PHP

LCA 2005

April 20, 2005. Canberra

Rasmus Lerdorf <rasmus@php.net>

http://talks.php.net/lca05

Why are you here?

Kernel Hacking is Hard

Rusty scares me!

Agenda

Introduction

PHP5

Security

Large developer teams

High-complexity applications

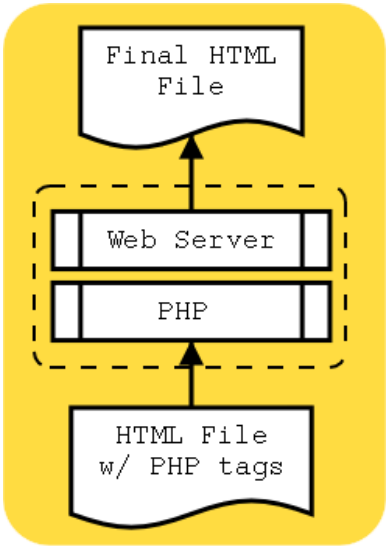
High-traffic Applications

I18n and L10n

Writing your own PHP Extension

PHP is a Server-side language

Even though it is embedded in HTML files much like the client-side Javascript language, PHP is server-side and all PHP tags will be replaced by the server before anything is sent to the web browser.



So if the HTML file contains:

```
<html>
<?php echo "Hello World" ?>
</html>
```

What the end user would see with a "view source" in the browser would be:

```
<html>
Hello World
</html>
```

Idea

This is a C program that generates an HTML page with a table populated from an SQL query.

```
#include <stdio.h>
#include <stdlib.h>
#include "mysql.h"

MYSQL mysql;

void err(void) {
    fprintf(stderr, "%s\n", mysql_error(&mysql) );
    exit(1);
}

int gen_page() {
    puts("<html><body><table>");
    if(!(mysql_connect(&mysql,"host","user","passwd")))
err();
    if(mysql_select_db(&mysql,"my_db")) err();
    if(mysql_query(&mysql,"SELECT * FROM my_table")) err();
    if(!(res = mysql_store_result(&mysql))) err();
    while((row = mysql_fetch_row(res))) {
        puts("<tr>");
        for (i=0 ; i < mysql_num_fields(res); i++)
            printf("<td>%s</td>\n",row[i]);
        puts("</tr>");
    }
    puts("</table></body></html>");

    if (!mysql_eof(res)) err();
    mysql_free_result(res);
    mysql_close(&mysql);
}
```

Here is the same thing in PHP which produces the same output.

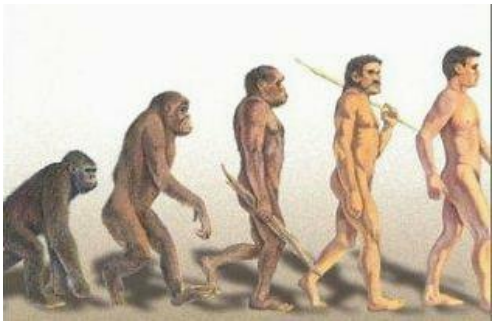
```
<html><body><table>
<?php
  mysql_connect("host","user","passwd") or
die(mysql_error());
  $res = mysql_db_query("my_db","SELECT * FROM my_table") or
die(mysql_error());
  while(($row = mysql_fetch_row($res))) {
      echo "<tr>";
      for ($i=0 ; $i < mysql_num_fields($res); $i++)
          printf("<td>%s</td>\n",$row[$i]);
      echo "</tr>";
  }
?>
</table></body></html>
```

Agenda

PHP5

An overview and some cool stuff

Evolutionary Changes



- Improved Performance.
- Some New Standard Functions.
- Improved Streams Support
- New & Exciting Bugs

Language internals remain mostly the same.

Revolutionary Changes

- OO support completely rewritten
- Much improved XML support
- SQLite
- MySQLi
- Tidy
- Many extensions moved to PECL

And in PHP 5.1

- PDO
- xmlreader/xmlwriter
- Performance Improvements

In PHP4

```
<?php
    class OS {
        var $name;
        function OS($name) {
            $this->name = $name;
        }
    }

    function changeName(&$obj, $name) {
        $obj->name = $name;
    }

    $linux = new OS('linux');
    $win = $linux;
    changeName($win, 'windows');
    echo $linux->name. "<br />\n";
    echo $win->name;
?>
linux
windows
```

In PHP5

```
<?php
    class OS {
        var $name;
        function OS($name) {
            $this->name = $name;
        }
    }

    function changeName($obj, $name) {
        $obj->name = $name;
    }

    $linux = new OS('linux');
    $win = clone $linux;
    changeName($win, 'windows');
    echo $linux->name. "\n";
    echo $win->name;
?>
linux
windows
```

Constructors and Destructors

```
<?php
    class OS {
        function __construct($name) {
            echo "Constructor called<br />";
        }

        function __destruct() {
            echo "Destructor called<br />";
        }
    }

    $linux = new OS('linux');
    echo "Step 1<br />";
    $ref   = $linux;
    echo "Step 2<br />";
    $linux = NULL;
    echo "Step 3<br />";
    $ref = NULL;
    echo "Step 4<br />";
?>
```

Output:

```
Constructor calledStep 1Step 2Step 3Destructor calledStep 4
```

PHP5 OO - Private

- 'Strongest' access level
- Can only be accessed/called from the same class
- Force use of get()/set() functions to access properties

```
<?php
class Bedroom {
    private $action;

    function __construct() {
        $this->action = 'fun';
    }
}

$br = new Bedroom();
echo $br->action. "\n";
?>
Fatal error: Cannot access private property bedroom::$action
in foo.php on line 11
```

Only accessible/callable by the same, or an inherited class. Here we see a call to a protected method from the global scope being rejected.

```
<?php
class Bedroom {
    protected $action;

    function __construct() {
        $this->action = 'fun';
    }

    protected function peek() {
        echo $this->action. "\n";
    }
}

class Keyhole extends Bedroom {
    function peek() {
        echo $this->action. "\n";
    }
}

$kh = new Keyhole();
$kh->peek();

$br = new Bedroom();
$br->peek();           / Doesn't work /
?>
fun
Fatal error: Call to protected method Bedroom::peek() from
context 'html' in script.php on line 24
```

PHP5 OO - set/get

Property access handlers combined with 'private' can be used as a replacement for dynamic properties.

```
<?php
class OSses {
    private $names = array();

    public function __set($name, $value) {
        if (!in_array($name, array('windows', 'dos')))
            $this->names[$name] = $value;
    }

    public function getNames() {
        return $this->names;
    }
}

$osses = new OSses();
$osses->linux = 'rocks';
$osses->windows = 'blows';

var_dump($osses->getNames());
?>
array(1) {
  ["linux"]=>
  string(5) "rocks"
}
```

__call() is used to catch method calls the same way __get() and __set() is used to catch property accesses

```
<?php
class hello {
    function __call($name, $args) {
        echo "Hello ".ucfirst($name)."!\n";
    }
}

$h = new hello;
$h->rusty();
$h->anton();
?>
Hello Rusty!
Hello Anton!
```

PHP4:

```
<?php
if (mysql_connect('localhost')) {
   if (mysql_select_db('game')) {
      $r = mysql_query('SELECT * FROM news');

      ...
```

PHP5:

```
<?php
try {
    mysql_connect('localhost');
    mysql_select_db('game');
    $r = mysql_query('SELECT * FROM news');
} catch (Exception $e) {
    / Do recovery /
    exit();
}
```

```
<?php
class a_class {
  const VAL = 'Something';
  public static $var = " Interesting";

  public static function foo() {
    return self::$var;
  }
}
echo a_class::VAL;
echo a_class::foo();
?>
```

Output:

```
Something Interesting
```

Autoload and __toString()

Base.inc

```
<?php
abstract class Base {
  public function __toString() {
    return '<pre>'.print_r($this,TRUE).'</pre>';
  }
}
?>
```

script.php

```
<?php
function __autoload($class) {
    echo "autoload called for $class\n";
    require "presentations/slides/intro/$class.inc";
}

class my_class2 extends Base {
  function __construct($arg) {
    $this->prop = $arg;
  }
}
$test = new my_class2('foo');
echo $test;
?>
```

Output:

```
autoload called for Base
my_class2 Object
(
    [prop] => foo
)
```

```
<?php
interface abc {
   public function setVariable($name, $var);
   public function getVariable($name);
}

class my_abc implements abc {
   private $vars = array();

   public function setVariable($name, $var) {
       $this->vars[$name] = $var;
   }

   public function getVariable($name) {
       return $this->vars[$name];
   }
}
$test = new my_abc;
$test->setVariable('foo','bar');
echo $test->getVariable('foo');
?>
```

Output:

```
bar
```

Now, how about if we don't implement the interface correctly:

```
<?php
class my_other_class implements abc {
    function __construct() {
        $this->test = 'something';
    }
}
?>
Fatal error: Class my_other_class contains 2 abstract methods
and must therefore be declared abstract (abc::setVariable,
abc::getVariable) in script.php on line 2
```

```
<?php
  class my_main_class {
    public function foo() { }
    final public function bar() { }
  }

  class my_other_class extends my_main_class {
    public function foo() { }
    public function bar() { }
  }
?>
Fatal error: Cannot override final method my_main_class::bar()
in script.php on line 9
```

Object Type Hinting

```
<?php
  class the_class {
    public $var = 'foo';
  }
  class my_other_class {
    public function __construct(the_class $obj) {
      echo $obj->var;
    }
  }
  $a = new the_class;
  $b = new my_other_class($a);
  $c = new my_other_class($b);
?>
foo
Fatal error: Argument 1 must be an instance of the_class in
script.php on line 6
```

```
<?php
class misc_class {
  public  $var1 = "abc";
  public  $var2 = 123;
  public  $var3 = "def";
  private $var4 = 456;
}
$test = new misc_class;
foreach($test as $key=>$value)
  echo "$key = $value<br />\n";
?>
```

Output:

```
var1 = abc
var2 = 123
var3 = def
```

You can overload PHP's internal iterator by redefining the rewind(), current(), key(), next() and valid() methods like this:

```
<?php
class MyIterator implements Iterator {
   private $var = array();

   public function __construct($array) {
       if (is_array($array)) {
           $this->var = $array;
       }
   }
   public function rewind() {
       echo "rewinding<br />\n";
       reset($this->var);
   }
   public function current() {
       $var = current($this->var);
       echo "current: $var<br />\n";
       return $var;
   }
   public function key() {
       $var = key($this->var);
       echo "key: $var<br />\n";
       return $var;
   }
   public function next() {
       $var = next($this->var);
       echo "next: $var<br />\n";
       return $var;
   }
   public function valid() {
       $var = $this->current() !== false;
       echo "valid: {$var}<br />\n";
       return $var;
   }
}

$it = new MyIterator(array(1,2,3));
foreach ($it as $key=>$value) {
   echo "$key: $value<br />\n";
}
?>
```

Output:

```
rewinding
current: 1
valid: 1
current: 1
key: 0
0: 1
next: 2
```

```
current: 2
valid: 1
current: 2
key: 1
1: 2
next: 3
current: 3
valid: 1
current: 3
key: 2
2: 3
next:
current:
valid:
```

If you just want to override some of them, use the IteratorAggregate class.

Controlling Cloning

```
<?php
class my_class3 {
  static $instances = 0;
  public $instance;

  public function __construct() {
    $this->instance = ++self::$instances;
  }
  public function __clone() {
    $this->instance = ++self::$instances;
  }
}
$test = new my_class3;
$test2 = clone $test;
echo $test->instance;
echo "<br />\n";
echo $test2->instance;
?>
```

Output:

```
1
2
```

Note that we are not overriding the cloning with our __clone() method. Think of it as a fixup that is called after the actual copy has been done.

Comparing Objects

You can compare objects using == and ===.

```
<?php
  class some_class {
    public $var = 'foo';
  }
  $a = new some_class;
  $b = $a;
  $c = clone $a;
  $d = clone $a;
  $d->var = 'bar';

  var_dump($a==$b);  echo '<br />';
  var_dump($a===$b); echo '<br />';
  var_dump($a==$c);  echo '<br />';
  var_dump($a===$c); echo '<br />';
  var_dump($a==$d);  echo '<br />';
  var_dump($a===$d); echo '<br />';
?>
```

Output:

```
true
true
true
false
false
false
```

```
<?php
class Reflection { }
interface Reflector { }
class ReflectionException extends Exception { }
class ReflectionFunction implements Reflector { }
class ReflectionParameter implements Reflector { }
class ReflectionMethod extends ReflectionFunction { }
class ReflectionClass implements Reflector { }
class ReflectionObject extends ReflectionClass { }
class ReflectionProperty implements Reflector { }
class ReflectionExtension implements Reflector { }
?>
```

Using this API you can poke things to see how they are put together. For example, using ReflectionClass you can look at a class:

```
<?php
  class my_test_class {
    public  $var1 = 'foo';
    private $var2 = 'bar';
    function __construct() {
       echo $this->var1;
    }
  }

  echo "<pre>";
  Reflection::export(new ReflectionClass('my_test_class'));
  echo "</pre>";
?>
```

Output:

```
Class [  class my_test_class ] {
  @@ /Users/rasmus/phpweb/pres2/display.php(1676) : eval()'d
code 2-8

  - Constants [0] {
  }

  - Static properties [0] {
  }

  - Static methods [0] {
  }

  - Properties [2] {
    Property [  public $var1 ]
    Property [  private $var2 ]
  }

  - Methods [1] {
    Method [   public method __construct ] {
      @@ /Users/rasmus/phpweb/pres2/display.php(1676) :
eval()'d code 5 - 7
    }
  }
}
```

We can of course also use Reflection to look at the ReflectionClass

```
<?php
  echo "<pre>";
  Reflection::export(new ReflectionClass('ReflectionClass'));
  echo "</pre>";
?>
```

Output:

```
Class [  class ReflectionClass implements Reflector ] {

  - Constants [0] {
```

```
}

- Static properties [0] {
}

- Static methods [1] {
  Method [  static public method export ] {
  }
}

- Properties [1] {
  Property [  public $name ]
}

- Methods [38] {
  Method [  final private method __clone ] {
  }

  Method [   public method __construct ] {
  }

  Method [  public method __toString ] {
  }

  Method [  public method getName ] {
  }

  Method [  public method isInternal ] {
  }

  Method [  public method isUserDefined ] {
  }

  Method [  public method isInstantiable ] {
  }

  Method [  public method getFileName ] {
  }

  Method [  public method getStartLine ] {
  }

  Method [  public method getEndLine ] {
  }

  Method [  public method getDocComment ] {
  }

  Method [  public method getConstructor ] {
  }

  Method [  public method hasMethod ] {
  }

  Method [  public method getMethod ] {
  }

  Method [  public method getMethods ] {
  }

  Method [  public method hasProperty ] {
  }

  Method [  public method getProperty ] {
  }

  Method [  public method getProperties ] {
  }

  Method [  public method hasConstant ] {
  }

  Method [  public method getConstants ] {
  }
```

```
    Method [  public method getConstant ] {
    }

    Method [  public method getInterfaces ] {
    }

    Method [  public method isInterface ] {
    }

    Method [  public method isAbstract ] {
    }

    Method [  public method isFinal ] {
    }

    Method [  public method getModifiers ] {
    }

    Method [  public method isInstance ] {
    }

    Method [  public method newInstance ] {
    }

    Method [  public method getParentClass ] {
    }

    Method [  public method isSubclassOf ] {
    }

    Method [  public method getStaticProperties ] {
    }

    Method [  public method getStaticPropertyValue ] {
    }

    Method [  public method setStaticPropertyValue ] {
    }

    Method [  public method getDefaultProperties ] {
    }

    Method [  public method isIterateable ] {
    }

    Method [  public method implementsInterface ] {
    }

    Method [  public method getExtension ] {
    }

    Method [  public method getExtensionName ] {
    }
  }
}
```

We saw that ReflectionClass implements the Reflector interface.

```
<?php
  echo "<pre>";
  Reflection::export(new ReflectionClass('Reflector'));
  echo "</pre>";
?>
```

Output:

```
Interface [  interface Reflector ] {

  - Constants [0] {
  }

  - Static properties [0] {
```

```
  }

  - Static methods [1] {
    Method [  abstract static public method export ] {
    }
  }

  - Properties [0] {
  }

  - Methods [1] {
    Method [  abstract public method __toString ] {
    }
  }
}
```

Or we can have a look at PHP's internal Exception class.

```
<?php
  echo "<pre>";
  Reflection::export(new ReflectionClass('Exception'));
  echo "</pre>";
?>
```

Output:

```
Class [  class Exception ] {

  - Constants [0] {
  }

  - Static properties [0] {
  }

  - Static methods [0] {
  }

  - Properties [6] {
    Property [  protected $message ]
    Property [  private $string ]
    Property [  protected $code ]
    Property [  protected $file ]
    Property [  protected $line ]
    Property [  private $trace ]
  }

  - Methods [9] {
    Method [  final private method __clone ] {
    }

    Method [    method __construct ] {

      - Parameters [2] {
        Parameter #0 [  $message ]
        Parameter #1 [  $code ]
      }
    }

    Method [  final public method getMessage ] {
    }

    Method [  final public method getCode ] {
    }

    Method [  final public method getFile ] {
    }

    Method [  final public method getLine ] {
    }

    Method [  final public method getTrace ] {
    }
```

```
    Method [  final public method getTraceAsString ] {
    }

    Method [  public method __toString ] {
    }
  }
}
```

```
<?php
    echo "<pre>";
    print_r(stream_get_wrappers());
    print_r(stream_get_filters());
    print_r(stream_get_transports());
    echo "</pre>";
?>
```

Output:

```
Array
(
    [0] => php
    [1] => file
    [2] => http
    [3] => ftp
    [4] => compress.zlib
    [5] => https
    [6] => ftps
)
Array
(
    [0] => string.rot13
    [1] => string.toupper
    [2] => string.tolower
    [3] => string.strip_tags
    [4] => convert.*
    [5] => convert.iconv.*
    [6] => zlib.*
)
Array
(
    [0] => tcp
    [1] => udp
    [2] => unix
    [3] => udg
    [4] => ssl
    [5] => sslv3
    [6] => sslv2
    [7] => tls
)
```

Examples

```
<?php
readfile("php://filter/read=string.toupper/resource=http://www.example.com");

$fp = fopen("compress.zlib://foo-bar.txt.gz", "wb");

$httpsfile =
file_get_contents("https://www.example.com/foo.txt");

$sock = fsockopen("ssl://secure.example.com", 443, $errno,
$errstr, 30);

?>
```

stream_copy_to_stream

```
<?php
$src = fopen('http://www.example.com','r');
$dest1 = fopen('first1k.txt','w');
$dest2 = fopen('remainder.txt','w');

stream_copy_to_stream($src, $dest1, 1024);
stream_copy_to_stream($src, $dest2);
?>
```

file_put_contents

```
<?php
```

```
$stream = fopen($url,'r');
$dest1 = fopen('first1k.txt','w');
stream_copy_to_stream($stream, $dest1, 1024);
file_put_contents('remainder.txt', $stream);
?>
```

SQL Interface for flat files!

