CONTENTS

What's new in PHP 5.5 & 5.6

By Luis Atencio

ABOUT

PHP is a server-side scripting language designed for the development of web applications. With its concise syntax, rich function library, and powerful language constructs, PHP continues to be the platform of choice amongst web developers

```
Generator implements Iterator {
   public mixed send ( mixed $value )
   public mixed throw ( Exception $exception )
   public void __wakeup ( void )
}
```

These methods allow a generator internally to save its state and resume where it left off next time it's called.

The PHP range($start, $end, [, number $step = 1]) function is used to generate an array of values with every element in it from start to end parameters inclusively, separated by the step value.

```
foreach (range(0, 100000) as $number) {
    echo $number;
}
```

This statement will create an array of 100001 elements (including 0), loop over all of them and print each one. Understandably, the performance of the range function is degraded as the end value becomes bigger. Generators can be used to implement range() much more effectively. Let's call it xrange($start, $end, [, number $step = 1]).

```
function xrange($start, $limit, $step = 1) {
    if ($start < $limit) {
        for ($i = $start; $i <= $limit; $i += $step) {
            yield $i;
        }
    } else {
        for ($i = $start; $i >= $limit; $i += $step) {
            yield $i;
        }
    }
}
```

Generator functions return a Generator instance, which we can place inside of a foreach statement and iterate over a range of elements without ever needing additional memory.

```
foreach (xrange(0, 100000) as $number) {
    echo $number;
}
```

Let's look at simplifying another task done on a daily basis where
ï L ï + ¤ 1 M

```
Let's look at simplifying another task done on a daily
basis where generators can play a significant role,
file I/O:
function readLinesFrom($filename) {
    $fh = fopen($filename, 'r');

    if (!$fh)
       throw new Exception('Error opening file '.
$filename);

    while (($line = fgets($fh)) !== false)
        yield $line;
    }
    fclose($fh);
}
```

6 ï F 6
F # L
generator function yields each line one at a time, starting where it left off without creating any additional memory allocations. The calling code looks like the following:

```
foreach (readLinesFrom('myFile.txt') as $line) {
   // do something with $line
}
```

#
to the generator, which could be useful for stopping the generation
F % ï Z[
above:

```
function xrange($start, $limit, $step = 1) {
   if ($start < $limit) {
      for ($i = $start; $i <= $limit; $i += $step) {
         $res = (yield $i);
         if($res == 'stop') {
           return;//exit the function
         }
      }
   }
}

$gen = xrange(1, 10);

foreach($gen as $v)
{
    if($v == 5) {
       // stop generating more numbers
       $gen->send('stop');
    }
    echo "{$v}\n";
}
```

The result of the current yield expression can be captured and evaluated. By calling the send function in the client code, a value
F

More information on generators can be found at http://php.net/manual/en/language.generators.overview.php.

EXCEPTION HANDLING AND THE FINALLY KEYWORD

Exception handling had been added to PHP in version 5. Code can be surrounded in a try-catch block to facilitate the catching
F % ï + ¤ 1
+ ¤ 1 F '
one catch block and multiple catch blocks can be used to handle different types of exceptions.

2 * 2 ¿ F ¿ ï F 5 , L
ï U
run regardless of whether an exception occurs within the try block or before normal execution resumes. Here is the general syntax:

```
$handle= fopen('myFile.txt', 'w+');
try {
   // Open a directory and run some code
   fopen($handle, 'Some Text');
}
catch (Exception $e) {
   echo 'Caught exception: ', $e->getMessage(), "\n";
}
finally {
   fclose($handle);
   echo "Close directory";
}
```

6 ï
taken up before or during the execution of the try statement.
# ï 2 * 2 e
handling mechanism to compete with other programming languages, and also to improve the quality of error handling code.
7 , L L 2 * 2
checked and unchecked exceptions – essentially all exceptions are runtime exceptions.

