PHP Coding Guidelines

Version: 1.0

**Table of Contents**

[1 Overview 5](#_Toc222315438)

[1.1 Scope 5](#_Toc222315439)

[1.2 Goals 5](#_Toc222315440)

[2 PHP File Formatting 5](#_Toc222315441)

[2.1 Script TAG 5](#_Toc222315442)

[2.2 Indentation 5](#_Toc222315443)

[2.3 Maximum Line Length 6](#_Toc222315444)

[2.4 Script Termination 6](#_Toc222315446)

[2.5 Line Termination 6](#_Toc222315447)

[3 Naming Conventions 6](#_Toc222315448)

[3.1 Classes 6](#_Toc222315449)

[3.2 Interfaces 6](#_Toc222315450)

[3.3 Filenames 7](#_Toc222315451)

[3.4 Functions and Methods 7](#_Toc222315452)

[3.5 Variables 8](#_Toc222315453)

[3.6 Constants 8](#_Toc222315454)

[4 Coding Style 8](#_Toc222315455)

[4.1 Definition of E\_STRICT-compatible package 9](#_Toc222315456)

[4.2 Implementation of Design Pattern 9](#_Toc222315457)

[5 PHP Code Demarcation 9](#_Toc222315458)

[5.1 Strings 9](#_Toc222315459)

[5.1.1 String Literals 9](#_Toc222315460)

[5.1.2 String Literals Containing Apostrophes 10](#_Toc222315461)

[5.1.3 Variable Substitution 10](#_Toc222315462)

[5.1.4 String Concatenation 10](#_Toc222315463)

[5.2 Arrays 10](#_Toc222315464)

[5.2.1 Numerically Indexed Arrays 10](#_Toc222315465)

[5.2.2 Associative Arrays 11](#_Toc222315466)

[5.3 Classes 11](#_Toc222315467)

[5.3.1 Class Declaration 11](#_Toc222315468)

[5.3.2 Class Member Variables 12](#_Toc222315469)

[5.4 Functions and Methods 12](#_Toc222315470)

[5.4.1 Function and Method Declaration 12](#_Toc222315471)

[5.4.2 Split function definitions onto several lines 13](#_Toc222315472)

[5.4.3 Split function call on several lines 14](#_Toc222315473)

[5.4.4 Function and Method Usage 14](#_Toc222315474)

[5.5 Control Statements 15](#_Toc222315475)

[5.5.1 If / Else / Elseif 15](#_Toc222315476)

[5.5.2 Switch 16](#_Toc222315478)

[5.6 Iteration 16](#_Toc222315479)

[All expression should be separated by single blank space. 17](#_Toc222315480)

[5.7 Script Termination 17](#_Toc222315481)

[5.8 Redirection 17](#_Toc222315482)

[The redirection of script execution can be done by using header(). As per the HTTP/1.1 standard The attribute text should contain “Location: “ followed by the absolute URI. 17](#_Toc222315483)

[Always exit(0) should follow all header redirection statement. 17](#_Toc222315484)

[5.9 Inline Documentation 17](#_Toc222315485)

[5.9.1 Documentation Format 17](#_Toc222315486)

[5.9.2 Files 17](#_Toc222315487)

[\*/ 18](#_Toc222315488)

[5.9.3 Classes 18](#_Toc222315489)

[5.9.4 Documenting Exception 18](#_Toc222315490)

[5.9.5 Functions 19](#_Toc222315491)

[6 References 19](#_Toc222315492)

[7 Best Practices 19](#_Toc222315493)

# Overview

## Scope

This document provides the code guidelines for developers in Advanstar project. The subjects covered are:

* PHP File Formatting
* Naming Conventions
* Coding Style
* Inline Documentation
* PHP development best practice.
* Recommended tools and techniques.

## Goals

Code conventions are important to programmers for number of reasons

* 80% of lifetime cost of a piece of software goes to maintenance
* Hardly any software is maintained for its whole life by its original author
* Code conventions improve the readability of the software, allowing developers to understand the new code more quickly and thoroughly
* Having coding standards helps ensure that the code is of high quality, has fewer bugs and is easily maintained.

# PHP File Formatting

## Script TAG

All php syntax should be enclosed within “<?php” tag

1. SHORT OPEN TAG should be avoided.
2. If the script contains only php syntax, then don’t use php close tag “?>”.

this is helpful when using ‘include’ or ‘require’, so unwanted whitespace will not occur at the end of files, and you will still be able to add headers to the response later. It is also handy if you use output buffering, and would not like to see added unwanted whitespace at the end of the parts generated by the included files.