Example

```
<?php
@exec('rm /tmp/lb.db');
$db = sqlite_open('/tmp/lb.db', 0666, $sqliteerror);
if ($db) {
  sqlite_query($db, 'CREATE TABLE foo (bar varchar(10))');
  sqlite_query($db, "INSERT INTO foo VALUES ('fnord')");

  $result = sqlite_query($db,'select bar from foo');
  $record = sqlite_fetch_array($result);
  echo "<pre>"; print_r($record); echo "</pre>";
} else {
  die ($sqliteerror);
}
?>
```

Output:

```
Array
(
    [0] => fnord
    [bar] => fnord
)
```

Same thing with the OO interface

Example

```
<?php
$db = new sqlite_db(':memory:');

$db->query("CREATE TABLE foo(c1 date, c2 time, c3
varchar(64))");
$db->query("INSERT INTO foo VALUES ('2002-01-02', '12:49:00',
NULL)");

$r = $db->query("SELECT * from foo");

$result = $r->fetch_array();
?>
```

With PHP5's solid XML support and improved internal OOP support, Web Services are now a natural fit for PHP. Most people seem to think of SOAP when we say Web Services, so here is the obligatory SOAP example:

```
<?php
$amazon_index = array(
  'DVD', 'Photo', 'Electronics', 'OfficeProducts',
'HealthPersonalCare',
  'Toys', 'Baby', 'VideoGames', 'MusicTracks',
'OutdoorLiving',
  'Blended', 'MusicalInstruments', 'Magazines',
'DigitalMusic',
  'Jewelry', 'Video', 'Tools', 'PCHardware', 'SportingGoods',
  'Classical', 'Software', 'Books', 'VHS', 'Wireless',
'Restaurants',
  'Music', 'GourmetFood', 'Miscellaneous', 'Kitchen',
'WirelessAccessories',
  'Merchants', 'Beauty', 'Apparel'
);

function amazon($index, $keywords, $timeout=7200) {
  $dest_file = "/tmp/aws_{$index}_".md5($keywords);
  if(file_exists($dest_file) && filemtime($dest_file) >
(time()-$timeout)) {
    $result = unserialize(file_get_contents($dest_file));
  } else {
    $aws = new SoapClient('http://webservices.amazon.com/'.

'AWSECommerceService/US/AWSECommerceService.wsdl',
                array("trace" => 1));
    $result = $aws->ItemSearch(array(
        'SubscriptionId'=>'XXXXXXXXXXXXX',
        'AssociateTag'=>'lerdorf-20',
        'Request'=>array(array('SearchIndex'=>$index,
                               'Keywords'=>$keywords))
      )
    );
    $tmpf = tempnam('/tmp','YWS');
    file_put_contents($tmpf, serialize($result));
    rename($tmpf, $dest_file);
  }
  return $result;
}
?>
```

```
<?php
function Add($x,$y) {
    return $x+$y;
}

$server = new
SoapServer(null,array('uri'=>"http://test-uri/"));
$server->addFunction("Add");
$server->handle();
?>
```

Or using a WSDL:

```
<?php
function Add($x,$y) {
    return $x+$y;
}

$server = new SoapServer("./add.wsdl"));
$server->addFunction("Add");
$server->handle();
?>
```

add.wsdl

```
<?xml version="1.0" ?>
<definitions
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:si="http://soapinterop.org/xsd"
 xmlns:tns="http://localhost/~rasmus/pecl/soap/test.wsdl"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="http://localhost/~rasmus/pecl/soap/test.wsdl">

 <types>
   <xsd:schema
targetNamespace="http://localhost/~rasmus/pecl/soap/test.wsdl">
     <xsd:import
namespace="http://schemas.xmlsoap.org/soap/encoding/" />
     <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"
/>
   </xsd:schema>
 </types>

 <message name="AddRequest">
   <part name="x" type="xsd:double" />
   <part name="y" type="xsd:double" />
 </message>
 <message name="AddResponse">
   <part name="result" type="xsd:double" />
 </message>

 <portType name="TestServicePortType">
   <operation name="Add">
     <input message="tns:AddRequest" />
     <output message="tns:AddResponse" />
   </operation>
 </portType>

 <binding name="TestServiceBinding"
type="tns:TestServicePortType">
   <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
   <operation name="Add">
     <soap:operation soapAction="Add" style="rpc" />
     <input>
```

```
       <soap:body use="encoded"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </input>
     <output>
       <soap:body use="encoded"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
   </operation>
  </binding>

 <service name="TestService">
   <port name="TestServicePort"
binding="tns:TestServiceBinding">
     <soap:address
location="http://localhost/~rasmus/pecl/soap/soap_server.php"/>
   </port>
 </service>

</definitions>
```

No more SOAP!

SOAP makes my head hurt. Let's look at some stuff we can all actually understand instead.

Take a pinch of RSS

```
<?php
$url = 'http://buzz.yahoo.com/feeds/buzzoverl.xml';
$xml = simplexml_load_file($url);
foreach($xml->channel->item as $item) {
  $ret[(string)$item->title] = (string)$item->link;
}
echo "<pre>"; print_r($ret); echo "</pre>";
?>
```

Output:

```
Array
(
    [1. Britney Spears] =>
http://search.yahoo.com/search?p=britney+spears&cs=bz
    [2. Boston Marathon] =>
http://search.yahoo.com/search?p=boston+marathon&cs=bz
    [3. 50 Cent] =>
http://search.yahoo.com/search?p=50+cent&cs=bz
    [4. Mariah Carey] =>
http://search.yahoo.com/search?p=mariah+carey&cs=bz
    [5. Jennifer Lopez] =>
http://search.yahoo.com/search?p=jennifer+lopez&cs=bz
    [6. Ultimate Fighting Championship] =>
http://search.yahoo.com/search?p=ultimate+fighting+championship&cs=bz
    [7. Ciara] => http://search.yahoo.com/search?p=ciara&cs=bz
    [8. Lindsay Lohan] =>
http://search.yahoo.com/search?p=lindsay+lohan&cs=bz
    [9. Green Day] =>
http://search.yahoo.com/search?p=green+day&cs=bz
    [10. Eminem] =>
http://search.yahoo.com/search?p=eminem&cs=bz
    [11. Star Wars] =>
http://search.yahoo.com/search?p=star+wars&cs=bz
    [12. Jesse McCartney] =>
http://search.yahoo.com/search?p=jesse+mccartney&cs=bz
    [13. Usher] =>
http://search.yahoo.com/search?p=usher&cs=bz
    [14. Gwen Stefani] =>
http://search.yahoo.com/search?p=gwen+stefani&cs=bz
    [15. Akon] => http://search.yahoo.com/search?p=akon&cs=bz
)
```

Add a spoonful of REST

```
<?php
$srv  =
'http://api.search.yahoo.com/ImageSearchService/V1/imageSearch';
foreach($ret as $key=>$link) {
  $url = $srv . "?query=$key&appid=RESTDemo";
  $obj = simplexml_load_file($url);
}
?>
```

A thimble of gradeschool math

```
x^2/a^2 + y^2/b^2 = 1
```

And we end up with something like this

http://buzz.progphp.com

By the way, throwing a cache layer between you and whatever remote service you are accessing tends to be a good idea. In our case we can do it like this:

```
<?php
$tmp = '/tmp/'.md5($q);
if(!file_exists($tmp) || filemtime($tmp) < (time()-7200)) {
   $stream = fopen($url,'r');
   $tmpf = tempnam('/tmp','YWS');
   file_put_contents($tmpf, $stream);
   fclose($stream);
   rename($tmpf, $tmp);
}
$obj = simplexml_load_file(file_get_contents($tmp));
?>
```

- Web Search
- Web Search with Context
- Creative Commons Web Search
- News Search
- Image Search
- Local Search
- Video Search
- Context Extraction Service
- Context Extraction Service

```
 Parameter       Value
Description
 appid     string (required) The application ID. See
Application IDs for more information.
 context   string (required) The context to extract terms from
(UTF-8 encoded).
 query     string            An optional query to help with
the extraction process.
```

Example

```
<?php
$context = "Italian sculptors and painters of the renaissance
favored the Virgin Mary for inspiration.";
$url =
'http://api.search.yahoo.com/ContentAnalysisService/V1/termExtraction';
$post =
"query=madonna&appid=RESTDemo&context=".rawurlencode($context);
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);
$xml = simplexml_load_string(curl_exec($ch));
curl_close($ch);
$config = array('indent'=>TRUE, 'indent-attributes'=>TRUE,
'wrap-attributes'=>TRUE,'input-xml'=>TRUE,
'output-xml'=>TRUE);
echo nl2br(str_replace('
',' ',htmlspecialchars(tidy_parse_string($xml->asXML(),$config))));
?>
```

Output:

```
&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?&gt;
&lt;ResultSet xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
            xmlns=&quot;urn:yahoo:
            xsi:schemaLocation=&qu

  &lt;Result&gt;italian sculptors&lt;/Result&gt;
  &lt;Result&gt;virgin mary&lt;/Result&gt;
  &lt;Result&gt;painters&lt;/Result&gt;
  &lt;Result&gt;renaissance&lt;/Result&gt;
  &lt;Result&gt;inspiration&lt;/Result&gt;
&lt;/ResultSet&gt;
&lt;!-- ws02.search.scd.yahoo.com uncompressed/chunked Tue Apr 
```

New Image Filtering

```
<?php
        header('Content-type: image/jpeg');
        $i = imagecreatefromjpeg('carl.jpg');
        imagefilter($i, IMG_FILTER_NEGATE);
        imagejpeg($i);
?>
```

Original

IMG_FILTER_NEGATE

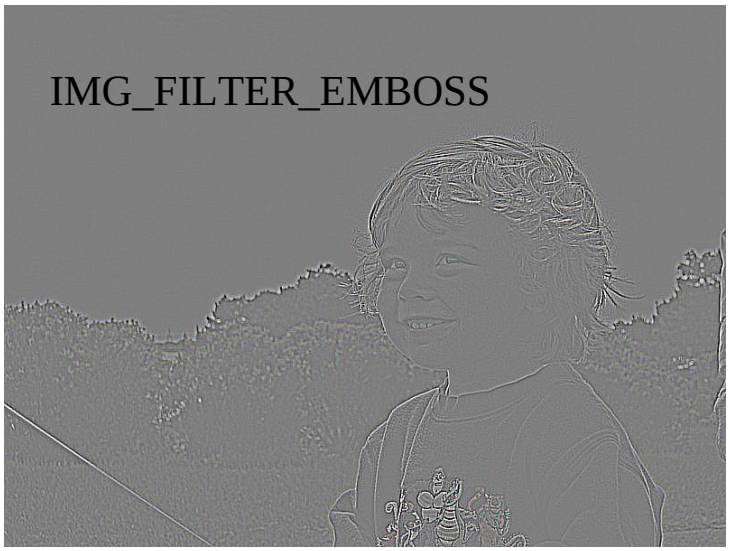

IMG_FILTER_GRAYSCALE

IMG_FILTER_BRIGHTNESS

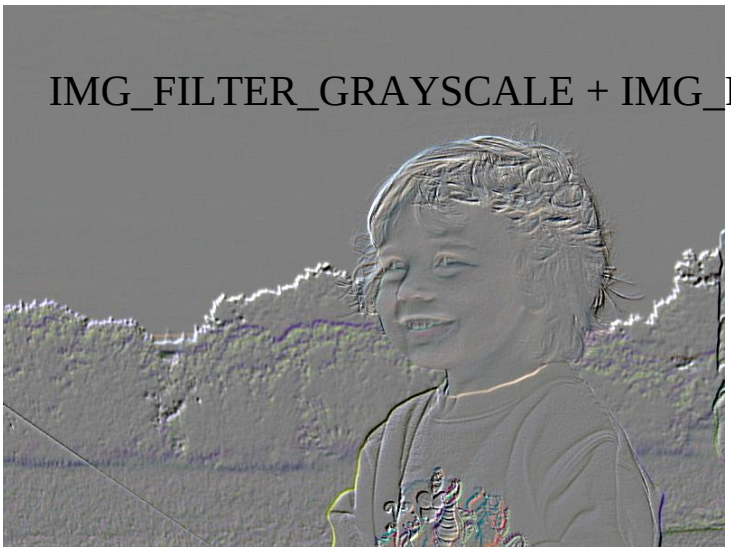
IMG_FILTER_CONTRAST


IMG_FILTER_COLORIZE


IMG_FILTER_EDGEDETECT


IMG_FILTER_EMBOSS

IMG_FILTER_GRAYSCALE + IMG_FILTER_EMBOSS




IMG_FILTER_GAUSSIAN_BLUR

IMG_FILTER_SELECTIVE_BLUR

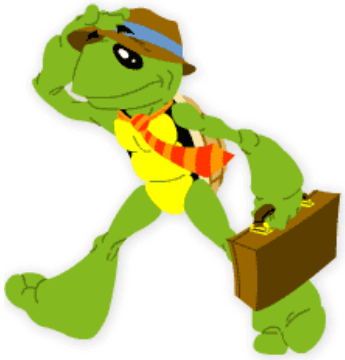

IMG_FILTER_MEAN_REMOVAL


IMG_FILTER_SMOOTH

Agenda

Security

Meet Dewie the turtle!



Brought to us by our friends at the FTC http://www.ftc.gov/infosecurity/

Worries

Indirect Attacks

* XSS - Cross-site scripting attacks

* Spoofing

Direct Attacks

* Buffer Overflows

* Path tricks

* Application logic attacks

Overflows

It is a bit difficult to talk about buffer overflows because we tend to fix them as soon as we discover them. But some overflows we have hit in the past include problems in:

- serialize/unserialize
- pack/unpack
- jpg algorithm
- exif header field overflows

There is also a current issue in most libc realpath() calls that is troublesome.

realpath man page

```
REALPATH(3)            FreeBSD Library Functions Manual
    REALPATH(3)

NAME
     realpath - returns the canonicalized absolute pathname

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <sys/param.h>
     #include <stdlib.h>

     char *
     realpath(const char *pathname, char
resolved_path[MAXPATHLEN]);

DESCRIPTION
     The realpath() function resolves all symbolic links,
extra ``/'' charac-
     ters and references to /./ and /../ in pathname, and
copies the resulting
     absolute pathname into the memory referenced by
resolved_path.  The
     resolved_path argument must refer to a buffer capable of
storing at least
     MAXPATHLEN characters.

     The realpath() function will resolve both absolute and
relative paths and
     return the absolute pathname corresponding to pathname.
All but the last
     component of pathname must exist when realpath() is
called.

RETURN VALUES
     The realpath() function returns resolved_path on success.
 If an error
     occurs, realpath() returns NULL, and resolved_path
contains the pathname
     which caused the problem.

ERRORS
     The function realpath() may fail and set the external
variable errno for
     any of the errors specified for the library functions
chdir(2), close(2),
     fchdir(2), lstat(2), open(2), readlink(2) and getcwd(3).

CAVEATS
     This implementation of realpath() differs slightly from
the Solaris
```

```
     implementation.  The 4.4BSD version always returns
absolute pathnames,
     whereas the Solaris implementation will, under certain
circumstances,
     return a relative resolved_path when given a relative
pathname.

SEE ALSO
     getcwd(3)

HISTORY
     The realpath() function call first appeared in 4.4BSD.
```

Imagine this C code

```
#define DOCROOT "/home/y/share/htdocs"
#define SCRIPT  "my_script"

int len = sizeof(DOCROOT)
          + sizeof(SCRIPT);
          + strlen(user_input);

char path = (char )malloc(len+1);
char safe_path[MAXPATHLEN];

snprintf(path, len, "s/s/%s", DOCROOT, user_input, SCRIPT);
if(realpath(path, safe_path)) {
  DIR *dir = opendir(safe_path);
  ...
}
```

Problem code in FreeBSD realpath.c

```
...
resolved_len = strlcat(resolved, next_token, PATH_MAX);
if (resolved_len >= PATH_MAX) {
    errno = ENAMETOOLONG;
    return (NULL);
}
...
```

It makes sure we never get a string back longer than PATH_MAX, but the silent strlcat truncation of the tokenized source path is a big problem!

Direct Attacks

Dumb things

Don't do dumb things!

```
<?php
    system($user_data);
?>

<?php
    include "$path/$user_data";
?>

<?php
    eval($user_data);
?>
```

Others

preg_replace with /e option, exec(), popen(), passthru, and backticks ``

Input Filtering - Current

Security in a web application boils down to always checking any user-supplied input data.

Exploits

- readfile($filename)
- system($cmd)
- file uploads into document_root
- XSS - Cross Site Scripting hacks

Input Filter hook

```
PHP_MINIT_FUNCTION(my_input_filter)
{
    sapi_register_input_filter(my_sapi_input_filter);
    return SUCCESS;
}
```

For a complete example, see README.input_filter in the PHP 5 source distribution. For PHP4, you will have to patch your source with http://lerdorf.com/php/input_filter.txt

Input Filtering - Future

Nobody is going to use that hook! People don't seem to understand how to filter their input. We'll need to spoonfeed again.

