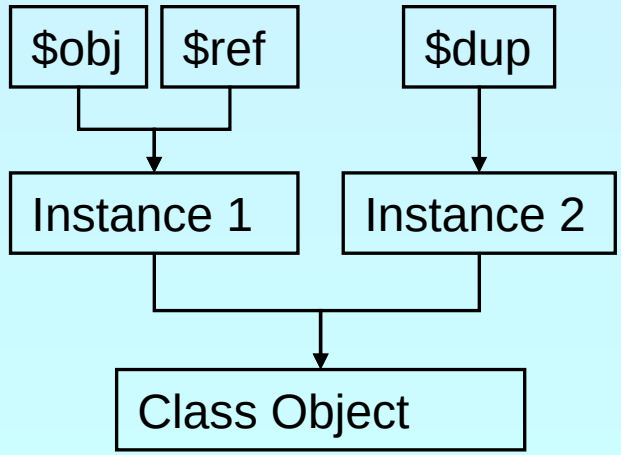
- ☑ Objects are no longer somewhat special arrays
- ☑ Objects are no longer copied by default
- ☑ Objects may be copied using clone/__clone()

```
class Object {};

$obj = new Object();

$ref = $obj;

$dup = clone $obj;
```



ZE2's revamped object model

- ☑ Objects are referenced by identifiers
- ☑ Constructors and Destructors
- ☑ Static members
- ☑ Default property values
- ☑ Constants
- ☑ Visibility
- ☑ Interfaces
- ☑ Final and abstract members
- ☑ Interceptors
- ☑ Exceptions
- ☑ Reflection API
- ☑ Iterators

Revamped Object Model

- ☑ PHP 5 has really good OOP support
  - ☑ Better code reuse
  - ☑ Better for team development
  - ☑ Easier to refactor
  - ☑ Some patterns lead to much more efficient code
  - ☑ Fits better in marketing scenarios

From engine overloading . . .

- ☑ Zend engine 2.0+ allows to overload the following

  - ☑ by implementing interfaces
    - ☑ Foreach by implementing `Iterator`, `IteratorAggregate`
    - ☑ Array access by implementing `ArrayAccess`
    - ☑ Serializing by implementing `Serializable`

  - ☑ by providing magic functions
    - ☑ Function invocation by method `__call()`
    - ☑ Property access by methods `__get()` and `__set()`
    - ☑ Automatic loading of classes by function `__autoload()`

. . . to SPL

It is easy in a complex way

- Lukas Smith
php conference 2004

- ☑ A collection of standard interfaces and classes
  - Most of which based around engine overloading
- ☑ A few helper functions

What is SPL about & what for

- ☑ Captures some common patterns
- ☑ Advanced Iterators
- ☑ Functional programming
- ☑ File and directory handling
- ☑ Makes __autoload() useable
- ☑ Exception hierarchy with documented semantics

Exceptions

- ☑ Respect these rules
  1. Exceptions are exceptions
  2. Never use exceptions for control flow
  3. Never ever use exceptions for parameter passing

```php
<?php
try {
    // your code
    throw new Exception();
}
catch (Exception $e) {
    // exception handling
}
?>
```

Practical use of exceptions

- ☑ Constructor failure
- ☑ Converting errors/warnings to exceptions
- ☑ Simplify error handling
- ☑ Provide additional error information by tagging

Constructor failure

- ☑ In PHP 4.4 you would simply `unset($this)`
- ☑ Provide a param that receives the error condition

```
<?php
class Object
{
   function __construct( &$failure)
   {
      $failure = true;
   }
}
$error = false;
$o = new Object($error);
if (!$error) {
   // error handling, NOTE: the object was constructed
   unset($o);
}
?>
```

Constructor failure

- ☑ In 5 constructors do not return the created object
- ☑ Exceptions allow to handle failed constructors

```php
<?php
class Object
{
   function __construct()
   {
      throw new Exception;
   }
}
try {
   $o = new Object;
}
catch (Exception $e) {
   echo "Object could not be instantiated\n";
}
?>
```