SECURITY

2*2 ¿F¿ #2+
hashes using the same underlying library as crypt(), which is a one-way password hashing function available in PHP since version 4. Password hashing is very useful when needing to store and validate passwords in a database for authentication purposes.

The new password_hash() function acts as a wrapper to crypt() making it very convenient to create and manage passwords in a
F 5 #2+
for developers to start adopting it over the much older and weaker md5() and sha1() functions, which are still heavily used today.

Since it is built-in to the PHP core, this extension has no dependencies on external libraries, and does not require special php.ini directives to use.

6 #2+ M

FUNCTION NAME	DESCRIPTION
password_get_info	Returns information about the given hash as an array: salt, cost and algorithm used
password_hash	Creates a password hash
password_needs_rehash	Checks if the given hash matches the options provided
password_verify	8 ï hash

Here is some sample code:

```
// generate new hash
$password = 'MyP@ssword!';
$new_hash = password_hash($password, PASSWORD_DEFAULT);
print "Generated New Password Hash: " + $new_hash + "\n";

// get info about this hash
$info = password_get_info($new_hash);
var_dump($info);

// verify hash
$isVerified = password_verify($password, $new_hash);
if($isVerified) {
		print "Password is valid!\n";
}

// check for rehash
$needs_rehash = password_needs_rehash($new_hash,
PASSWORD_DEFAULT);
if(!$needs_rehash) {
		print "Password does not need rehash as it is
valid";
}
```

6 #2+
password hashes: salt, the algorithm, and a cost.

OPTION NAME	DESCRIPTION
salt	This is an optional string to base the hashing F 6 #2+ require or recommend the use of a custom salt.
algo	The default algorithm built in to PHP 5.5 is the $ Z $ ï [L free to use others supported by the crypt() function $ ï 5 &'5F
cost	# value that represents the number of iterations used when generating a hash. In the password hash #2+L »º Z¼'»º ˆ »º¼¾ iterations) is used, but you can increment this on better hardware. Cost values range from 04 – 31.

More information on password hashing, visit http://php.net/manual/en/book.password.php.

PHP 5.5 PLATFORM ENHANCEMENTS

The PHP 5.5 release also contains minor enhancements to the language constructs.

ENHANCEMENT TO *FOREACH( )*

Support for list( )

The foreach() statement now supports the unpacking of nested arrays (an array of arrays) to perform quick variable assignments via the list() function. list() had been present in PHP since version 4, but could not be used in this capacity until now.

Let's see an example:

```
$vector = [
   [0, 1, 2],
   [3, 4, 5],
   [2, 1, 2],
];

foreach ($vector as list($x, $y, $z)) {
   printf("Vector components [%d, %d, %d] \n", $x, $y,
$z);
}
```

L p L p L p!
F + e ï
L 2*2 F 1 L
use fewer values to initialize variables:

```
$array = [
   [1, 2],
   [3, 4],
];

foreach ($array as list($x)) {
   // Note that there is no $y here.
   echo "$x\n";
}
```

Support for non-scalar keys

PHP 5.5 lifts the restriction on foreach() valid keys to allow
L F 6
only to iterator keys (iterator::key()) present in the loop statement; non-scalar keys still cannot occur in native PHP arrays.

6 52. 5 1 5
L F 6
purpose can be useful when having the need to uniquely identify
ï F

Suppose you want to uniquely identify cities by its coordinates. Using simple scalar keys will not be enough; for instance, we can use a custom Coordinate class, as such:

```
$coords = new SplObjectStorage();

$ny = new Coordinate(40.7127, 74.0059);
$mia = new Coordinate(25.7877, 80.2241);


$coords [$ny] = "New York";
$coords [$mia] = "Miami";

foreach($coords as $c -> $city) {
   echo "City: $city | Lat: $c->lat | Long: $c->long";
}
```

Before 5.5, the code above would not have been allowed, as the
p U F

Note: as of this writing, the PHP manual for the Iterator::key() function has not been updated to account for the non-scalar return type.

Find more information at http://php.net/manual/en/iterator.key.php.