API?

```
<?php
    $name = filter(POST, 'name');  / Default filter /
    $age  = filter(POST, 'age', PFILT_INTEGER);
    $addr = filter(POST, 'addr', PFILT_TEXT, 'UTF-8');
?>
```

Config

```
input_filter.default = FILTER_TEXT
```

Strip or Escape?

```
abc;123{def}

abc 123 def
abc3B1237Bdef%7D
```

Agenda

Large Development Teams

Make sure your infrastructure is solid before you start

What is Large?

- 75 properties
- 25 international portals
- 15 Languages
- Hundreds of millions of registered users
- Literally billions of page views per day
- Hundreds of engineers spread around the world

Merger of 24 Web Companies

Net Controls	Four11	Classic Games
ViaWeb	WebCal	Yoyodyne
Sportacy	Hyperparallel	Log-Me-On
Geocities	Encompass	Online Anywhere
Broadcast.com	MyQuest	Arthas.com
eGroups	Kimo	Sold.com
Launch Media	HotJobs	Inktomi
Overture	3721	Kelkoo

Make that 27 now with FareChase, MusicMatch and Flickr

Development Tools

- Revision control - CVS, SVN, Perforce, Bitkeeper
- Mailing Lists
- Twikis, Personal Pages
- Bug tracking - Bugzilla
- Regression Testing - phpt, phpunit

PHPT is a simple test framework for PHP.

hello.phpt

```
--TEST--
Hello World test
--FILE--
<?php
    echo "Hello World";
?>
--EXPECT--
Hello World
```

filter.phpt

```
--TEST--
Input Filter test
--SKIPIF--
if(!extension_loaded('input_filter')) print "skip";
--POST--
a=<b>1</b>
--GET--
b=<i>2</i>
--FILE--
<?php
    echo $_POST['a'];
    echo $_GET['b'];
?>
--EXPECT--
12
```

phpt output

```
TIME START 2003-10-15 10:19:50
=====================================
PASS Hello World test [hello.phpt]
PASS Input Filter test [filter.phpt]

=====================================
TIME END 2003-10-15 10:19:50
=====================================
TEST RESULT SUMMARY
-------------------------------------
Number of tests :    2
Tests skipped   :    0 ( 0.0%)
Tests warned    :    0 ( 0.0%)
Tests failed    :    0 ( 0.0%)
Tests passed    :    2 (100.0%)
-------------------------------------
Time taken      :    0 seconds
=====================================
```

phpt failed test output

```
TIME START 2003-10-15 10:32:48
=====================================
PASS Hello World test [hello.phpt]
FAIL Input Filter test [filter.phpt]

=====================================
TIME END 2003-10-15 10:32:48
=====================================
TEST RESULT SUMMARY
-------------------------------------
Number of tests :    2
Tests skipped   :    0 ( 0.0%)
```

```
Tests warned    :    0 ( 0.0%)
Tests failed    :    1 (50.0%)
Tests passed    :    1 (50.0%)
-------------------------------------
Time taken      :    0 seconds
=====================================

=====================================
FAILED TEST SUMMARY
-------------------------------------
YIV test [filter.phpt]
=====================================
Some tests failed and a complete report has
been saved to /tmp/php_test_results_20031015.txt
```

failed test detailed output

```
=====================================================
/home/rasmus/t/filter.phpt
=====================================================
--TEST--
YIV test
--SKIPIF--
if(!extension_loaded('input_filter')) print "skip";
--POST--
a=<b>1</b>
--GET--
b=<i>2</i>
--FILE--
<?php
        echo $_POST['a'];
        echo $_GET['b'];
?>
--EXPECT--
1 2

=====================================================
---- EXPECTED OUTPUT
1 2
---- ACTUAL OUTPUT
12
---- FAILED
```

PHPT Sections

--TEST--	title of the test
--SKIPIF--	php code which prints "skip"
--POST--	POST variables passed to test script
--GET--	GET variables passed to test script
--INI--	each line contains an ini setting e.g. foo=bar
--FILE--	the test script
--EXPECT--	the expected output from the test script
--EXPECTF--	sscanf version of expected output
--EXPECTREGEX--	regex version of expected output

Agenda

High Complexity Applications?

Only if you don't build them right

Maximize Dev Resources

Be Lazy!

The greatest inefficiencies come from solving problems you will never have.

Problem Solving Exercise

Create a simple hit counter that meets the following requirements:

- Each Page has a separate counter
- Hits reporting should show hourly hit distribution over a 24-hour window

Where do we start? What do we need to solve this problem?

Problem Solving Exercise

Start Simple

```
<?php
  $cnt_file = '/tmp/'.basename(_FILE_).'.cnt';;
  $num = (int)@file_get_contents($cnt_file);
  $num++;
  file_put_contents($cnt_file, $num);
  echo "File has been accessed $num times";
?>
```

Output:

```
File has been accessed 7 times
```

Add Locking

```
<?php
  $cnt_file = '/tmp/'.basename(_FILE_).'.cnt';;
  $fp = fopen($cnt_file, 'r+');
  flock($fp, LOCK_EX);
  $num = (int)@fgets($fp);
  $num++;
  rewind($fp);
  fputs($fp, $num);
  flock($fp, LOCK_UN);
  fclose($fp);
  echo "File has been accessed $num times";
?>
```

Output:

```
File has been accessed 8 times
```

How About this?

```
<?php
  $cnt_file = '/tmp/'.basename(_FILE_).'.cnt';;
  $fp = fopen($cnt_file, 'a');
  fwrite($fp,'.');
  fclose($fp);
  $num = filesize($cnt_file);
  echo "File has been accessed $num times";
?>
```

Output:

```
File has been accessed 2 times
```

How About this?

```
<?php
  $cnt_file = '/tmp/'.basename(_FILE_).'.cnt';;
  $fp = fopen($cnt_file, 'a');
  fwrite($fp,chr(65+date('H')));
  fclose($fp);
  $num = filesize($cnt_file);
  echo "File has been accessed $num times";
?>
```

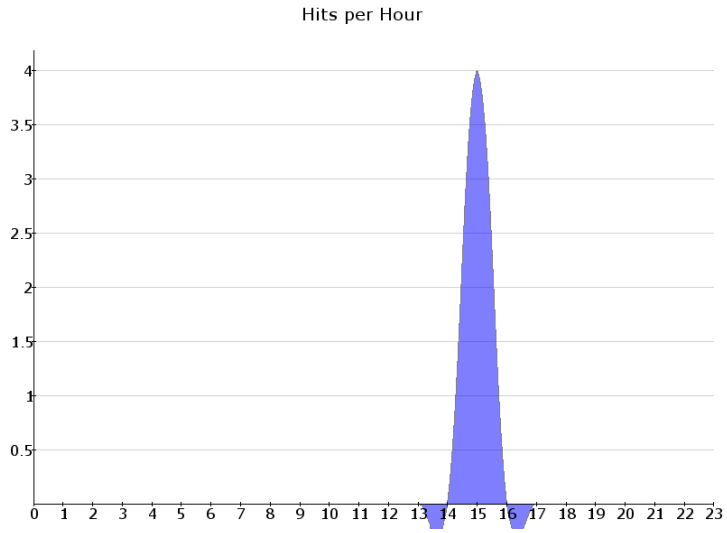
Output:

```
File has been accessed 2 times
```

Fancy Output

```
<?php
  $cnt_file = '/tmp/count5.php.cnt';;
  $fp = fopen($cnt_file, 'a');
  fwrite($fp,chr(65+date('H')));
  fclose($fp);
  $hits = file_get_contents($cnt_file);
  $total = strlen($hits);
  $distribution = count_chars($hits);
  $distribution = array_slice($distribution,65,24);

  require 'Image/Graph/Simple.php';
  $Graph = Image_Graph_Simple::factory(
    800, 600, 'Image_Graph_Plot_Smoothed_Area', $distribution,
    'Hits per Hour', 'gray', 'blue@0.5',
    array('/usr/share/fonts/Verdana.ttf',12)
  );
  $Graph->done();
?>
```

Output:



Design

Outside-in and Inside-out view

- What do outsiders (users) see looking in?
- What do insiders (developers) see looking out?

Outside-in

- The URL - The API to your Web App
- User interface flow

Inside-out

- File Layout
- Separation of layout from business logic
- Application API
- Server Architecture and Load Balancing

```
http://www.example.com/a/b?c=1&d=2
```

/a/b is what we are looking for

?c=1&d=2 are optional modifiers

$PATH_INFO

$PATH_INFO is your friend when it comes to creating clean URLS. Take for example this URL:

```
http://www.company.com/products/routers
```

If the Apache configuration contains this block:

```
<Location "/products">
  ForceType application/x-httpd-php
</Location>
```

Then all you have to do is create a PHP script in your DOCUMENT_ROOT named 'products' and you can use the $PATH_INFO variable which will contain the string, '/routers', to make a DB query.

ErrorDocument

Apache's ErrorDocument directive can come in handy. For example, this line in your Apache configuration file:

```
ErrorDocument 404 /error.php
```

Can be used to redirect all 404 errors to a PHP script. The following server variables are of interest:

- $REDIRECT_ERROR_NOTES - File does not exist: /docroot/bogus
- $REDIRECT_REQUEST_METHOD - GET
- $REDIRECT_STATUS - 404
- $REDIRECT_URL - /docroot/bogus

Don't forget to send a 404 status if you choose not to redirect to a real page.

```
<? Header('HTTP/1.0 404 Not Found'); ?>
```

Interesting uses

- Search for closest matching valid URL and redirect
- Use attempted url text as a DB keyword lookup
- Funky caching

Super-cool Dynamic Image Generator

Want to be cooler than all your friends? Well here it is!

First, set up an ErrorDocument 404 handler for your images directory.

```
<Directory /home/doc_root/images>
    ErrorDocument 404 /images/generate.php
</Directory>')
```

Then generate.php looks like this:

```
<?php
$filename = basename($_SERVER['REDIRECT_URL']);
if(preg_match('/^([^]?)([^]?)([^_]?)\.(.?)$/',$filename,
$reg)) {
    $type = $reg[1];
    $text = $reg[2];
    $rgb = $reg[3];
    $ext = $reg[4];
}

if(strlen($rgb)==6) {
    $r = hexdec(substr($rgb,0,2));
    $g = hexdec(substr($rgb,2,2));
    $b = hexdec(substr($rgb,4,2));
} else $r = $g = $b = 0;

    switch(strtolower($ext)) {
        case 'jpg':
            Header("Content-Type: image/jpg");
            break;
        case 'png':
        case 'gif': / We don't do gif - send a png instead /
            Header("Content-Type: image/png");
            break;
        default:
            break;
    }

    switch($type) {
        case 'solid':
            $im = imagecreatetruecolor(80,80);
            $bg = imagecolorallocate($im, $r, $g, $b);
            imagefilledrectangle($im,0,0,80,80,$bg);
            break;
        case 'button':
            $si = 32; $font = "php";
            $im = imagecreatefrompng('blank_wood.png');
            $tsize = imagettfbbox($si,0,$font,$text);
            $dx = abs($tsize[2]-$tsize[0]);
            $dy = abs($tsize[5]-$tsize[3]);
            $x = ( imagesx($im) - $dx ) / 2;
            $y = ( imagesy($im) - $dy ) / 2 + $dy;
            $white = ImageColorAllocate($im,255,255,255);
            $black = ImageColorAllocate($im,$r,$g, $b);
            ImageTTFText($im, $si, 0, $x, $y, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x+2, $y, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x, $y+2, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x+2, $y+2, $white, $font,
$text);
            ImageTTFText($im, $si, 0, $x+1, $y+1, $black, $font,
$text);
        break;
    }
    Header("HTTP/1.1 200 OK");
    $dest_file =
dirname($_SERVER['SCRIPT_FILENAME']).'/'.$filename;
    switch(strtolower($ext)) {
        case 'png':
```

```
            case 'gif':
                @ImagePNG($im,$dest_file);
                ImagePNG($im);
                break;
            case 'jpg':
                @ImageJPEG($im,$dest_file);
                ImageJPEG($im);
                break;
        }
    ?>
```

The URL, http://localhost/images/button_test_000000.png produces this image:

An interesting way to handle caching is to have all 404's redirected to a PHP script.

```
ErrorDocument 404 /generate.php
```

Then in your generate.php script use the contents of $REDIRECT_URI to determine which URL the person was trying to get to. In your database you would then have fields linking content to the URL they affect and from that you should be able to generate the page. Then in your generate.php script do something like:

```
<?php
    $s = $REDIRECT_URI;
    $d = $DOCUMENT_ROOT;
    // determine requested uri
    $uri = substr($s, strpos($s,$d) + strlen($d)+1);
    ob_start();  // Start buffering output
    // ... code to fetch and output content from DB
    $data = ob_get_contents();
    $fp = fopen("$DOCUMENT_ROOT/$uri",'w');
    fputs($fp,$data);
    fclose($fp);
    ob_end_flush(); // Flush and turn off buffering
?>
```

So, the way it works, when a request comes in for a page that doesn't exist, generate.php checks the database and determines if it should actually exist and if so it will create it and respond with this generated data. The next request for that same URL will get the generated page directly. So in order to refresh your cache you simply have to delete the files.

File Layout

Keep it simple!

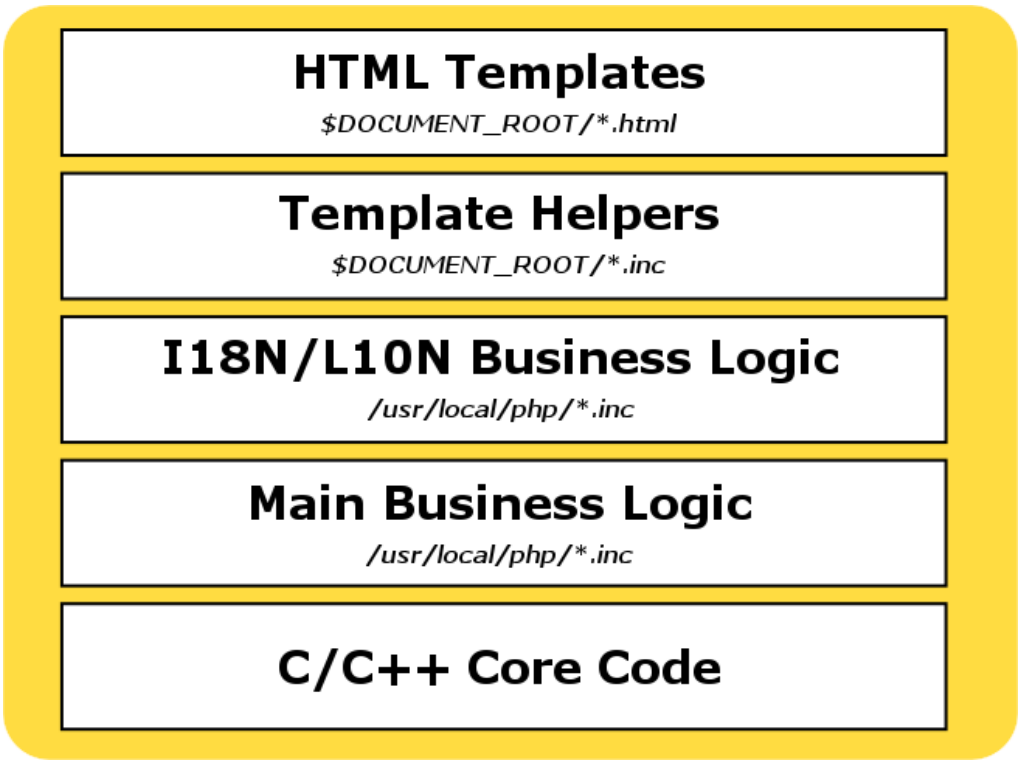
- All files (.html/.php/.inc) with HTML under $DOC_ROOT
- All non-HTML files, somewhere else

No direct access to .inc

```
<Files ~ "\.inc$">
    Order allow,deny
    Deny from all
</Files>
```

App Architecture

A suggested architecture for a PHP application. The template layer should have as little business logic as possible. As you go down you have less presentation and more business logic.



Simplistic Example

In terms of a real-world example of what goes in these 4 different layers, assume we are writing a database-backed application that needs to fetch a user record containing various fields. This is a very common thing to do in a web app. Our template layer might look something like this:

php.ini

```
auto_prepend_file = "./helpers.inc"
include_path = "/usr/local/lib/php"
```

Template Layer

```
<?title('Example Page')?>
<?greeting()?>
<h1>Heading 1</h1>
  <p>
    Page content
  </p>
<h1>Heading 2</h1>
  <p>
    Yada Yada
  </p>
<h1>Heading 3</h1>
  <p>
    Page content
  </p>
<?footer()?>
```

Template Helpers

```
<?php
  include "logic.inc";
  echo '<?xml version="1.0" encoding="UTF-8"?>';
?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<?php
  $user = get_user_record($_COOKIE['user_id']);
  function greeting() {
    global $user;
    echo "Hi " .$user['first_name'] ."!<br />\n";
    if($age=birthday($user)) {
      echo "Congratulations, today is your ";
      echo "$age birthday!<br />\n";
    }
  }
  function title($title) {
    echo "<head><title>$title</title></head>\n";
    echo "<body>\n";
  }
  function footer() {
    echo "</body>\n</html>";
  }
?>
```

Business Logic

```
<?php
  function get_user_record($id) {
    mysql_connect('localhost');
    mysql_select_db('users');
    $res = mysql_query("select * from users where id='$id'");
    if(!$res) echo mysql_error();
```

```
    else $row = mysql_fetch_array($res);
    return $row;
  }
  function birthday($user) {
    if(strftime('m d')==strftime('m d',$user['bday']))
      $age = strftime('Y') - strftime('Y',$user['bday']);
      if(($age100)>10 && ($age100)<20) $ap='th';
      else switch($age%10) {
        case 1: $ap = 'st'; break;
        case 2: $ap = 'nd'; break;
        case 3: $ap = 'rd'; break;
        default:  $ap = 'th'; break;
      }
      return $age.$ap;
    else
      return false;
  }
?>
```

In this case the final layer written in C contains the mysql_* functions, and the date() function. These happen to be standard PHP functions. If birthday() is called many times on every single request and since how you figure out if it is someone's birthday is unlikely to change very often, this may be a good candidate to translate into C. Although, in this example, the birthday function is probably too simple to see much of a performance improvement. On the other hand, other than a little bit of added parameter parsing, if you compare the C version of birthday() to the PHP version, it isn't that much harder to write it in C.