Simplify error handling

- ☑ Typical database access code contains lots of if's

```
<html><body>
<?php
$ok = false;
$db = new PDO('CONNECTION');
if ($db) {
   $res = $db->query('SELECT data');
   if ($res) {
      $res2 = $db->query('SELECT other');
      if ($res2) {
         // handle data
         $ok = true; // only if all went ok
      }
   }
}
if ($ok) echo '<h1>Service currently unabvailable</h1>';
?>
</body></html>
```

Simplify error handling

- ☑ Trade code simplicity with a new complexity

```
<html><body>
<?php
try {
   $db = new PDO('CONNECTION');
   $db->setAttribute(PDO::ATTR_ERRMODE,
                                 PDO::ERRMODE_EXCEPTION);
   $res = $db->query('SELECT data');
   $res2 = $db->query('SELECT other');
   // handle data
}
catch (Exception $e) {
   echo '<h1>Service currently unabvailable</h1>';
   error_log($e->getMessage());
}
?>
</body></html>
```

What are Iterators

- ☑ Iterators are a concept to iterate anything that contains other things.
- ☑ Iterators allow to encapsulate algorithms

What are Iterators

- ☑ Iterators are a concept to iterate anything that contains other things. Examples:
  - ☑ Values and Keys in an array — `ArrayObject`, `ArrayIterator`
  - ☑ Text lines in a file — `SplFileObject`
  - ☑ Files in a directory — [`Recursive`]`DirectoryIterator`
  - ☑ XML Elements or Attributes — ext: SimpleXML, DOM
  - ☑ Database query results — ext: PDO, SQLite, MySQLi
  - ☑ Dates in a calendar range — PECL/date (?)
  - ☑ Bits in an image — ?
- ☑ Iterators allow to encapsulate algorithms

What are Iterators

- Iterators are a concept to iterate anything that contains other things. Examples:
  - Values and Keys in an array — `ArrayObject`, `ArrayIterator`
  - Text lines in a file — `SplFileObject`
  - Files in a directory — `[Recursive]DirectoryIterator`
  - XML Elements or Attributes — ext: SimpleXML, DOM
  - Database query results — ext: PDO, SQLite, MySQLi
  - Dates in a calendar range — PECL/date (?)
  - Bits in an image — ?
- Iterators allow to encapsulate algorithms
  - Classes and Interfaces provided by SPL:

    `AppendIterator`, `CachingIterator`, `LimitIterator`, `FilterIterator`, `EmptyIterator`, `InfiniteIterator`, `NoRewindIterator`, `OuterIterator`, `ParentIterator`, `RecursiveIterator`, `RecursiveIteratorIterator`, `SeekableIterator`, `SplFileObject`, . . .

Array vs. Iterator

- An array in PHP — `$ar = array()`
  - can be rewound: `reset($ar)`
  - is valid unless it's key is NULL: `!is_null(key($ar))`
  - have current values: `current($ar)`
  - have keys: `key($ar)`
  - can be forwarded: `next($ar)`

- Something that is traversable — `$it = new Iterator;`
  - **may** know how to be rewound: `$it->rewind()`
    (does not return the element)
  - should know if there is a value: `$it->valid()`
  - **may** have a current value: `$it->current()`
  - **may** have a key: `$it->key()`
    (may return NULL at any time)
  - can forward to its next element: `$it->next()`

24 Nov 2005: PHP 5.1

- ☑ New extension XMLReader
- ☑ Added new date/time handling
- ☑ Add streams filters bz2 and zlib
- ☑ Improved interactive mode php –a
- ☑ Major changes/improvements to MySQLi and SPL
- ☑ Since 5.1.6 `var` is an alias for `public`

- ☑ Sometimes called 'beta' version
- ☑ Discontinued

02 Nov 2006: PHP 5.2

- ☑ Current stable version
- ☑ 5.2.1 released on 9th February
- ☑ 5.2.2+ only small feature changes and additions
- ☑ Improved the Zend Memory Manager
- ☑ Over 300 fixes between 5.1.6 and 5.2.1