ARBITRARY EXPRESSIONS AND *EMPTY()*

5 # , 5 F
6 ï L
F # , 5
manually bootstrap the application. You will need to remove the ng-app directive from the HTML and use the angular.bootstrap function instead. Make sure that you declare your modules before using them in the angular.bootstrap function. The following code shows a snippet of using manual bootstrap:

```
!isset($var) || $var == false
```

Let's take a look at some examples:

```
function is_false() {
    return false;
}

if (empty(is_false())) {
    echo "Print this line";
}

if (empty(true)) {
    echo "This will NOT be printed. ";
}
if (empty(0) && empty("") && empty(array())) {
    echo "This will be printed. ";
}
```

1 L Z[
For more information on the use of empty, visit http://php.net/manual/en/function.empty.php.

ARRAY AND STRING LITERAL DEREFERENCING

Similar to Perl and Python, PHP 5.5 has added language support for direct dereferencing via the index operator. This provides a quick way to obtain the character or index value of an array or string literal, staring at zero.
Examples:

```
echo 'Direct Array dereferencing: ';
echo [1, 2, 3, 4, 5, 6, 7, 8, 9][2];  // will print 3

echo 'String character dereferencing: ';
echo 'DZONE'[0];                       // will print D
```

If you're dereferencing an index value outside of the length of an
L 7 ï 7 !
error will be thrown, respectively.

CLASS NAME RESOLUTION VIA ::CLASS

PHP 5.5 has made the addition of the basic keyword class which, preceded by the scope resolution operator ::, is treated as a static or constant property of a class. This keyword will be used to obtain
ï F 6
especially useful with namespaced classes. For more information on namespaces, visit http://php.net/manual/en/language.namespaces.php.

For example:

```
namespace MyAppNs\Sub {

   class ClassName {

   }

   echo ClassName::class;

   // this will print MyAppNs\Sub\ClassName
}
```

For more information on class name resolution, visit http://php.net/manual/en/language.oop5.basic.php - language.oop5.basic.class.class.

CLASS NAME RESOLUTION VIA ::CLASS

PHP is an interpreted language; every time a script runs it gets compiled to bytecode form, which is then executed. Since scripts don't change on every request, it makes sense to include a level of opcode caching (code compilation caching) that will improve the performance of script execution.

6 < 1 „ 1 2 2 * 2
1 2 % L # 2 % F 1
caching is a platform optimization feature applied to the execution lifecycle of a PHP script. This extension drastically improves the performance of PHP by storing precompiled script bytecode into shared memory and removing the need to reload and parse entire scripts on each request. There have been cases where a performance
½ 1 2 F

This extension is bundled into PHP 5.5 but is also available as a PECL package, which can be used with previous versions of PHP. PECL is a repository for the distribution and sharing of PHP extensions.

F M¤¤ F F ¤ ¤< 1

6 < 1 2 L
following function from a PHP terminal:

```
!isset($var) || $var == false
```

You should see the following included as part of the output:

```
'version' =>
       array(2) {
       'version' => string(9) "7.0.4-dev"
       'opcache_product_name' => string(12) "Zend
OPcache"
```

4 F 1 2 M

KEY	VALUE
opcache.memory_consumption	128
opcache.interned_strings_buffer	8
opcache.revalidate_freq	60
opcache.fast_shutdown	1
opcache.enable_cli	1
opcache.save_comments	enable or disable depending on application
opcache.max_accelerated_files	4000

For more information about the meaning of these directives please
12 ï M
M¤¤ F ¤ ¤ ¤ F ï F

NEW FUNCTIONS ADDED

Many functions were added in PHP 5.5 — too many to be listed
F 5 ï L ! !
was completely revamped. To get the complete list of all functions added, please visit http://php.net/manual/en/migration55.new-functions.php.