C Layer

```
PHP_FUNCTION(birthday)
{
  time_t timestamp, now;
  struct tm ta1, tmbuf1, ta2, tmbuf2;
  int age;
  char ret_age[8];

  if (zend_parse_parameters(1 TSRMLS_CC, "l", &timestamp) ==
FAILURE)
      return;

  ta1 = php_localtime_r(&timestamp, &tmbuf1);
  time(&now);
  ta2 = php_localtime_r(&now, &tmbuf2);

  if(tmbuf1.tm_mday==tmbuf2.tm_mday &&
tmbuf1.tm_mon==tmbuf2.tm_mon) {
      age = tmbuf2.tm_year - tmbuf1.tm_year;
      if((age100)>10 && (age100)<19)
sprintf(ret_age,"%dth",age);
      else switch(age % 10) {
          case 1: sprintf(ret_age,"%dst",age); break;
          case 2: sprintf(ret_age,"%dnd",age); break;
          case 3: sprintf(ret_age,"%drd",age); break;
          default:sprintf(ret_age,"%dth",age); break;
      }
  } else {
      RETURN_FALSE;
  }
  RETURN_STRING(ret_age,1);
}
```

For larger projects you want to be very careful when designing how the layers talk to each other. I like to provide a very clean API to the template layer and start each project by thinking about what the ideal template should look like.

Poll Example

Let's design a little application that allows people to answer a set of questions in various formats. We want to make it really easy for our content people to create new polls and we want to provide them with a flexible template API.

Poll Template

```
<?php
start_poll(1);
$os = array("FreeBSD","Linux","OSX","Windows","Other");
?>
<p class="purpose">
Please answer a couple of questions for us.
</p>

<p class="question">
1. What is your name?
</p>
<?php text_answer('name',64)?>

<p class="question">
2. Which operating systems do you use on a daily basis?
</p>
<?php select_any_of($os)?>

<p class="question">
3. Which operating system do you prefer?
</p>
<?php select_one_of($os)?>

<?php end_poll(); ?>
```

In our template helper layer we implement our frontend API.

Helper Layer

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <base href="http://php.net" />
    <title>Questionnaire</title>
    <style type="text/css" media="screen">@import
"/poll/style.css";</style>
  </head>
<body>
<h3>Questionnaire</h3>
<?php

require 'logic.inc';

function start_poll($poll, $qn=1) {
  global $at_least_one_vote;

  init($poll,    $qn);
  $at_least_one_vote = false;
  $self = getenv('REQUEST_URI');
  // consider adding a crumb here to prevent poll spoofing

echo <<<EOT
  <form action="$self" method="POST">
  <div class="main_box">
EOT;

}

function end_poll() {
  global $at_least_one_vote;

  if(!$at_least_one_vote) {
    echo '<div align="center"><input type="submit"
          value=" Submit Answers " /></div></form>';
  } else {
    echo '<div align="center"><input type="submit"
          value=" Change Answers "/></div></form>';
  }
  echo "</div\n";
}

function select_one_of($choices) {
  global $id, $qn, $at_least_one_vote;

  if(!is_array($choices))
    trigger_error("You must pass an array to select_one_of()",
E_USER_ERROR);

  // Grab the user's recorded answer to this question if any
  $answer  = voted($id,$qn);
  $total_votes_on_this_question = total_votes($id,$qn);
  $i='a';
  echo '<table class="results">';
  foreach($choices as $c) {
    $checked = $answer?in_array($i,$answer):false;
    echo "<tr><td><input type=\"radio\" name=\"q[$qn][]\"
          value=\"$i\" ".($checked?'CHECKED':'')."/> $c</td>";
    if($answer) {
      $at_least_one_vote = true;
      $percentage = 100*(($v=(int)results($id,$qn,$i))/
                    $total_votes_on_this_question);
      $img = ($percentage) ?
        ('<img  src="/poll/bline.gif" height="12" width="'.
        (int)$percentage.'" />') :
```

```
        '';
      echo "<td> $v of $total_votes_on_this_question (".
           sprintf(".1f",$percentage).")</td><td>
$img</td></tr>\n";
    } else {
      echo '</tr>';
    }
    $i++;
  }
  echo "</table>\n";
  $qn++;
}

function select_any_of($choices) {
  global $id, $qn, $at_least_one_vote;

  if(!is_array($choices))
    trigger_error("You must pass an array to select_any_of()",
E_USER_ERROR);
  // Grab the user's recorded answer to this question if any
  $answer  = voted($id,$qn);
  $total_votes_on_this_question = total_votes($id,$qn);
  $i='a';
  echo '<table class="results">';
  foreach($choices as $c) {
    $checked = $answer?in_array($i,$answer):false;
    echo "<tr><td><input type=\"checkbox\" name=\"q[$qn][]\"
          value=\"$i\" ".($checked?'CHECKED':'')."/> $c</td>";
    if($answer) {
      $at_least_one_vote = true;
      $percentage = 100*(($v=(int)results($id,$qn,$i))/
                    $total_votes_on_this_question);
      $img = ($percentage>0.0) ?
        ('<img  src="/poll/bline.gif" height="12" width="'.
        (int)$percentage.'" />') :
        '';
      echo "<td> $v of $total_votes_on_this_question (".
           sprintf(".1f",$percentage).")</td><td>
$img</td></tr>\n";
    } else {
      echo '</tr>';
    }
    $i++;
  }
  echo "</table>\n";
  $qn++;
}

function text_answer($field_name,$len) {
  global $id, $qn;

  $answer = get_text_answer($id,$qn);
  $size = (int)($len/2);
echo <<<EOT
  <table class="results">
  <tr><td>
  <input type="textfield" size="$size" maxlength="$len"
name="t[$qn]" value="$answer" />
  </td></tr>
  </table>
EOT;

  $qn++;
}?>
```

SQL Schema

```
CREATE TABLE poll_votes (
  user_name varchar(32) binary NOT NULL default '',
  poll_id smallint(6) NOT NULL default '0',
  question smallint(6) NOT NULL default '0',
  answer
set("","a","b","c","d","e","f","g","h","i","j","k","l","m","n","o",
             "p","q","r","s","t","u","v","w","x","y","z") NOT
NULL default "",
  text_answer varchar(255) binary NOT NULL default '',
  PRIMARY KEY (user_name,poll_id,question)
) TYPE=MyISAM;

CREATE TABLE poll_results (
  poll_id smallint(6) NOT NULL default '0',
  question smallint(6) NOT NULL default '0',
  answer char(1) NOT NULL default '',
  text_answer varchar(255) binary NOT NULL default '',
  votes int(11) NOT NULL default '0',
  INDEX (poll_id)
) TYPE=MyISAM;
```

Logic Layer

```
<?php

function db_connect() {
  mysql_pconnect("localhost","poll_user","password");
  mysql_select_db("polls");
}

function db_query($q) {
  $res = mysql_query($q);
  if(!$res) {
    trigger_error("db_query($q): ".mysql_error(),
E_USER_ERROR);
  }
  return $res;
}

function voted($poll, $question) {
  global $user;

  $res = db_query("select answer from poll_votes
                   where poll_id=$poll
                   and user_name='$user'
                   and question=$question");

  return mysql_num_rows($res) ? mysql_result($res,0):false;
}

function total_votes($poll, $question) {
  $res = db_query("select sum(votes) as total from
poll_results
                   where poll_id=$poll and
question=$question");

  return mysql_num_rows($res) ? mysql_result($res,0):false;
}

function results($poll, $question, $answer) {
  $res = db_query("select votes from poll_results
                   where poll_id=$poll
                   and question=$question
                   and answer=$answer");

  return mysql_num_rows($res) ? mysql_result($res,0):false;
}
```

```
function init($poll, $start_qn = 1) {
  $GLOBALS['user'] = $user = $_SERVER['PHP_AUTH_USER'];
  $GLOBALS['qn']   = $start_qn;
  $GLOBALS['id']   = $poll;

  db_connect();

  if(getenv('REQUEST_METHOD')=="POST") {
    // check bread crumb here
    foreach($_POST['q'] as $qn=>$answer) {
      if(($oanswer=voted($poll,$qn))!==false) {
        // if user already voted on this question, decrement
old vote
        $where = "where poll_id=$poll and question=$qn and
answer=$oanswer";
        db_query("update poll_results set votes=votes-1
$where");
      }

      // MySQL 4.1 specific query
      db_query("insert into poll_results
                values ($poll,$qn,$answer,1)
                on duplicate key update set votes=votes+1");

      // record user's [new] answer in poll_votes
      db_query("replace into poll_votes values
('$user',$poll,$qn,$answer)");
    }
  }
}
?>
```

Agenda

Debugging Applications

with Xdebug

- Protects against infinite recursion
- Colored var_dump()
- Stacktraces
- Functiontraces
- parameters to functions
- function/methodname, classname
- file:line locations
- memory allocation
- Profiling
- Script execution analysis
- "Remote" debugging

Protection

```
<?php
function a() {
    a();
}
a();
?>
```

```
xdebug.max_nesting_level=4
```

Fatal error: Maximum function nesting level of '4' reached, aborting! in **/home/httpd/html/test/xdebug/infinite.php** on line **3**

Call Stack

#	Function	Location
1	{main}()	/home/httpd/html/test/xdebug/infinite.php:0
2	a()	/home/httpd/html/test/xdebug/infinite.php:6
3	a()	/home/httpd/html/test/xdebug/infinite.php:3
4	a()	/home/httpd/html/test/xdebug/infinite.php:3

New var_dump()


```
array
  0 => null
  1 => 3.1415926
  2 =>
    array
      'dutch' =>
        object(locale)[1]
          public 'lang' => 'nl'
          public 'variation' => 'NL'
          public 'charsets' =>
            array
              0 => 'iso-8859-1'
              1 => 'iso-8859-15'
          private 'mb_supported' => false
          private 'self' =>
            &object(locale)[1]
          protected 'id' => 1
```

Local variables

```
; enable showing local vars in error messages
xdebug.show_local_vars=1
```

Warning: *chdir() [function.chdir]: No such file or directory (errno 2)* in /dat/docs/magazine/phpa/code/xdebug/listing2.php on line 34

Call Stack

#	Function	Location
1	{main}()	/dat/docs/magazine/phpa/code/xdebug/listing2.php:0
2	change_dir('foo123')	/dat/docs/magazine/phpa/code/xdebug/listing2.php:37
3	chdir ('/foo123')	/dat/docs/magazine/phpa/code/xdebug/listing2.php:34

Variables in local scope (#2)

Variable	Value
$path =	'/foo123'
$dir =	'foo123'

```
;show all GET variables in error messages
xdebug.dump.GET=*
; show the login and password POST variables
xdebug.dump.POST=login,password
; show the PHP's session cookie
xdebug.dump.COOKIE=PHPSESSID
```

Warning: ***chdir(): No such file or directory (errno 2)*** **in** ***/dat/docs/magazine/phpa/code/xdebug/listing2.php*** **on line** ***36***

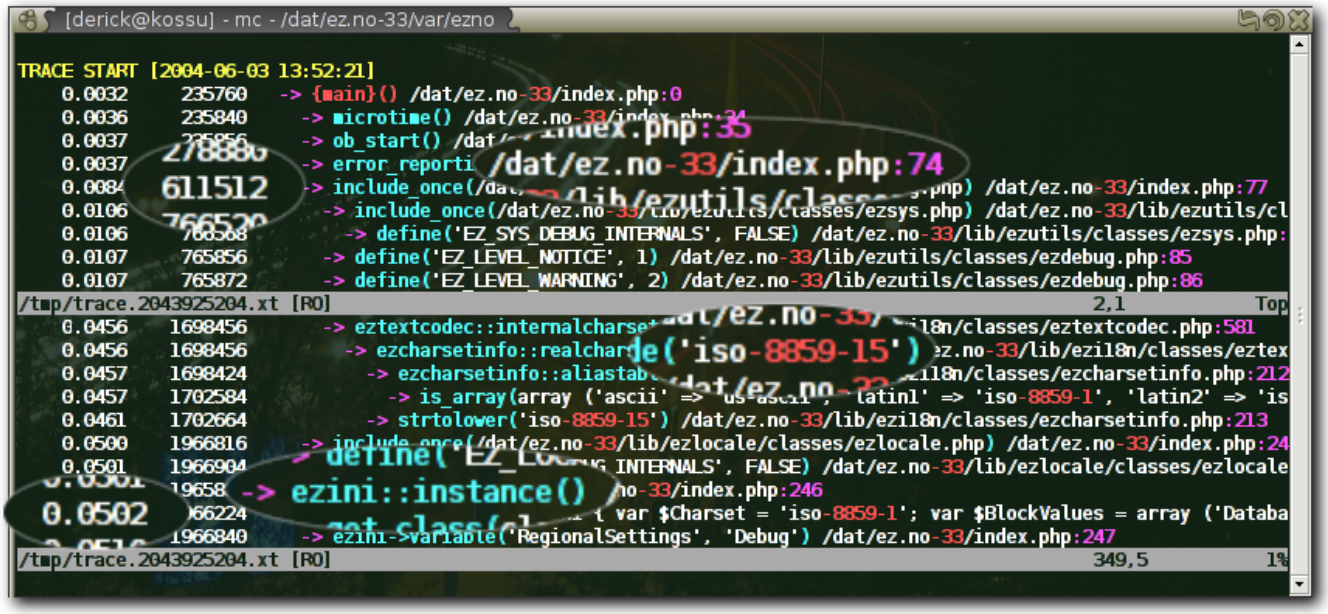
Call Stack

#	Function	Location
1	{main}()	/dat/docs/magazine/phpa/code/xdebug/listing2.php:0
2	change_dir('foo123')	/dat/docs/magazine/phpa/code/xdebug/listing2.php:39
3	chdir ('/foo123')	/dat/docs/magazine/phpa/code/xdebug/listing2.php:36

Dump ***$_GET***

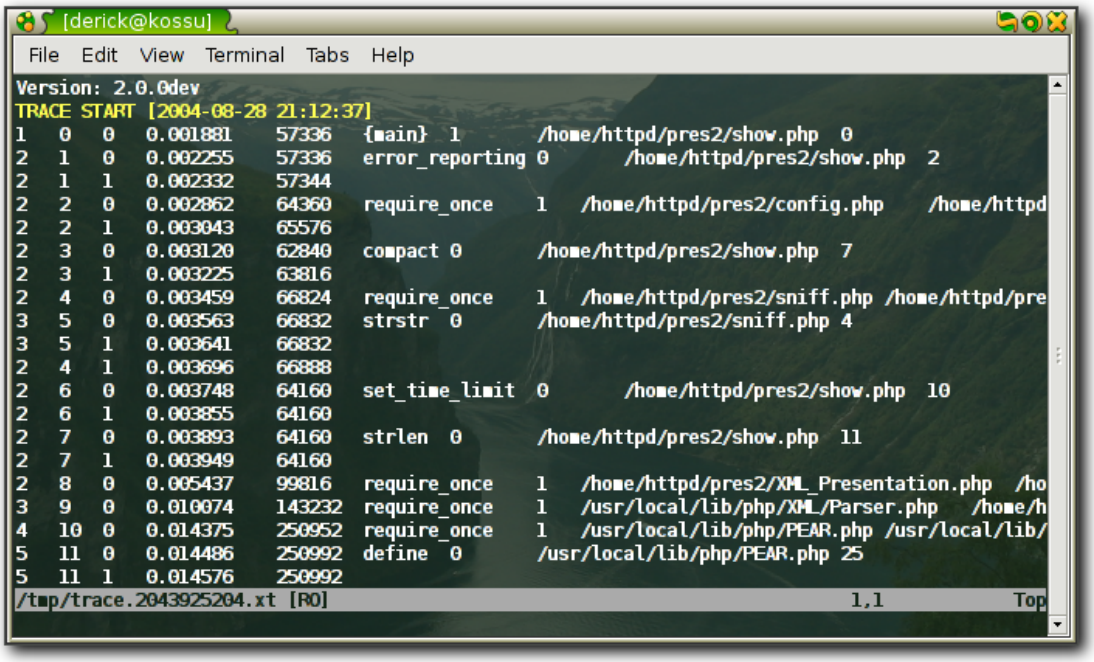
$_GET[42] =	'the answer to life, the universe and everything'

Execution trace to file

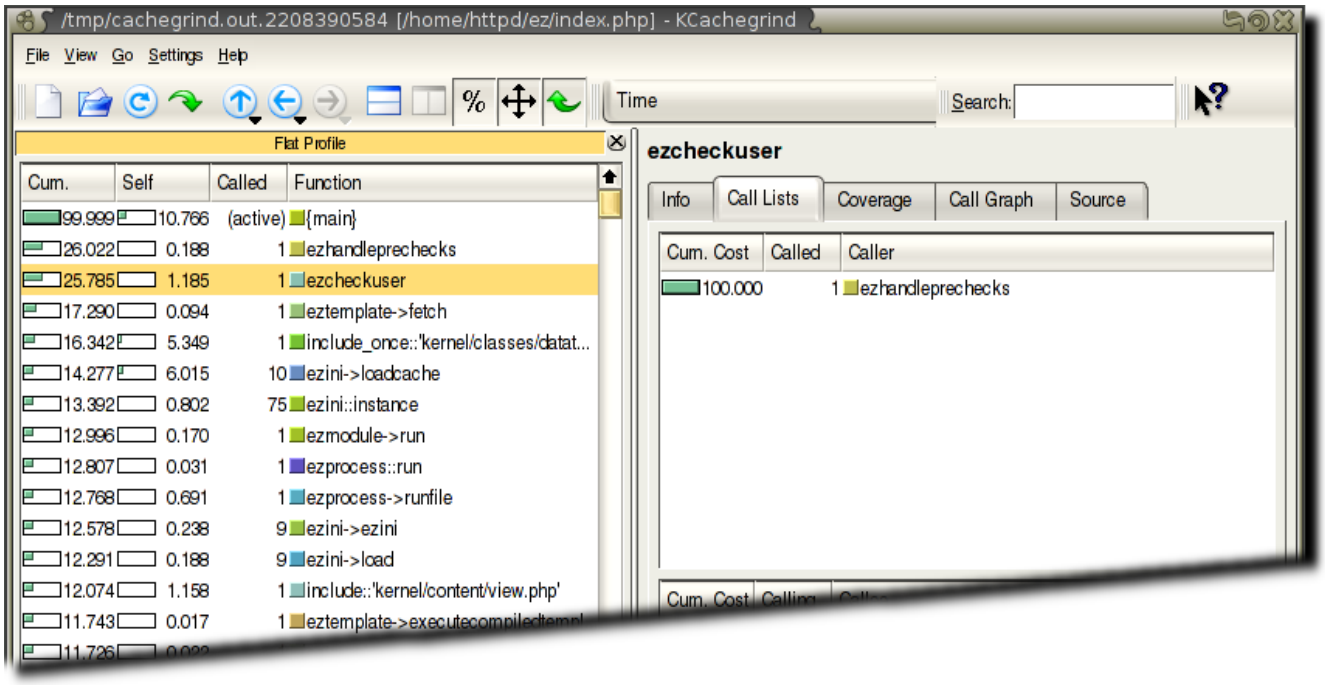