PHP 5.2: New extensions

☑	Filter	Input filtering/validation
☑	Date	Date handling functions and objects
☑	JSON	JavaScript Object Notation
☑	XMLWriter	Counterpart to XMLReader
☑	ZIP	Full read & write ZIP support

PHP 5.2: New features

- ☑ Now __tostring() works as expected everywhere
- ☑ New error severity E_RECOVERABLE_ERROR
- ☑ Added/improved SPL features
  - ☑ RegexIterator
  - ☑ SplFileObject with CSV read support
  - ☑ CachingIterator now allows caching
- ☑ RFC 2397 (data: stream) support
- ☑ Added forward support for 'b' prefix to strings
- ☑ Lots of minor additions/improvements

PHP 5.2: Performance

- ☑ New Memory Manager + Heap Protection
- ☑ Faster include/require_once
- ☑ Optimized str_replace() and implode() functions
- ☑ Faster try { } catch { } blocks
- ☑ Significantly faster performance on Win32
- ☑ Optimized shutdown sequence
- ☑ Many other optimizations

PHP 5.2: Security

- ☑ New configuration option allow_url_include
- ☑ Over 40 security fixes
- ☑ More accurate memory usage tracking
- ☑ Filter extension can help filter out hostile input
  - ☑ XSS
  - ☑ SQL Injection
- ☑ Memory limit is always enabled.

PHP 5.2: Changes

- ☑ Respect new E_RECOVERABLE_ERROR in E_ALL
- ☑ No more 'abstract static' methods in classes
- ☑ When using external resources, allow_url_include
- ☑ Finally activated classes DateTime, DateTimeZone
- ☑ Inheritance rules check for return by reference

- ☑ Detailed instructions in README.UPDATE_5_2

PHP 5.3

- ☑ Scheduled for end of 2007
- ☑ New features that cannot be done in 5.2

- ☑ In discussion:
  - ☑ Bundling APC or another compiler cache
  - ☑ Namespaces
  - ☑ Adding operator `ifsetor` or `?:`

```
$a = ifsetor($input, $default);

$a = $input ?: $default;
```

PHP 6

- ☑ Integrated native Unicode support leveraging ICU
- ☑ Extensions iconv and mbstring get deprecated

- ☑ Native Unicode string type using UTF-16
- ☑ Additional binary string type
- ☑ Backwards compatible
- ☑ Controlled via INI --unicode-semantics={on|off}
- ☑ New escape sequences \uXXXX and \UXXXXXXXX

--unicode.semantics=off

- ☑ Old behavior where 1 character means 1 byte

```
$str = "Hello, world";      // ASCII encoding
echo strlen($str);          // result is 12

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

--unicode.semantics=on

- ☑ Strings are unicode type; 1 character >= 1 byte

```
$str = "Hello, world";      // Unicode string
echo strlen($str);          // result is 12

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

Wait, I like binary

- ☑ The prefix b denotes binary literals

```
$str = b"Hello, world";      // Binary string
echo strlen($str);           // result is 12
```

At Last some Hints

- ☑ List of all SPL classes — PHP 5.0.0

```
php –r 'print_r(array_keys(spl_classes()));'
```

- ☑ Reflection of a built-in class — PHP 5.1.2

```
php --rc <Class>
```

- ☑ Reflection of a function or method — PHP 5.1.2

```
php --rf <Function>
```

- ☑ Reflection of a loaded extension — PHP 5.1.2

```
php --re <Extension>
```

- ☑ Extension information/configuration — PHP 5.2.2

```
php --ri <Extension>
```

THANK YOU

- ☑ This Presentation
  http://somabo.de/talks/
- ☑ PHP
  http://php.net
- ☑ Upgrading to 5.2
  README.UPDATE_5_2
- ☑ SPL Documentation
  http://php.net/~helly
- ☑ Phar
  http://pecl.php.net/packages/phar
  http://php.net/phar