In addition, here are some other functions added that are useful to keep at hand:

KEY	USAGE
array_column($array, $column_key)	When dealing with nested array elements, you can use this function to ï
boolval($var)	1
json_last_error_msg()	1 json_encode() or json_decode() function call
mysqli::begin_transaction($flags)	Marks the beginning of a transaction
mysqli::release_savepoint($name)	Rolls back a transaction to the named
	savepoint
mysqli::savepoint()	Sets a named transaction savepoint

PHP 5.5 INCOMPATIBILITIES

PHP 5.5 introduces a few backward incompatible changes as well
#2+ F #
checked thoroughly when planning to do a migration to 5.5

DEPRECATIONS

- ' ¤ 'X&'24'%#6'&
  F 7 / 53. 2&1X/ 53.
  extensions instead. The biggest change here is that mysql_connect() has been deprecated.
- 6 ¤ ï X Z[F
  6 ï
  when using backreferences in strings with either single or
  F # L 2*2
  replacement parameter. Instead, use preg_replace_callback(), a faster, cleaner, and easier-to-use function.

For instance:

```
$line = 'Hello World';
echo preg_replace_callback(' /(\\w+)/ ',
        function ($matches) {
            return strtolower($matches[0]);
        },
        $line
    );  // will print hello world
```

- Encryption functions from the mcryt extension have been deprecated:

```
o mcrypt_cbc()
o mcrypt_cfb()
o mcrypt_ecb()
o mcrypt_ofb()
```

BACKWARD INCOMPATIBLE CHANGES

- Windows XP and 2003 support were dropped, which means the
  ï F 6
  Windows build for PHP will now require Windows Vista or newer.
- Up until now, PHP language constructs are case-sensitive. Meaning, you could write if-else and for loop statements in any case (including mixed case).
- # U L L
  U #5%++
  rules. This improves support for languages with unusual collating rules, such as Turkish. This may cause issues with code bases that
  U U#5%++ 76(U
  8. This was done because some locales (like Turkish) have every unexpected rule for lowercasing and uppercasing. For example, lowercasing the letter "I" gives different results in English
  6 F 6 L #5%++
  lowercasing.
- # L self, parent, and static keywords are now case-insensitive. Prior to 5.5, these keywords were treated in a case-sensitive manner. This has been resolved; hence, self::CONSTANT and SELF::CONSTANT will now be treated identically.

BACKWARD INCOMPATIBLE CHANGES

In PHP 5.6, it is now possible to use scalar expressions involving numeric and string literals in static contexts, such as constant and property declarations as well as default function arguments. Before 5.6, you could only do this with static variables. Let's take a look:

```
const ONE = 1;
const TWO = ONE + ONE;

class MyClass {
   const THREE = TWO + 1;
   const ONE_THIRD = ONE / self::THREE;
   const SENTENCE = 'The value of THREE is '.self::THREE;

   public function f($a = ONE + self::THREE) {
      return $a;
   }
}

echo (new MyClass)->f();  // prints 4
echo MyClass::SENTENCE;   // prints The value of THREE is 3
```

In previous releases, this would have generated parser or syntax errors. PHP 5.6 augments the parsing order allowing PHP expressions as constant values, which can contain variables as well as other constants. Similar to static variable references, you can access a class's constant values via the :: scope resolution operator.

VARIADIC FUNCTIONS

This is an enhancement to the variable-length argument list (or [F # 2*2 ¿F¿L of the functions func_num_args(), func_get_arg(), and func_get_args() in order work with variable-length arguments.

& 2 4 "*" to tell the interpreter to pack as many parameters provided in a function invocation into an array. Starting with PHP 5.6, arguments may include the ellipsis token to establish a function as variadic, , e F 6 will be treated as an array.

```
function sum(...$numbers) {
    $s = 0;
    foreach($numbers as $n) {
        $s += $n;
    }
    return $s;
}
echo sum(1, 2, 3, 4);  // prints 10
```

L ï Z`[F

ARGUMENT UNPACKING

In somewhat similar fashion to the use of list()introduced in PHP 5.5, you can use the ellipsis token to unpack a Traversable structure such as an array into the argument list.

```
function sum($a, $b, $c) {
    return $a + $b + $c;
}

echo sum(...[1, 2,2]); // prints 6
$arr = [1, 2, 3];
echo add(...$arr);     // prints 6
```