```
xdebug.auto_trace=1            ; enable tracing
xdebug.trace_output_dir=/tmp   ; trace output directory
xdebug.trace_output_name=crc32 ; tracefile extension
xdebug.collect_params=0        ; parameter collection
xdebug.collect_includes=1      ; collecting includes
xdebug.extended_info=0         ; extended code generation
xdebug.show_mem_delta=1        ; memory difference
```

Execution trace to file

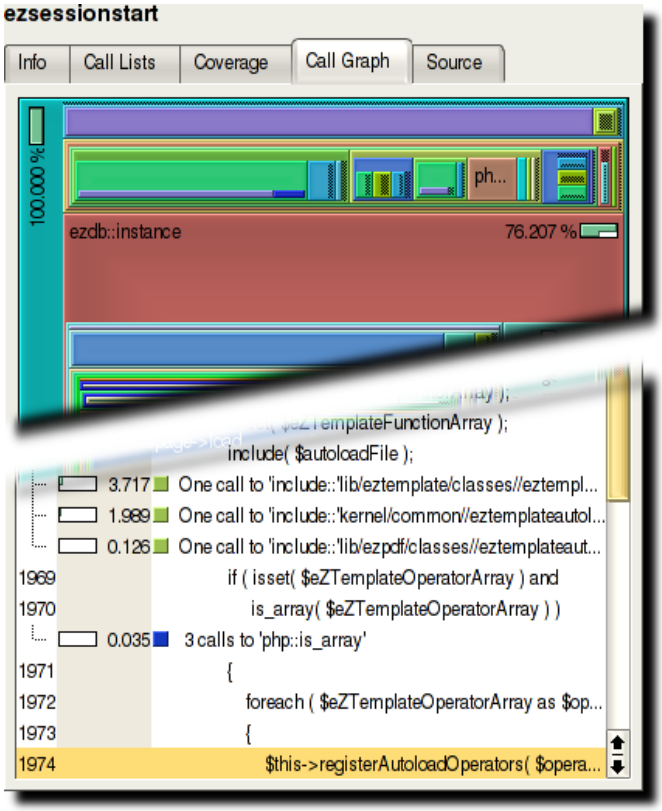



```
xdebug.auto_trace=1     ; enable tracing
xdebug.trace_format=1   ; selects computerized format
xdebug.trace_options=0  ; sets extra option (1 = append)
```




```
xdebug.profiler_enable=1          ; enable profiler
xdebug.profile_output_dir=/tmp    ; output directory
xdebug.profile_output_name=crc32  ; file extension
```

Profiling



- Call graph
- Area shows time spend
- Stacked to show callees
- Source annotations
- Number of calls
- Total time per function

Function to test (parse_time.php):

```
<?php
    function test_parse_time($f)
    {
        if (preg_match("/[0-9]{12}/", $f)) {
            $type = "YYYYMMDDHHii";
        } else if (preg_match("/[0-9]{8}\ [0-9]{4}/", $f)) {
            $type = "YYYYMMDD HHii";
        } else if (preg_match("/[0-9]{4}-[0-9]{2}-[0-9]{2}/",
$f)) {
            $type = "YYYY-MM-DD";
        } else {
            $type = "unknown";
        }

        echo "This is a $type format.\n";
    }
?>
```

Test driver (basic.t):

```
<?php

    include 'parse_time.php';

    $formats = array('2004-08-17 21:20', '20040817 2120');
    foreach ($formats as $format) {
        test_parse_time($format);
    }
?>
```

Testing App

```
<?php
  echo "Output:\n\n";
  xdebug_start_code_coverage();
  include 'presentations/slides/intro/tests/basic.t';
  $cc = xdebug_get_code_coverage();
  xdebug_stop_code_coverage();

  $clean = array();
  $clean[$test] = $cc[$test];

  foreach ($clean as $file => $cc) {
    echo "<pre>";
    $fc = file($file);
    $line_nos = array_keys($cc);
    foreach ($fc as $ln => $line) {
      if (in_array($ln + 1, $line_nos)) {
        $bgc = "aaaaaa";
      } else {
        $bgc = "dddddd";
      }
      echo "<div style='background-color: #$bgc'>";
      echo @sprintf("3d [2d]:\t", $ln + 1, $cc[$ln + 1]);
      echo htmlspecialchars($line);
      echo "</div>";
    }
    echo "</pre>";
  }
?>
```

Output:

```
Output:

This is a YYYY-MM-DD format.
This is a YYYYMMDD HHii format.
  1 [ 0]:    &lt;?php
  2 [ 1]:        function test_parse_time($f)
  3 [ 0]:        {
  4 [ 1]:            if (preg_match(&quot;/[0-9]{12}/&quot;, $f))
{
  5 [ 0]:                $type = &quot;YYYYMMDDHHii&quot;;
  6 [ 1]:            } else if (preg_match(&quot;/[0-9]{8}\
[0-9]{4}/&quot;, $f)) {
  7 [ 1]:                $type = &quot;YYYYMMDD HHii&quot;;
  8 [ 1]:            } else if
(preg_match(&quot;/[0-9]{4}-[0-9]{2}-[0-9]{2}/&quot;, $f)) {
  9 [ 1]:                $type = &quot;YYYY-MM-DD&quot;;
 10 [ 1]:            } else {
 11 [ 0]:                $type = &quot;unknown&quot;;
 12 [ 0]:            }
 13 [ 0]:
 14 [ 1]:            echo &quot;This is a $type format.\n&quot;;
 15 [ 1]:        }
 16 [ 0]:    ?&gt;
```

Remote Interfaces

- GDB debugger mode
- GDB like environment
- Two output possibilities: text and XML
- Available in Xdebug 1.3.x and 2.x
- DBGp debugger mode
- Language independent
- Plain text in, XML out
- Available in Xdebug 2.x

Activating the Remote Debugger

php.ini settings:

```
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_host=localhost
xdebug.remote_port=9000
xdebug.extended_info=1
```

On the shell:

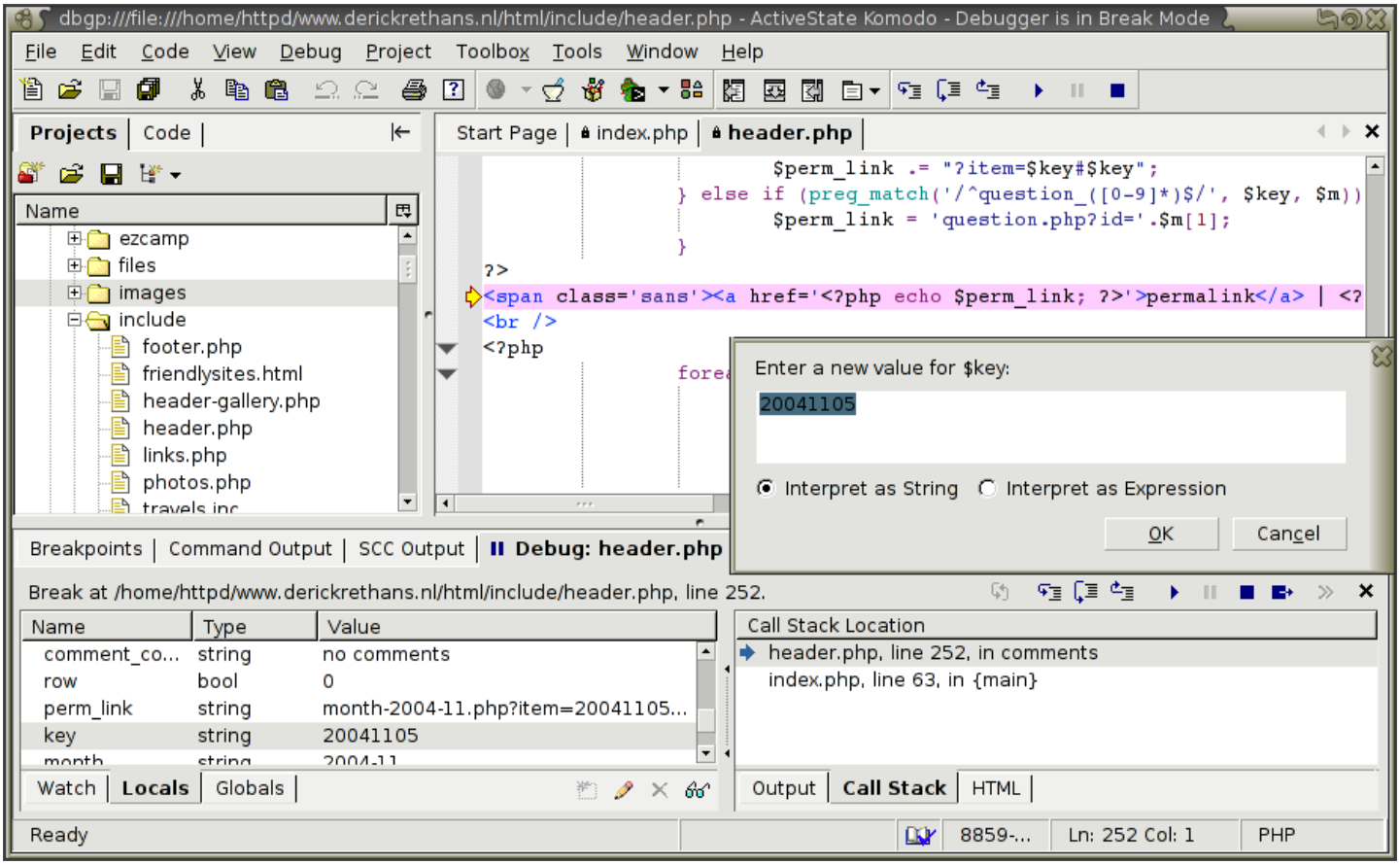
```
export XDEBUG_CONFIG="idekey=phpworks remote_enable=1"
```

With a browser:

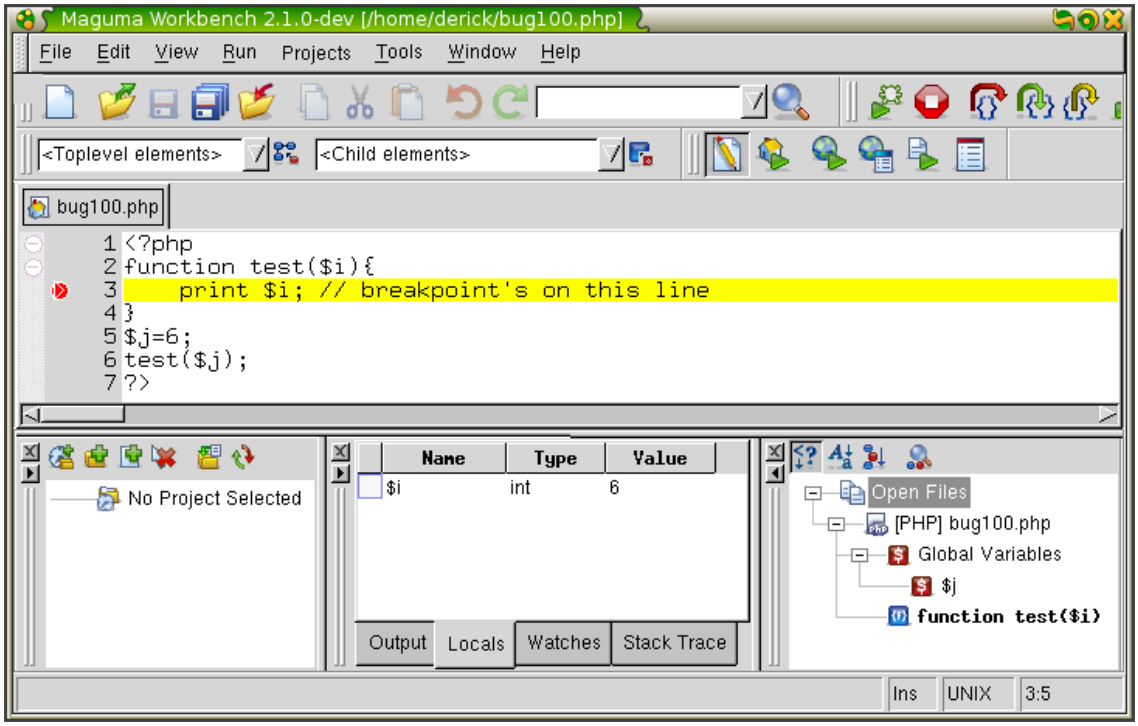
```
http://pres/show.php?XDEBUG_SESSION_START=phpworks2
```

Xdebug clients

ActiveState Komodo

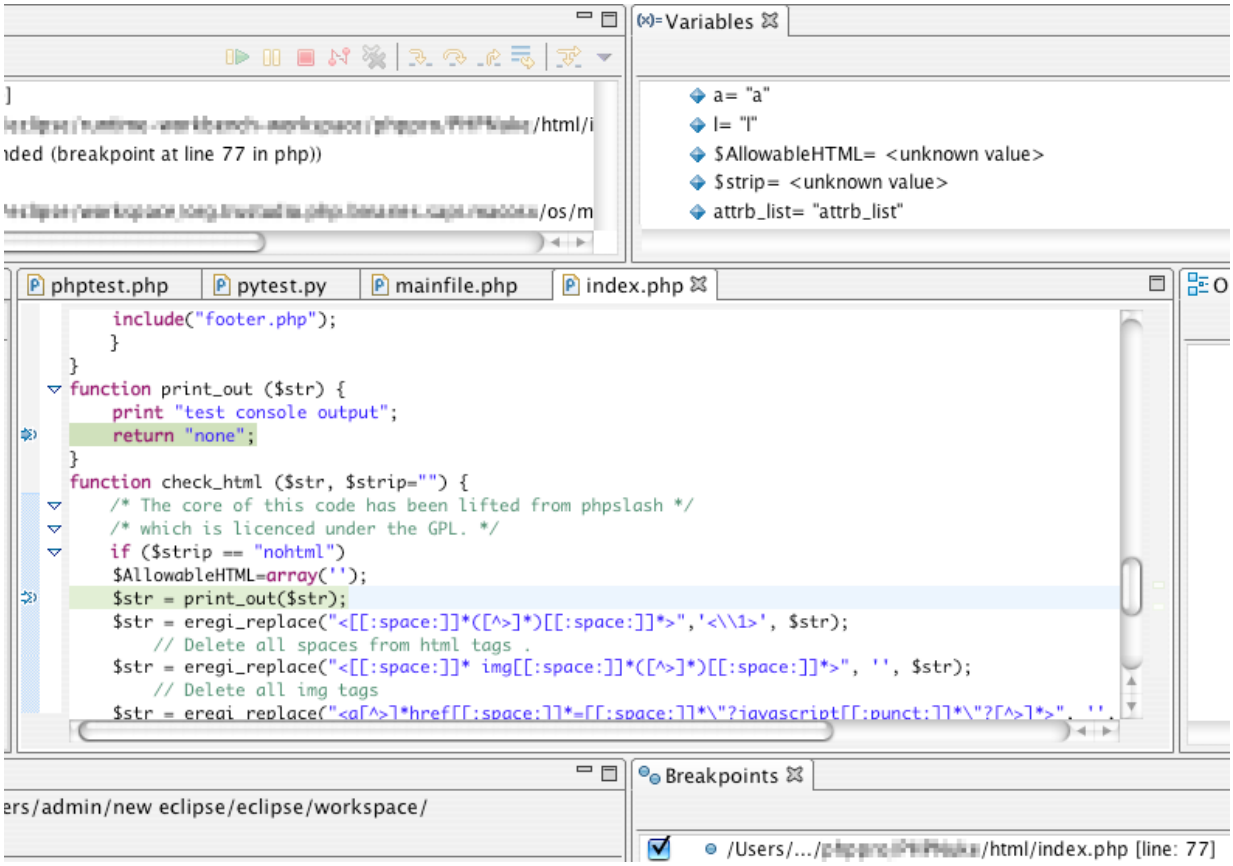


Maguma Workbench



Xdebug clients

TruStudio



Agenda

I18n and L10n

Why can't we all just speak a single 7-bit ASCII Language anyway?

Template System

- A mechanism to separate logic from layout.
- The defining characteristic is where and how the intersection between logic and layout is done.
- PHP is a general-purpose templating system.
- Any general-purpose templating system will eventually become PHP.

Templating

Runtime Templating

vs.

Offline Templating

PHP's strings are binary safe and mostly oblivious to character encodings and everything is fine if you just pass the data through. If you need to manipulate it you will need to do some work.

- setlocale
- utf8_decode
- iconv
- mbstring
- preg_* functions support UTF-8
- pecl/translit
- Full ICU-based Unicode Support eventually

php.ini

```
default_charset = "utf-8"
```

l10n

There isn't much localization support directly in PHP. A few things can help out.

- Generated templates
- setlocale
- gettext

Agenda

High Traffic Applications

Profile, Optimize and Scale it

Some simple guidelines

- Try to limit yourself to 5 or less includes per request
- Don't go overboard on OOP. Use where appropriate.
- Same goes for Layers. Abstraction, Indirection, abstract classes.
- Everything has a cost
- Use an opcode cache
- Watch your regular expressions!
- Cache! Cache! Cache!
- If you have plenty of CPU but limited bandwidth, turn on output compression

./configure

