Do you PHP?

Linux Bangalore 2003

Dec.3, 2003. Bangalore

Rasmus Lerdorf <rasmus@php.net>

http://lerdorf.com/lb1.pdf

Many Things

PHP is Many Things to Many People

- The BASIC of the Web
- Web Template System
- General-purpose Scripting Language
- Advanced Application Framework
- Application Server?

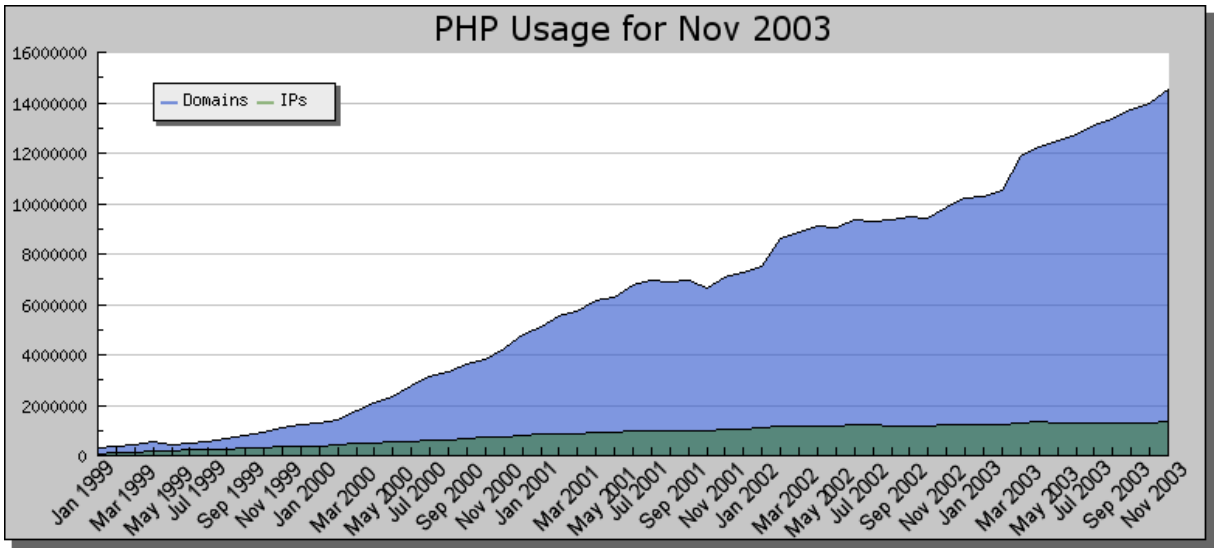
Template System

- 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.

PHP is Big!

November 2003 Netcraft Report

- 44,946,965 Domains queried
- 14,528,748 Domains. 1,328,604 IP addresses
- PHP installed on 32% of all domains



Source: Netcraft

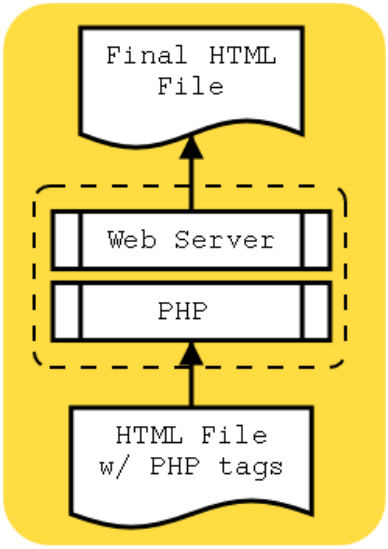
November 2003 Apache Module Report

- 12,220,278 Apache Servers surveyed
- 4,311,776 (52.04%) PHP
- 2,542,854 (30.69%) OpenSSL
- 2,465,141 (29.75%) mod_ssl
- 1,825,732 (22.03%) Frontpage
- 1,607,948 (19.41%) mod_perl
- 406,032 (4.90%) DAV
- 368,240 (4.44%) mod_throttle
- 330,237 (3.99%) mod_jk
- 319,777 (3.86%) mod_log_bytes
- 314,140 (3.79%) mod_bwlimited
- 249,816 (3.01%) mod_fastcgi
- 216,022 (2.61%) AuthMySQL

Source: SecuritySpace.com

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>
```

The 4 available tag styles

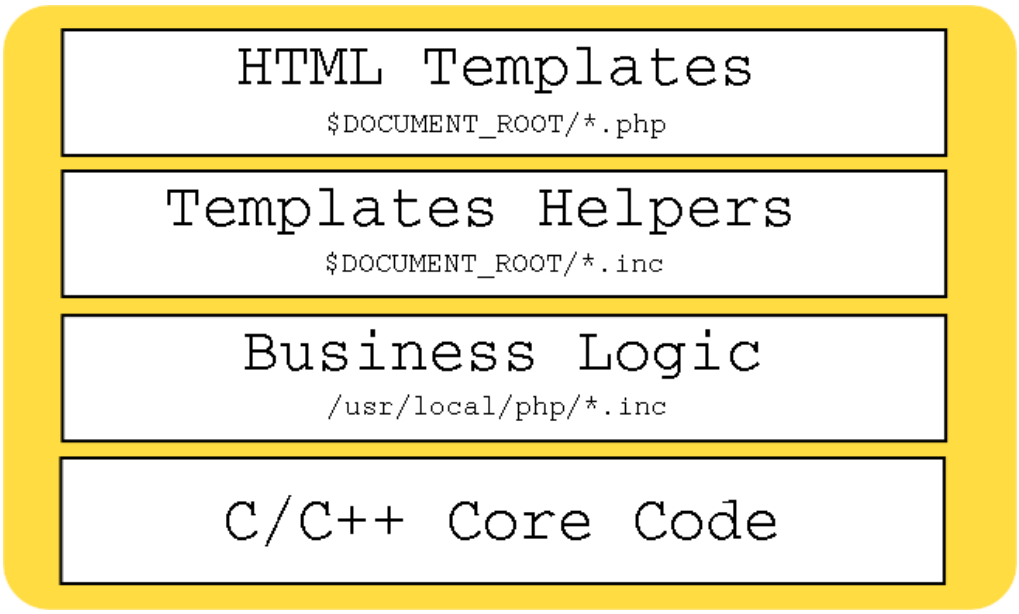
```
<html>
<body>
<? echo 'Short Tags - Most common' ?>
<br />
<?php echo 'Long Tags - Portable' ?>
<br />
<= 'ASP Tags' >
<br />
<script language="php">
 echo 'Really Long Tags - rarely used';
</script>
<br />
</body>
</html>
```

Output:

```
Short Tags - Most common
Long Tags - Portable
ASP Tags
Really Long Tags - rarely used
```

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);
}
```

Index