Furthermore, you can mix normal (positional) arguments with varargs, with the condition that the latter be placed at the end of the arguments list. In this case, trailing arguments that don't match the positional argument will be added to the vararg.

```
function logMsg($message, ...$args) {
    echo "${message} : on line ${args[0]} Class:
${args[1]}";
}
logMsg("Error", 32, "MyClass");
// prints Error: on line 32 Class MyClass
```

EXPONENTIATION

PHP 5.6 has added the operator **, a right-associative operator L YYˆ assignment.

```
printf("num= %d ", 2**3**2); // prints num= 512

$a = 2;
$a **= 3;
printf("a= %d", $a);        // prints a=8
```

NAMESPACING AND ALIASING

Introduced in PHP 5.3, namespaces provided a way to encapsulate or group a set of items, in very much the same way to how ï F 6 ï ð F 6 2*2F 6 , e F

Before namespaces existed, PHP developers would include the directory structure as part of their class names, yielding incredibly long class names. In other words, you would see class names like:

```
PHPUnit_Framework_TestCase
```

This was the best practice at the moment. This would refer to the class TestCase that lives inside the PHPUnit/Framework directory. Namespaces are designed to solve 2 problems:

1. # L 2*2 L and third party libraries
2. Improve readability by aliasing (shortening) long class names

The use keyword provides developers the ability to refer to an L U ï F 6 2 e F # 2*2 support namespaces and up to PHP 5.5 support aliasing, or importing a class name, interface name, or namespace. PHP 5.6 has extended this behavior to allow aliasing of functions as well as constant names.

Here is an example of the different uses of namespaces and the use keyword. The following statements build on each other:

```
namespace foo;

// Alias (shorten) full qualified class
use com\dzone\Classname as Another;

// Similar to using com\dzone\NSname as NSname
use com\dzone\NSname;

// Importing a global class
use ArrayObject;  // ArrayObject is a global PHP class

// Importing a function (PHP 5.6+)
use function com\dzone\functionName;

// Aliasing a function (PHP 5.6+)
use function com\dzone\functionName as func;

// Importing a constant (PHP 5.6+)
use const com\dzone\CONSTANT;

// Instantiates object of class foo\Another
$anotherObj = new namespace\Another;

// Similar to instantiating object of class com\dzone\
Classname
$anotherObj = new Another;

// Calls function com\dzone\NSname\subns\func
dzone\subns\func();

// Instantiates object of class core class ArrayObject
$a = new ArrayObject(array(1));

// Prints the value of com\dzone\CONSTANT
echo CONSTANT;
```

Here is an example of the different uses of namespaces and the use keyword. The following statements build on each other:

```
namespace com\dzone {
    const FOO = 42;
    function myFunc() { echo __FUNCTION__; }
}

namespace {
    use const com\dzone\FOO;
    use function com\dzone\ myFunc;

    echo FOO;   // prints 42
    myFunc ();  // prints com\dzone\myFunc
}
```

DEBUGGING

Until now, the most common way to debug PHP scripts was by using the Xdebug extension. Xdebug would instrument a running PHP script by setting up a special connection to the server. Using ï L
to developers.

PHP 5.6 now includes a built-in interactive debugger called *phpdbg* ï U 5#2+ Z5 #2+[F #
the debugger can exert complete control over the environment without any additional connections and overhead. This lightweight, powerful debugger will run without impacting the functionality or performance of your code. For more information about the debugger and to install it, visit the homepage:

http://phpdbg.com/

phpdbg has several features including:

- 5 U &
- Flexible Breakpoints
- ' # 2*2 U Z[
- ' # % ' %
- 7 #2+
- 5#2+ # U ' +
- 2*2 % ï (5
- ,+6 5)
- 1 5 U % 6 1
- 4 & 5 U $,)7+

STRING COMPARISON

PHP 5.6 added a function called hash_equals()to compare strings in constant time. This is not meant to be used for all string L ï
hide the amount of time taken to compare 2 strings. This is very useful to safeguard against timing attacks.