```
./configure --with-apxs=/usr/sbin/apxs --without-pic \
            --with-config-file-scan-dir=/etc/php
```

include_path

```
include_path = "/usr/share/pear:."
```

```
<?php
    include './template_helpers.inc';
    include 'business_logic.inc';
?>
```

open_basedir

```
open_basedir = "/usr/share/htdocs/:/usr/share/pear/"
```

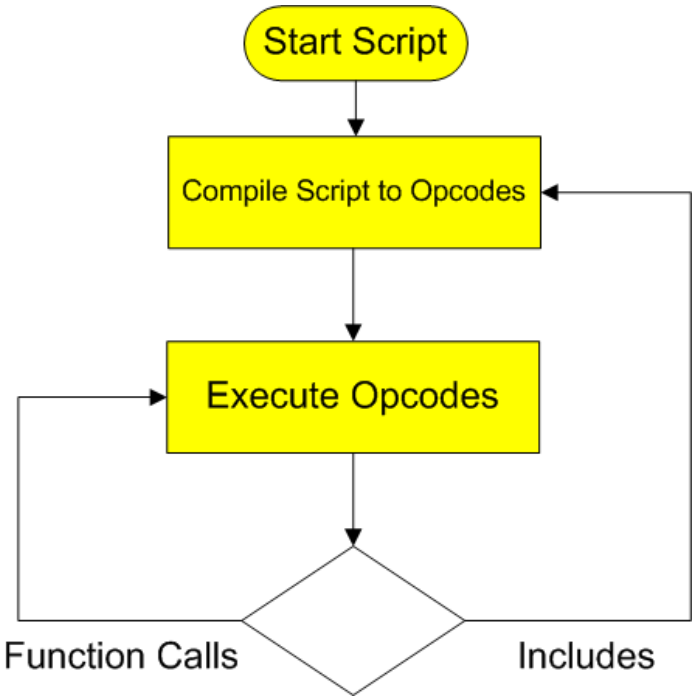
open_basedir checking adds some extra syscalls to every file operation. It can be useful, but it is rather expensive, so turn it off if you don't you need it.

variables_order

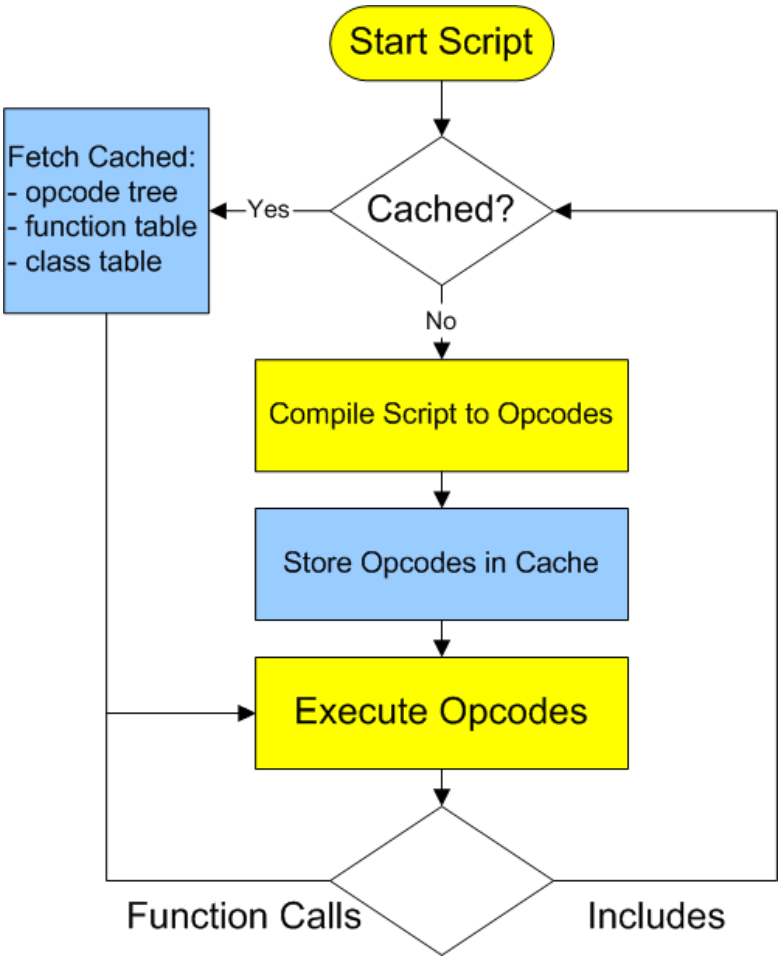
```
variables_order = "GPC"
```

```
<?php
    echo $_SERVER['DOCUMENT_ROOT'];
    echo getenv('DOCUMENT_ROOT');
?>
```

Standard PHP



PHP with an Opcode Cache



- Better memory handling when getting near the shared memory limit.
- user-level access to store PHP variables in shared memory easily.
- Quick way to load constants from shared memory

apc_store/fetch

```
<?php
  $a = array(1,2,3);
  apc_store('my_array', $a, 5);
  $a = 0;
  echo "<pre>"; var_dump($a); echo "</pre>";
  $b = apc_fetch('my_array');
  echo "<pre>"; var_dump($b); echo "</pre>";
?>

array
  0 =&gt; 1
  1 =&gt; 2
  2 =&gt; 3

array
  0 =&gt; 1
  1 =&gt; 2
  2 =&gt; 3
```

save/load constants

```
<?php
  if(!apc_load_constants('my_constants')) {
    $constants = array('ABC'=>'testing',
                       'DEF'=>123,
                       'FOO'=>'Bar');
    apc_define_constants('my_constants', $constants);
  }
?>
```

Let's have a look at how we might benchmark and subsequently tune a PHP server. I use http_load from acme.com to do the actual testing.

This is our benchmark script

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html><head><title>Simple PHP Benchmark</title></head>
<body>
<h1>String Manipulation</h1>
<p>
<?php
$str = 'This is just a silly benchmark that doesn\'t do
anything useful.';
$str .= 'Here we are just uppercasing the first two characters
of every word ';
$str .= 'in this long string';
$parts = explode(' ',$str);
foreach($parts as $part) {
    $new[] = strtoupper(substr($part,0,2)).substr($part,2);
}
echo implode(' ',$new);
?>
</p>
<p>
<h1>Including 3 files</h1>
<p>
<?php
@chdir("slides/intro/tb");
include 'presentations/slides/intro/tb/bench_config.inc';
include 'presentations/slides/intro/tb/bench_include1.inc';
include 'presentations/slides/intro/tb/bench_include2.inc';
?>
</p>
<h1>for-loop and calling a function many times</h1>
<p>
<?php
$a = range(1,200);
$b = range(200,1);
for($i=0; $i<200; $i++) {
    echo foo($a[$i],$b[$i]);
}
?>
</p>
<h1>Define and Instantiate an object and call some
methods</h2>
<p>
<?php
class test_class {
    var $test_var1;
    var $test_var2;
    var $test_var3;
    var $test_var4;

    function test_class() {
        $this->test_var1 = 111;
        $this->test_var2 = 222;
        $this->test_var3 = 333;
        $this->test_var4 = 444;
    }
    function get_props_sum() {
        return $this->test_var1 + $this->test_var2 +
$this->test_var3 + $this->test_var4;
    }
    function set_var1($value) {
        $this->test_var1 = $value;
        echo "test_var1 property set to $value<br />\n";
        return true;
    }
```

```
    function set_var2($value) {
        $this->test_var2 = $value;
        echo "test_var2 property set to $value<br />\n";
        return true;
    }
    function set_var3($value) {
        $this->test_var3 = $value;
        echo "test_var3 property set to $value<br />\n";
        return true;
    }
    function set_var4($value) {
        $this->test_var4 = $value;
        echo "test_var4 property set to $value<br />\n";
        return true;
    }
    function disp() {
        echo "<pre>";
        print_r($this);
        echo "</pre>";
    }
}

$obj = new test_class();
echo $obj->get_props_sum();
$obj->set_var1('test1');
$obj->set_var2(123);
$obj->set_var3($a);  / Array from previous test /
$obj->set_var4(array(1,2,3,4,5,6,7,8,9));
$obj->disp();
?>
<h1>And finally a bit of XML and XSLT</h1>
<?php
$xml =
domxml_open_file('presentations/slides/intro/menu.xml');
$xsl =
domxml_xslt_stylesheet_file('presentations/slides/intro/menu.xsl');
$out = $xsl->process($xml);
echo $out->dump_mem();
?>
</p>
</body>
</html>
```

bench_include1.inc

```
<h1>Some defines</h1>
<?php
define("MYMOD_VERSION","1.234-alpha");
for($i=0; $i<1000; $i++) {
    define(sprintf("MY_CONST_%04d",$i),$i);
}
echo MYMOD_VERSION;
echo MY_CONST_0001;
echo MY_CONST_0002;
echo MY_CONST_0003;
echo MY_CONST_0004;
echo MY_CONST_0005;
echo MY_CONST_0006;
echo MY_CONST_0007;
echo MY_CONST_0008;
echo MY_CONST_0009;
echo MY_CONST_0010;
echo MY_CONST_0011;
echo MY_CONST_0012;
echo MY_CONST_0012;
echo MY_CONST_0013;
echo MY_CONST_0014;
echo MY_CONST_0015;
echo MY_CONST_0016;
echo MY_CONST_0017;
echo MY_CONST_0018;
echo MY_CONST_0019;
```

```
echo MY_CONST_0020;
?>
```

bench_include1.inc

```
<?php
function foo($arg1, $arg2) {
    if($arg1>$arg2) return $arg1;
    elseif($arg1<$arg2) return $arg2;
    else return 'xxx';
}
?>
```

bench_include2.inc

```
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
This is just a bunch of plain text in an include file.<br />
```

menu.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<breakfast_menu>
 <food>
  <name>Belgian Waffles</name>
  <price>$5.95</price>
  <description>two of our famous Belgian Waffles with
             plenty of real maple syrup</description>
  <calories>650</calories>
 </food>
 <food>
  <name>Strawberry Belgian Waffles</name>
  <price>$7.95</price>
  <description>light Belgian waffles covered with
strawberries
             and whipped cream</description>
  <calories>900</calories>
 </food>
 <food>
  <name>Berry-Berry Belgian Waffles</name>
  <price>$8.95</price>
  <description>light Belgian waffles covered with an
assortment of
             fresh berries and whipped cream</description>
  <calories>900</calories>
 </food>
 <food>
  <name>French Toast</name>
  <price>$4.50</price>
  <description>thick slices made from our homemade sourdough
bread</description>
```

```
  <calories>600</calories>
 </food>
 <food>
  <name>Homestyle Breakfast</name>
  <price>$6.95</price>
  <description>two eggs, bacon or sausage, toast, and
               our ever-popular hash browns</description>
  <calories>950</calories>
 </food>
</breakfast_menu>
```

menu.xsl

```
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
    <xsl:for-each select="breakfast_menu/food">
      <div
style="background-color:teal;color:white;padding:4px">
        <span style="font-weight:bold;color:white">
          <xsl:value-of select="name"/>
        </span>
        - <xsl:value-of select="price"/>
      </div>
      <div
style="margin-left:20px;margin-bottom:1em;font-size:10pt">
        <xsl:value-of select="description"/>
        <span style="font-style:italic">
          (<xsl:value-of select="calories"/> calories per
serving)
        </span>
      </div>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Benchmarking Results

An http_load run

```
5000 fetches, 5 max parallel, 5.1645e+07 bytes, in 74.0229
seconds
10329 mean bytes/connection
67.5466 fetches/sec, 697689 bytes/sec
msecs/connect: 0.219601 mean, 1.106 max, 0.171 min
msecs/first-response: 72.1792 mean, 1209.1 max, 14.354 min
HTTP response codes:
  code 200 -- 5000
```

Whenever you do any sort of load testing, you need look beyond just the raw numbers and have a look at what your server is actually doing. Use vmstat:

Base PHP Load

```
6:33am riddler:~> vmstat 1
 procs      memory      page                    disks
faults      cpu
 r b w     avm    fre  flt  re  pi  po  fr  sr ad0 ad2   in
sy  cs us sy id
 5 1 0  175880  59292  491   0   0   0 408   0   0   0  858
7725 128 90 10  0
 5 1 0  175844  59296  481   0   0   0 414   0   0   6  869
7714 133 85 15  0
 5 1 0  175880  59256  492   0   0   0 414   0   0   0  873
7798 114 82 18  0
 5 1 0  175880  59248  472   0   0   0 402   0   2   1  848
7599 146 89 11  0
 5 1 0  175880  59232  479   0   0   0 408   0   0   0  858
7649 130 89 11  0
 5 1 0  176232  59244  479   0   0   0 414   0   1   0  863
7673 144 82 18  0
 5 1 0  171444  59256  790   0   0   0 672   0   3   0  826
7770 158 91  9  0
 5 1 0  171480  59216  478   0   0   0 402   0   0   0  843
7626 139 89 11  0
 5 1 0  171444  59224  480   0   0   0 414   0   0   0  862
7673 124 90 10  0
 5 1 0  171516  59168  491   0   0   0 408   0   0   0  846
7699 147 87 13  0
```

Our benchmark test was deliberately designed to have quite a bit of PHP processing and not a whole lot of output. 10k is somewhat small for a web page. If instead we have a whole lot of output, chances are we will be io-bound instead of cpu-bound. If you are io-bound, there is little sense in optimizing at the PHP level.

Evidence of an io-bound test

```
 procs      memory      page                    disks
faults      cpu
 r b w     avm    fre  flt  re  pi  po  fr  sr ad0 ac0   in
sy  cs us sy id
 4 0 0  132860  15724 1033   0   0   0   0   0   0   0 4457
954 3704  2 25 74
 5 0 0  132860  15724 1009   0   0   0   0   0   0   0 4436
714 3597  3 24 73
 6 0 0  132860  15716  980   0   0   0   0   0   0   0 4446
925 3603  5 23 72
 2 0 0  132860  15716 1028   0   0   0   0   0   6   0 4514
720 3696  2 24 73
 3 0 0  132472  15716 1018   0   0   0   0   0   2   0 4501
946 3673  2 22 76
```

```
 4 0 0  132472  15716 1039   0   0   0   0   0   0   0 4565
737 3718  2 26 73
 3 0 0  132472  15708 1010   0   0   0   0   0   2   0 4498
938 3639  2 24 75
 2 0 0  132472  15708 1012   0   0   0   0   0   0   0 4543
730 3665  5 25 70
```

Things to try if you are io-bound

```
[php.ini]
output_handler = ob_gzhandler


[Apache-1 httpd.conf]
LoadModule gzip_module lib/apache/mod_gzip.so

[Apache-2 httpd.conf]
LoadModule deflate_module lib/apache/mod_deflate.so
```

Benchmarking Results

So, we have determined we are cpu-bound and we need to go faster. What can we do? Some low-hanging fruit:

include_path

```
include_path = "/usr/share/pear:."
```

```
<?php
    include './template_helpers.inc';
    include 'business_logic.inc';
?>
```

open_basedir

```
open_basedir = "/usr/share/htdocs/:/usr/share/pear/"
```

open_basedir checking adds some extra syscalls to every file operation. It can be useful, but it is rather expensive, so turn it off if you don't think you need it.

variables_order

```
variables_order = "GPC"
```

```
<?php
    echo $_SERVER['DOCUMENT_ROOT'];
    echo getenv('DOCUMENT_ROOT');
?>
```

If you never use cookies, turn those off too

Output buffering?

```
output_buffering = On
```

Add an Opcode Cache

```
extension=apc.so
apc.shm_segments=1
apc.shm_size=64
apc.num_files_hint=1000
apc.mmap_file_mask=/tmp/apc.XXXXXX
```

Switch to semaphore locking in APC

```
CPPFLAGS=-I/usr/include/apache-1.3 \
./configure' --enable-apc --enable-mmap --enable-sem
```

Apply some of the hacks floating around

- http://lerdorf.com/php/syscall.txt
- http://lerdorf.com/php/non-pic.txt

Why Profile?

Because your assumptions of how things work behind the scenes are not always correct. By profiling your code you can identify where the bottlenecks are quantitatively.

How?

PECL to the rescue!

```
www:~> pear install apd
downloading apd-0.4p1.tgz ...
...done: 39,605 bytes
16 source files, building
running: phpize
PHP Api Version        : 20020918
Zend Module Api No     : 20020429
Zend Extension Api No  : 20021010
building in /var/tmp/pear-build-root/apd-0.4p1
running: /tmp/tmprFlAqf/apd-0.4p1/configure
running: make
apd.so copied to /tmp/tmprFlAqf/apd-0.4p1/apd.so
install ok: apd 0.4p1
```

Then in your php.ini file:

```
zend_extension = "/usr/local/lib/php/apd.so"
apd.dumpdir = /tmp
```

It isn't completely transparent. You need to tell the profiler when to start profiling. At the top of a script you want to profile, add this call:

```
<?php
apd_set_pprof_trace();
?>
```

The use the command-line tool called pprofp:

```
wwww: ~> pprofp
pprofp <flags> <trace file>
    Sort options
    -a          Sort by alphabetic names of subroutines.
    -l          Sort by number of calls to subroutines
    -m          Sort by memory used in a function call.
    -r          Sort by real time spent in subroutines.
    -R          Sort by real time spent in subroutines
(inclusive of child calls).
    -s          Sort by system time spent in subroutines.
    -S          Sort by system time spent in subroutines
(inclusive of child calls).
    -u          Sort by user time spent in subroutines.
    -U          Sort by user time spent in subroutines
(inclusive of child calls).
    -v          Sort by average amount of time spent in
subroutines.
    -z          Sort by user+system time spent in subroutines.
(default)

    Display options
    -c          Display Real time elapsed alongside call tree.
    -i          Suppress reporting for php builtin functions
    -O <cnt>    Specifies maximum number of subroutines to
display. (default 15)
    -t          Display compressed call tree.
    -T          Display uncompressed call tree.


% pprofp -z /tmp/pprof.48478
```

```
Trace for /home/y/share/htdocs/bench_main.php
Total Elapsed Time =    0.01
Total System Time  =    0.01
Total User Time    =    0.01


         Real         User        System              secs/
cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call
s/call  Memory Usage Name
--------------------------------------------------------------------------------------
100.0  0.00  0.00   0.01  0.01   0.01  0.01   200   0.0001
0.0001            0 foo
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->set_var2
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->set_var1
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->set_var3
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->set_var4
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 var_dump
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->disp
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 test_class->test_class
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 main
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    26   0.0000
0.0000            0 strtoupper
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    52   0.0000
0.0000            0 substr
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 implode
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     2   0.0000
0.0000            0 include
  0.0  0.01  0.01   0.00  0.00   0.00  0.00     2   0.0000
0.0000            0 range
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            0 explode


Trace for /home/rasmus/phpweb/index.php
Total Elapsed Time =    0.69
Total System Time  =    0.01
Total User Time    =    0.08

         Real         User        System              secs/
cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call
s/call  Memory Usage Name
--------------------------------------------------------------------------------------
 33.3  0.11  0.13   0.02  0.03   0.01  0.01     7   0.0043
0.0057       298336 require_once
 22.2  0.02  0.02   0.02  0.02   0.00  0.00   183   0.0001
0.0001       -33944 feof
 11.1  0.01  0.01   0.01  0.01   0.00  0.00     3   0.0033
0.0033       -14808 define
 11.1  0.04  0.04   0.01  0.01   0.00  0.00   182   0.0001
0.0001       112040 fgetcsv
 11.1  0.25  0.25   0.01  0.01   0.00  0.00     6   0.0017
0.0017         3768 getimagesize
 11.1  0.01  0.01   0.01  0.01   0.00  0.00    55   0.0002
0.0002         2568 sprintf
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     7   0.0000
0.0000         -136 printf
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000          136 htmlspecialchars
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000          -16 mirror_provider_url
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     7   0.0000
0.0000          112 spacer
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    10   0.0000
0.0000         -552 delim
```

```
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000           112 mirror_provider
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    20   0.0000
0.0000          -624 print_link
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000            24 have_stats
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000           -72 make_submit
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     2   0.0000
0.0000           112 strrchr
  0.0  0.08  0.08   0.00  0.00   0.00  0.00     2   0.0000
0.0000           168 filesize
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000
0.0000           -16 commonfooter
  0.0  0.00  0.11   0.00  0.00   0.00  0.00     2   0.0000
0.0000             0 download_link
  0.0  0.00  0.25   0.00  0.01   0.00  0.00     6   0.0000
0.0017           208 make_image
```

KCacheGrind Output

```
www:~> cd /tmp
www:/tmp> ls -lat pprof.*
-rw-r--r--  1 nobody 4294967295  62771 Apr 17 07:36
pprof.02466.0
-rw-r--r--  1 nobody 4294967295 269311 Apr 17 07:36
pprof.02465.0
-rw-r--r--  1 nobody 4294967295  62779 Apr 17 07:34
pprof.02464.0
www:/tmp> pprof2calltree -f pprof.02465.0
Writing kcachegrind compatible output to
cachegrind.out.pprof.02465.0
www:/tmp> kcachegrind cachegrind.out.pprof.02465.0
```

Profiling with XDebug

Function Summary Profile (sorted by total execution time)			
Total Time Taken	**Avg. Time Taken**	**Number of Calls**	**Function Name**
0.0012639508	0.0012639508	1	{include}
0.0009410545	0.0009410545	1	{include}
0.0003150582	0.0003150582	1	*hits
0.0002689241	0.0000179283	15	*url
0.0001970418	0.0000985209	2	mysql_query
0.0001859924	0.0001859924	1	mysql_pconnect
0.0001290126	0.0001290126	1	{include}
0.0001019851	0.0001019851	1	mysql_select_db
0.0000070533	0.0000070533	1	{include}
0.0000059746	0.0000059746	1	{include}
0.0000049799	0.0000049799	1	mysql_num_rows

Opcode Compiling	0.0709690896
Function Execution	0.0025330913
Ambient Code Execution	0.0031329459
Total Execution	0.0056660372
Total Processing	0.0766351268

- Basic profiling.
- xdebug_start_profile()
- xdebug_dump_function_profile(mode)

Benchmarking Results

After adding APC

```
5000 fetches, 5 max parallel, 5.1645e+07 bytes, in 66.1261
seconds
10329 mean bytes/connection
75.6131 fetches/sec, 781008 bytes/sec
msecs/connect: 0.21676 mean, 0.939 max, 0.174 min
msecs/first-response: 64.1843 mean, 1074.19 max, 12.774 min
HTTP response codes:
  code 200 -- 5000
```

Using apc_load_constants()

```
<?php
if(!apc_load_constants('bench_constants')) {
    $constants = array("MYMOD_VERSION"=>"1.234-alpha");
    for($i=0; $i<1000; $i++) {
        $constants[sprintf("MY_CONST_%04d", $i)] = $i;
    }
    apc_define_constants('bench_constants', $constants);
}
?>
```

With apc_load_constants()

```
5000 fetches, 5 max parallel, 5.1645e+07 bytes, in 46.0241
seconds
10329 mean bytes/connection
108.639 fetches/sec, 1.12213e+06 bytes/sec
msecs/connect: 0.225471 mean, 0.938 max, 0.174 min
msecs/first-response: 44.1477 mean, 805.834 max, 8.793 min
HTTP response codes:
  code 200 -- 5000
```

apc_load_constants() is ok, but writing your own simple PHP extension and using the Module Init (MINIT) hook is much better

MINIT Constants

```
PHP_MINIT_FUNCTION(mymod)
{
    int i;
    char s[] = "MY_CONST_XXXX";
    char num[] = "0000";

    REGISTER_STRING_CONSTANT("MYMOD_VERSION", "1.234-alpha",
                             CONST_CS | CONST_PERSISTENT);
    for(i=0; i<1000; i++) {
        sprintf(s+9,"%04d",i);
        REGISTER_LONG_CONSTANT(s, i, CONST_CS |
CONST_PERSISTENT);
    }
    return SUCCESS;
}?>
```

Using MINIT Instead

```
5000 fetches, 5 max parallel, 5.1645e+07 bytes, in 35.9252
seconds
10329 mean bytes/connection
139.178 fetches/sec, 1.43757e+06 bytes/sec
msecs/connect: 0.234849 mean, 7.323 max, 0.174 min
msecs/first-response: 34.366 mean, 706.119 max, 6.764 min
HTTP response codes:
  code 200 -- 5000
```

Replace DomXML

```
<?php
$xml = domxml_open_file('menu.xml');
$xsl = domxml_xslt_stylesheet_file('menu.xsl');
$out = $xsl->process($xml);
echo $out->dump_mem();
?>
```

With FastXSL

```
<?php
$xml = fastxsl_xml_parsefile('menu.xml');
$res = fastxsl_prmcache_transform('menu.xsl',$xml);
echo fastxsl_prmcache_tostring('menu.xsl',$res);
?>
```

FastXSL

```
5000 fetches, 5 max parallel, 5.16e+07 bytes, in 32.6279
seconds
10320 mean bytes/connection
153.243 fetches/sec, 1.58147e+06 bytes/sec
msecs/connect: 0.230745 mean, 1.294 max, 0.174 min
msecs/first-response: 30.8827 mean, 845.914 max, 6.218 min
HTTP response codes:
  code 200 -- 5000
```

PECL_Gen replaces ext_skel in PHP5

```
<?xml version="1.0"?>
<extension name="myext">
 <summary>My Cool Extension</summary>
 <logo src="myext.png" mimetype="image/png" />

 <maintainers>
  <maintainer>
   <user>rasmus</user>
   <name>Rasmus Lerdorf</name>
   <email>rasmus@php.net</email>
   <role>lead</role>
  </maintainer>
 </maintainers>

 <release>
  <version>0.1</version>
  <date>2005-04-01</date>
  <state>alpha</state>
  <license>php</license>
  <notes>
      - This is cool
  </notes>
 </release>

 <deps language="c" platform="unix">
  <header name="math.h"/>
  <lib    name="m" function="floor"/>
 </deps>

 <constant name="MYMODE" type="int" value="1" />
 <global name="foo" type="long" value="42"/>

 <function role="internal" name="MINIT">
  <code>init();</code>
 </function>

 <function name="get_foo">
  <proto>int get_foo(void)</proto>
  <code>
    RETURN_LONG(MYEXT_G(foo));
  </code>
 </function>

 <function name="foobar">
  <proto>string foobar(void)</proto>
  <summary>A dummy function</summary>
  <description>
   This function returns a static string.
  </description>
  <code>
   RETURN_STRING("foobar", 1);
  </code>
 </function>

</extension>

$ pecl-gen myext.xml

Creating 'myext' extension in '/home/rasmus/myext'
Your extension has been created in directory myext.
See myext/README for further instructions.
```

This will create the following directory:

```
myext
|-- EXPERIMENTAL
|-- README
|-- config.m4
```

```
|-- manual
|   `--myext
|       |-- configure.xml
|       |-- functions
|       `-- reference.xml
|-- package.xml
|-- php_myext.h
|-- tests
|-- myext.c
`-- myext.dsp

$ cd myext
$ phpize
$ ./configure
$ make
$ sudo make install
```

the pval/zval datatype

Data in PHP is stored in the pval or zval datatype. Two names for the same thing. They are defined like this:

```
typedef union _zvalue_value {
    long lval;          / long value /
    double dval;        / double value /
    struct {
        char *val;
        int len;
    } str;
    HashTable ht;      / hash table value */
    zend_object obj;
} zvalue_value;

struct _zval_struct {
    zvalue_value value; / value /
    zend_uchar type;    / active type /
    zend_uchar is_ref;
    zend_ushort refcount;
};

typedef struct _zval_struct zval;
typedef zval pval;
```

Convenience Macros

```
#define Z_LVAL(zval)        (zval).value.lval
#define Z_BVAL(zval)        ((zend_bool)(zval).value.lval)
#define Z_DVAL(zval)        (zval).value.dval
#define Z_STRVAL(zval)      (zval).value.str.val
#define Z_STRLEN(zval)      (zval).value.str.len
#define Z_ARRVAL(zval)      (zval).value.ht
#define Z_OBJ(zval)         &(zval).value.obj
#define Z_OBJPROP(zval)     (zval).value.obj.properties
#define Z_OBJCE(zval)       (zval).value.obj.ce
#define Z_RESVAL(zval)      (zval).value.lval

#define Z_LVAL_P(zval_p)        Z_LVAL(*zval_p)
#define Z_BVAL_P(zval_p)        Z_BVAL(*zval_p)
#define Z_DVAL_P(zval_p)        Z_DVAL(*zval_p)
#define Z_STRVAL_P(zval_p)      Z_STRVAL(*zval_p)
#define Z_STRLEN_P(zval_p)      Z_STRLEN(*zval_p)
#define Z_ARRVAL_P(zval_p)      Z_ARRVAL(*zval_p)
#define Z_OBJ_P(zval_p)         Z_OBJ(*zval_p)
#define Z_OBJPROP_P(zval_p)     Z_OBJPROP(*zval_p)
#define Z_OBJCE_P(zval_p)       Z_OBJCE(*zval_p)
#define Z_RESVAL_P(zval_p)      Z_RESVAL(*zval_p)

#define Z_LVAL_PP(zval_pp)      Z_LVAL(**zval_pp)
#define Z_BVAL_PP(zval_pp)      Z_BVAL(**zval_pp)
#define Z_DVAL_PP(zval_pp)      Z_DVAL(**zval_pp)
#define Z_STRVAL_PP(zval_pp)    Z_STRVAL(**zval_pp)
#define Z_STRLEN_PP(zval_pp)    Z_STRLEN(**zval_pp)
#define Z_ARRVAL_PP(zval_pp)    Z_ARRVAL(**zval_pp)
#define Z_OBJ_PP(zval_pp)       Z_OBJ(**zval_pp)
#define Z_OBJPROP_PP(zval_pp)   Z_OBJPROP(**zval_pp)
#define Z_OBJCE_PP(zval_pp)     Z_OBJCE(**zval_pp)
#define Z_RESVAL_PP(zval_pp)    Z_RESVAL(**zval_pp)','100%')
```

Parsing Parameters

In its simplest form you can call ZEND_NUM_ARGS() to check the number of parameters passed to your function.

```
PHP_FUNCTION(count_args)
{
    RETURN_LONG(ZEND_NUM_ARGS());
}
```

You can use the WRONG_PARAM_COUNT macro to output a generic wrong parameter count warning.

```
PHP_FUNCTION(my_func)
{
    if(ZEND_NUM_ARGS() > 1) {
        WRONG_PARAM_COUNT;
    }
    RETURN_TRUE;
}
```

Real Parameter Parsing

To do real parameter handling, use the zend_parse_parameters() function.

```
PHP_FUNCTION(return_arg)
{
    int argc = ZEND_NUM_ARGS();
    long arg;

    if (zend_parse_parameters(argc TSRMLS_CC, "l", &arg) ==
FAILURE)
        return;

    RETURN_LONG(arg);
}
```

You can make it an optional argument by putting a "|" in the argument format string. Like this:

```
PHP_FUNCTION(return_arg)
{
    int argc = ZEND_NUM_ARGS();
    long arg;

    if (zend_parse_parameters(argc TSRMLS_CC, "|l", &arg) ==
FAILURE)
        return;

    if(argc) RETURN_LONG(arg)
    else RETURN_FALSE
}
```

Here are all the possible modifiers in the argument format string:

```
'l'  long (integer)
'd'  double (floating point)
's'  string
'b'  boolean
'r'  resource
'a'  array
'o'  any object
'O'  specific object
'z'  mixed
```

So, for example, a function that takes a string, an integer, and array and an optional resource would look like this:

```
PHP_FUNCTION(complex_func)
{
    int argc = ZEND_NUM_ARGS();
    char *str = NULL;
    int str_len;
    int res_id = -1;
    long arg;
    zval *arr = NULL;
    zval *res = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sla|r", &str, &str_len,
                              &arg, &arr, &res) == FAILURE)
        return;

    if (res) {
        ZEND_FETCH_RESOURCE(???, ???, res, res_id, "???",
???_rsrc_id);
    }

    RETURN_FALSE;
}
```

Array Parameters

If your function takes an array parameter. You can check it and access it like this:

```
PHP_FUNCTION(my_arr)
{
    int argc = ZEND_NUM_ARGS();
    zval *tmp, arr = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC, "a", &arr) ==
FAILURE)
        return;

    zend_hash_internal_pointer_reset(Z_ARRVAL_P(arr));
    while (zend_hash_get_current_data(Z_ARRVAL_P(arr),
                                      (void **)&tmp) ==
SUCCESS) {
        convert_to_long_ex(tmp);
        php_printf("%ld\n",Z_LVAL_PP(tmp));
        zend_hash_move_forward(Z_ARRVAL_P(arr));
    }
    RETURN_TRUE;
}
```

Here the call to zend_parse_parameters() does the type check. If the user passes a non-array to this function s/he will get a message that looks like this:

```
foo.php(4) : Warning - my_arr() expects argument 1 to be
array, integer given
```

If you wanted to write a function that could take either an array of integers or perhaps a single integer and have your function figure it out automatically you do do it like this:

```
PHP_FUNCTION(my_arr)
{
    int argc = ZEND_NUM_ARGS();
    zval *tmp, arg = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC, "z", &arg) ==
FAILURE)
        return;

    switch(Z_TYPE_P(arg)) {
    case IS_ARRAY:
        zend_hash_internal_pointer_reset(Z_ARRVAL_P(arg));
        while (zend_hash_get_current_data(Z_ARRVAL_P(arg),
                                      (void **)&tmp) == SUCCESS)
{
            convert_to_long_ex(tmp);
            php_printf("%ld\n",Z_LVAL_PP(tmp));
            zend_hash_move_forward(Z_ARRVAL_P(arg));
        }
        break;
    case IS_STRING:
    case IS_LONG:
    case IS_DOUBLE:
        convert_to_long_ex(&arg);
        php_printf("%ld\n",Z_LVAL_P(arg));
        break;
    }
    RETURN_TRUE;
}
```

The possible types are:

```
#define IS_NULL     0
#define IS_LONG     1
#define IS_DOUBLE   2
#define IS_STRING   3
#define IS_ARRAY    4
#define IS_OBJECT   5
```

```
#define IS_BOOL     6
#define IS_RESOURCE 7
#define IS_CONSTANT 8
#define IS_CONSTANT_ARRAY   9
```

Returning Values

The simple types are returned using the following convenience macros:

```
#define RETVAL_RESOURCE(l)
ZVAL_RESOURCE(return_value, l)
#define RETVAL_BOOL(b)              ZVAL_BOOL(return_value, b)
#define RETVAL_NULL()               ZVAL_NULL(return_value)
#define RETVAL_LONG(l)              ZVAL_LONG(return_value, l)
#define RETVAL_DOUBLE(d)            ZVAL_DOUBLE(return_value,
d)
#define RETVAL_STRING(s, dup)       ZVAL_STRING(return_value,
s, dup)
#define RETVAL_STRINGL(s, l, dup)   ZVAL_STRINGL(return_value,
s, l, dup)
#define RETVAL_EMPTY_STRING()
ZVAL_EMPTY_STRING(return_value)
#define RETVAL_FALSE                ZVAL_BOOL(return_value, 0)
#define RETVAL_TRUE                 ZVAL_BOOL(return_value, 1)

#define RETURN_RESOURCE(l)          { RETVAL_RESOURCE(l);
return; }
#define RETURN_BOOL(b)              { RETVAL_BOOL(b); return;
}
#define RETURN_NULL()               { RETVAL_NULL(); return;}
#define RETURN_LONG(l)              { RETVAL_LONG(l); return;
}
#define RETURN_DOUBLE(d)            { RETVAL_DOUBLE(d);
return; }
#define RETURN_STRING(s, dup)       { RETVAL_STRING(s, dup);
return; }
#define RETURN_STRINGL(s, l, dup)   { RETVAL_STRINGL(s, l,
dup); return; }
#define RETURN_EMPTY_STRING()       { RETVAL_EMPTY_STRING();
return; }
#define RETURN_FALSE                { RETVAL_FALSE; return; }
#define RETURN_TRUE                 { RETVAL_TRUE; return; }
```

As you can see, returning a value from a function involves setting the special return_value zval to the return value and then just returning from the function call. At the C level, all functions return void.