Timing attacks attempt to discover username and/or password lengths by doing a relative comparison of response time differences, as a result of processing a login form with an invalid username against a known valid one.

2*2 ¿F¿ ï #2+ Z
before) already makes use of this function.

```
bool hash_equals ( string $known_string , string
$user_string )
```

OBJECT INFO

The magic method __debugInfo has been added in order to control X Z[F
This is very similar to a class overriding the toString() method in , F

```
class MySquareClass {
    private $val;

    public function __construct($val) {
        $this->val = $val;
    }

    public function __debugInfo() {
        return [
            'valSquared' => $this->val ** 2,
        ];
    }
}

var_dump(new C(3));
```

This will print:

```
var_dump(new C(3));

This will print:

object(MyClass)#1 (1) {

  ["propSquared"]=>

  int(9)
}
```

ENCODING

In PHP 5.6 and onwards, "UTF-8" will be used as the default character encoding for forhtmlentities(), html_entity_decode() and htmlspecialchars() if the encoding parameter is omitted. The value of the default_charset php.ini will be used to set the default encoding for iconv functions as well as multi-byte mbstring functions.

PHP 5.6 INCOMPATIBILITIES

PHP 5.6 introduces a few backward incompatible changes as well #2+ F 9
new release of PHP, the core team has committed to reducing incompatibility changes going forward.

DEPRECATIONS

- Methods from an incompatible context are deprecated and will 'X&'24'%#6'& F 6
  invoking a non-static function in a static context. For instance:

```
class A {
    function f() { echo get_class($this); }
}

class B {
    function f() { A::f(); } // incompatible context
}
```

  Support for these calls will later be removed altogether.
- 6 ï
  encoding have been deprecated in favor of default_charset.

BACKWARD INCOMPATIBLE CHANGES

- # e ï property of a class via an array literal. Before 5.6, arrays declared as a class property with mixed keys could have array elements silently overridden when an explicit key had the same value as a sequential implicit key. For example:

```
class MyClass {
    const ONE = 1;
    public $array = [
        1 => 'foo',
        'bar',
        'joe',
    ];
}

var_dump((new C)->array);
```

In PHP 5.5, this will output:

```
array(3) {
  [1]=>
  string(3) "foo"
  [2]=>
  string(3) "bar"
  [3]=>
  string(4) "quux"
}
```

- 6 X U ,510 L L L ï F

- + #2+ PHP 5.5, the Mcrypt functions now require valid keys and initialization vectors (IV) to be provided.

CONCLUSION

In sum, PHP 5.5 and 5.6 have introduced many important language enhancements that improve the overall quality and stability of the F # L ï introduction of function generators and the yield keyword; the ï U N #2+N 12 N expressions, variadic functions, a lightweight debugger, and many additional functions and changes to the core platform.

While these releases did not send tremors through the PHP community, some backward incompatible changes were introduced that can make migration to this platform a bit unnerving. Thus, a degree of caution and overall regression-testing measures must be taken when upgrading production systems to 5.5 or 5.6. The PHP team is strongly committed to making future releases much more backward compatible.

Before migration, please take a look at the following pages, for both 5.5 and 5.6, respectively:

http://php.net/manual/en/migration55.php

,510 http://php.net/manual/en/migration56.php

ABOUT THE AUTHOR

Luis Atencio is a Staff Software Engineer for Citrix Systems in Ft. Lauderdale, FL. He has earned both B.S. and M.S. degrees in % 5 F . , 2*2 platforms. In addition, he writes a developer blog at http://www.luisatencio.net focusing on software engineering. When Luis is not coding or writing, he likes to practice soccer and Muay Thai kickboxing, and play guitar.

RECOMMENDED BOOK

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1

50795

9 781936 502776

VERSION 1.0 $7.95