Simple sum() function:

```
PHP_FUNCTION(my_sum)
{
    int argc = ZEND_NUM_ARGS();
    long arg1;
    long arg2;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "ll", &arg1, &arg2) == FAILURE)
        return;

    RETURN_LONG(arg1+arg2);
}
```

Here we uppercase every other character of a string.

```
PHP_FUNCTION(my_upper)
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    char *p;
    int i;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "s", &str, &str_len) == FAILURE)
        return;

    for(p=str, i=0; *p; p++,i++) {
```

```
        if(i%2) p = toupper(p);
    }
    RETURN_STRINGL(str,str_len,1);
}
```

Returning an Array

Returning an array is a little bit more complex. Here we take a string and a count and return an array with the specified number of elements containing the given string.

```
PHP_FUNCTION(my_array)
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    long count;
    int i;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sl", &str, &str_len, &count) ==
FAILURE)
        return;

    array_init(return_value);
    for(i=0; i<count; i++) {
        add_index_stringl(return_value, i, str, str_len, 1);
    }
}
```

So, to return an array you first call array_init() on the return_value zval and then populate the array by calling the various add_* functions. The functions available to populate an array are:

```
add_index_long(zval *arg, uint idx, long n)
add_index_null(zval *arg, uint idx)
add_index_bool(zval *arg, uint idx, int b)
add_index_resource(zval *arg, uint idx, int r)
add_index_double(zval *arg, uint idx, double d)
add_index_string(zval arg, uint idx, char str, int dup)
add_index_stringl(zval arg, uint idx, char str, uint length,
int dup)
add_index_zval(zval arg, uint index, zval value)
```

Recall that PHP supports the non-indexed array syntax. eg.

```
<? $a[] = 1; ?>
```

This same concept is available in the extension API using these functions:

```
add_next_index_long(zval *arg, long n)
add_next_index_null(zval *arg)
add_next_index_bool(zval *arg, int b)
add_next_index_resource(zval *arg, int r)
add_next_index_double(zval *arg, double d)
add_next_index_string(zval arg, char str, int dup)
add_next_index_stringl(zval arg, char str, uint length, int
dup)
add_next_index_zval(zval arg, zval value)
```

You can also create an array with non-numeric indices. Otherwise known as an associative array using these functions:

```
add_assoc_long(zval arg, char key, long n)
add_assoc_null(zval arg, char key)
add_assoc_bool(zval arg, char key, int b)
add_assoc_resource(zval arg, char key, int r)
add_assoc_double(zval arg, char key, double d)
add_assoc_string(zval arg, char key, char *str, int dup)
add_assoc_stringl(zval arg, char key, char *str, uint length,
int dup)
add_assoc_zval(zval arg, char key, zval *value)
```

To return a 2-dimensional array, you simply make new arrays and add them as elements of the top array. Like this:

```
PHP_FUNCTION(my_array)
```

```
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    long count;
    int i;
    pval *tmp_arr;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sl", &str, &str_len,
                &count) == FAILURE)
        return;

    array_init(return_value);
    for(i=0; i<count; i++) {
        MAKE_STD_ZVAL(tmp_arr);
        array_init(tmp_arr);
        add_assoc_long_ex(tmp_arr,
                      str, str_len,
              count+i);
        add_next_index_zval(return_value, tmp_arr);
    }
}
```

Note the use of add_assoc_long_ex() in the above example. If we know the length of the key, then this is slightly faster than calling add_assoc_long(). The macro definition shows why:

```
int add_assoc_long_ex(zval arg, char key, uint key_len, long
n);
#define add_assoc_long(__arg, __key, __n) \\
        add_assoc_long_ex(__arg, __key, strlen(__key)+1, __n)
```

Object Factory Approach

This gets quite a bit more complex than returning simple types, or even arrays. The first thing to do is to declare an extension-wide global class ptr along with the methods available in this object you are creating.

```
static zend_class_entry *my_class_entry_ptr;
static zend_function_entry php_my_class_functions[] = {
    PHP_FE(add,  NULL)
    PHP_FALIAS(del,           my_del,     NULL)
    PHP_FALIAS(list,          my_list,    NULL)
    {NULL, NULL, NULL}
};
```

Next, in our module_init function we initialize and register our class definition.

```
PHP_MINIT_FUNCTION(test) {
    zend_class_entry my_class_entry;

    INIT_CLASS_ENTRY(my_class_entry, "my_class",
php_my_class_functions);
    my_class_entry_ptr =
zend_register_internal_class(&my_class_entry TSRMLS_CC);
    return SUCCESS;
}
```

And then we create our object factory function which will return an instantiated object.

```
PHP_FUNCTION(my_object) {
{
    char *fn = NULL;
    int argc = ZEND_NUM_ARGS();
    int fn_len;
    long length;
    double price;
    zend_bool setting;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sbld", &fn, &fn_len,
                              &setting, &length, &price) ==
FAILURE)
        return;

    object_init_ex(return_value, my_class_entry_ptr);

    add_property_stringl(return_value, "filename", fn, fn_len,
1);
    add_property_bool   (return_value, "toggle", setting ? 0 :
1);
    add_property_long   (return_value, "length", length);
    add_property_double (return_value, "price", price);
}
```

From user space it would be called like this:

```
<?php
  $obj = my_object();
  $obj->add(...);
?>
```

If you want to have the more traditional class instantiation syntax and want to have a constructor run for it. eg.

```
<?php
  $obj = new my_class();
?>
```

Then create a constructor function:

```
PHP_FUNCTION(my_class) {
```

```
    char *fn = NULL;
    int argc = ZEND_NUM_ARGS();
    int fn_len;
    long length;
    double price;
    zend_bool setting;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sbld", &fn, &fn_len,
                              &setting, &length, &price) ==
FAILURE)
        return;

    add_property_stringl(this_ptr, "filename", fn, fn_len, 1);
    add_property_bool(this_ptr, "toggle", setting?0:1);
    add_property_long(this_ptr, "length", length);
    add_property_double(this_ptr, "price", price);
}
```

Note the use of this_ptr here.

If you want to access properties from within one of the methods, you can do this:

```
pval **tmp;
if(zend_hash_find(HASH_OF(this_ptr), "my_property", 11, (void
**)&tmp) == SUCCESS) {
    convert_to_string_ex(tmp);
    php_printf("my_property is set to %s\n",
Z_STRVAL_PP(tmp));
}
```

SAPI Globals

SAPI is the Server abstraction API and you can access server-related global variables using the SG() macro. Include SAPI.h in your .c file to get the SG macro. You can then access the elements of the sapi_globals_struct which looks like this:

```
typedef struct _sapi_globals_struct {
    void *server_context;
    sapi_request_info request_info;
    sapi_headers_struct sapi_headers;
    int read_post_bytes;
    unsigned char headers_sent;
    struct stat global_stat;
    char *default_mimetype;
    char *default_charset;
    HashTable *rfc1867_uploaded_files;
    long post_max_size;
    int options;
} sapi_globals_struct;
```

To access the default MIME type, you would use:

```
SG(default_mimetype)
```

And to access the request_uri you would use:

```
SG(request_info).request_uri
```

Executor Globals

These are run-time globals. The ones you are likely to be interested in are the symbol_table and active_symbol_table globals. For example, to access the user-space $foo variable from the current global context, you would do this:

```
pval **tmp;
if(zend_hash_find(&EG(symbol_table),
                  "foo", 4, (void **)&tmp) == SUCCESS) {
  RETURN_STRINGL(Z_STRVAL_PP(tmp), Z_STRLEN_PP(tmp), 1);
} else {
  RETURN_FALSE;
}
```

Memory Management

PHP has its own memory management functions which adds a safety net in case you forget to free things. It also helps catch overflows and other memory-related issues. The functions are exactly like their typical C counterparts:

```
emalloc
efree
estrdup
estrndup
ecalloc
erealloc
```

Resources

A generic data container. Used to hold file handles, database connections, or anything else that needs to be initialized and then passed around for other functions to act on. To create a resource in your extension, first declare an extension-wide true global. We don't have to worry about thread-safety here. And then if your resource holds a complex datatype of some sort, typedef it as well at the top of your .c file (on in your header file):

```
static int le_test;

typedef struct _test_le_struct {
    char *name;
    long age;
} test_le_struct;
```

Next we need to create a resource destructor function.

```
static void _php_free_test(zend_rsrc_list_entry *rsrc
TSRMLS_DC) {
    test_le_struct test_struct = (test_le_struct )rsrc->ptr;

    efree(test_struct->name);
    efree(test_struct);
}
```

Then in our MINIT function we declare/initialize our resource:

```
le_test = zend_register_list_destructors_ex(_php_free_test,
                                            NULL, "test",
module_number);
```

Then we might have a function that looks like this that returns a resource:

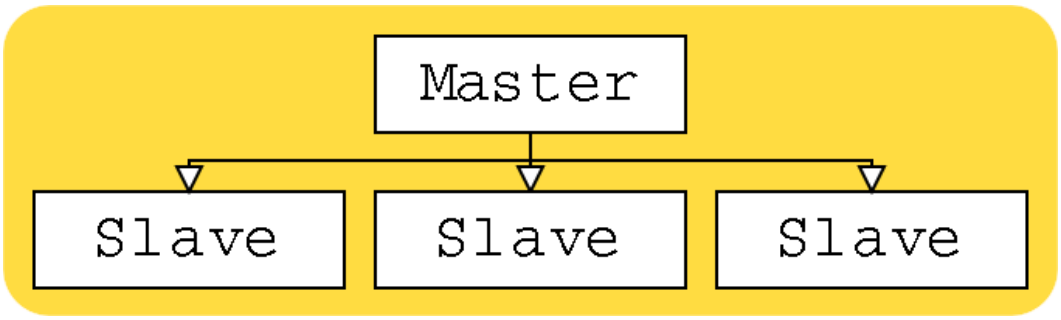
```
PHP_FUNCTION(my_init) {
    char *name = NULL;
    int name_len, age;
    test_le_struct *test_struct;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "sl", &name, &name_len, &age) ==
FAILURE) {
        return;
    }
    test_struct = emalloc(sizeof(test_le_struct));
    test_struct->name = estrndup(name, name_len);
    test_struct->age = age;
    ZEND_REGISTER_RESOURCE(return_value, test_struct,
le_test);
}
```

Followed by other functions that then take a resource as an argument and do something with it:

```
PHP_FUNCTION(my_get)
{
    test_le_struct *test_struct;
    pval *res;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "r", &res) == FAILURE) {
        return;
    }

    ZEND_FETCH_RESOURCE(test_struct, test_le_struct *,
                        &res, -1, "test", le_test);

    if(!test_struct) RETURN_FALSE;

    array_init(return_value);
    add_assoc_string(return_value, "name", test_struct->name,
1);
```

```
    add_assoc_long(return_value, "age", test_struct->age);
}
```

MySQL supports one-way replication. Since most web applications usually have more reads than writes, an architecture which distributes reads across multiple servers can be very beneficial.



In typical MySQL fashion, setting up replication is trivial. On your master server add this to your "my.cnf" file:

```
[mysqld]
log-bin
server-id=1
```

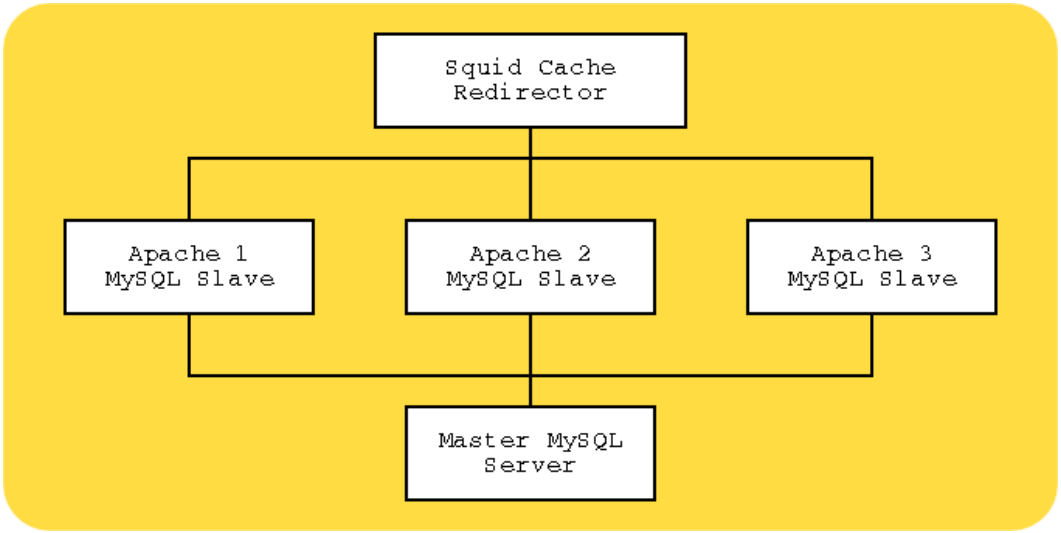
And add a replication user id for slaves to log in as:

```
GRANT REPLICATION SLAVE ON . TO repl@"%" IDENTIFIED BY
'foobar';

[mysqld]
set-variable = max_connections=200
log-bin
master-host=192.168.0.1
master-user=repl
master-password=foobar
master-port=3306
server-id=2
```

Make sure each slave has its own unique server-id. And since these will be read-only slaves, you can start them with these options to speed them up a bit:

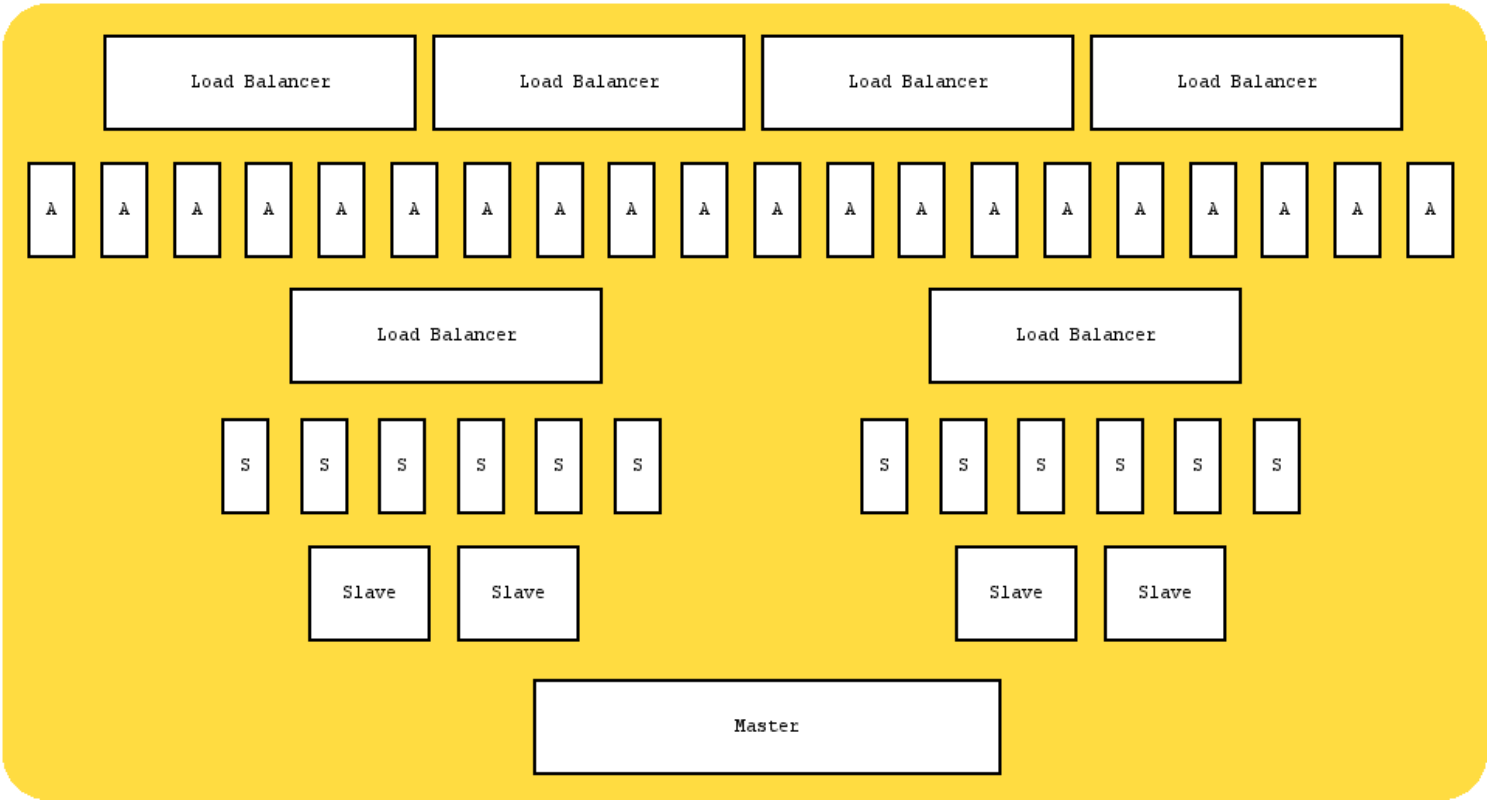
```
--skip-bdb --low-priority-updates
--delay-key-write-for-all-tables
```

Stop your master server. Copy the table files to each of your slave servers. Restart the master, then start all the slaves. And you are done. Combining MySQL replication with a Squid reverse cache and redirector and you might have an architecture like this:



You would then write your application to send all database writes to the master server and all reads to the local slave. It is also possible to set up two-way replication, but you would need to supply your own application-level logic to maintain atomicity of distributed writes. And you lose a lot of the advantages of this architecture if you do this as the writes would have to go to all the slaves anyway.

Bigger?


Load Balancer
Load Balancer
Load Balancer
Load Balancer
A A A A A A A A A A A A A A A A A A A A
Load Balancer
Load Balancer
S S S S S S
S S S S S S
Slave
Slave
Slave
Slave
Master

Resources

Home Page: http://www.php.net

Manual: http://php.net/manual

Books: http://php.net/books.php

Bugs: http://bugs.php.net

PEAR: http://pear.php.net

PECL: http://pecl.php.net

Talks: http://talks.php.net

Xdebug: http://xdebug.org

Index