## Indentation

Use an indent of 4 spaces, with no tabs.

## Maximum Line Length

The target line length is 80 characters, i.e. developers should aim keep code as close to the 80-column boundary as is practical. However, longer lines are acceptable. The maximum length of any line of PHP code is 120 characters.

## Script Termination

End of script should not contain any extra linefeed or white spaces.

## Line Termination

Line termination is the standard way for UNIX text files. Lines must end only with a linefeed (LF). Linefeeds are represented as ordinal 10, or hexadecimal 0x0A.

Do not use carriage returns (CR) like Macintosh computers (0x0D).

Do not use the carriage return/linefeed combination (CRLF) as Windows computers (0x0D, 0x0A).

# Naming Conventions

## Classes

Class names may only contain alphabets. Numbers are not permitted in class names. The only special character allowed in class names is Underscore.

If a class name is comprised of more than one word, the first letter of each new word must be capitalized. Successive capitalized letters are not allowed, e.g. a class "UploadPDF" is not allowed while "UploadPdf" is acceptable.

These are examples of acceptable names for classes:

**UserName**  
  
**ShowImage**  
  
**ViewFaq**

## Interfaces

Interface classes must follow the same conventions as other classes (see above), however must end with the word "Interface", such as in these examples:

**ShowImageInterface**  
  
**ViewFaqInterface**

## Filenames

File names of classes should have the file name like class name. A class named ShowImage should have the file name as ShowImage.php

For all other files, only alphabets are permitted. Spaces are prohibited.

Any file that contains any PHP code must end with the extension ".php". These examples show the acceptable filenames are as below:

**Db.php**  
  
**ChangePassword.php**  
  
**Contact.php**

**ShowImage.class.php**

## Functions / Methods

Function names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in function names but are discouraged.

Function names must always start with a lowercase letter. When a function name consists of more than one word, the first letter of each new word must be capitalized. This is commonly called "camelCase" formatting.

Verbosity is encouraged. Function names should be as verbose as is practical to enhance the understandability of code.

These are examples of acceptable names for functions:

**filterInput()**  
  
**getElementById()**  
  
**username()**

**dateTime()**

For object-oriented programming, accessors for objects should always be prefixed with either "get" or "set". When using design patterns, such as the singleton or factory patterns, the name of the method should contain the pattern name where practical to make the pattern more readily recognizable.

For methods on objects that are declared with the "private" or "protected" construct, the first character of the variable name must be a single underscore. This is the only acceptable usage of an underscore in a method name. Methods declared "public" may never start with an underscore.

Functions in the global scope ("floating functions") are permitted but discouraged. It is recommended that these functions should be wrapped in a static class.

## Variables

Variable names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in variable names but are discouraged.

For class member variables that are declared with the "private" or "protected" construct, the first character of the variable name must be a single underscore. This is the only acceptable usage of an underscore in a variable name. Member variables declared "public" may never start with an underscore.

Like function names (see section 3.4, above) variable names must always start with a lowercase letter and follow the "camelCaps" capitalization convention.

Verbosity is encouraged. Variables should always be as verbose as practical. The variable names such as "$i" and "$n" are discouraged for anything other than the smallest loop contexts. If a loop contains more than 20 lines of code, the variables for the indices need to have more descriptive names.

Global constants should be all caps with '\_' separators.

It's tradition for global constants to name this way. We must be careful to not conflict with other predefined global. Always use project related prefix in CONSTANT naming,

**define("USR\_DATABASE\_TYPE", "mysql");**

here ‘USR\_’ should be replace with project specific term.

## Constants

Constants may contain both alphanumeric characters and the underscore. Numbers are permitted in constant names.

Constants must always have all letters capitalized.

To enhance readability, words in constant names must be separated by underscore characters. For example, DB\_ERROR is permitted but DBERROR is not.

Constants must be defined as class members by using the "const" construct. Defining constants in the global scope with "define" is permitted but discouraged.

# Coding Style

## Definition of E\_STRICT-compatible package

The package is considered E\_STRICT-compatible if

* It can be used under PHP 5.1.4+
* Its files do not emit error messages when used with error reporting level set to E\_ALL | E\_STRICT under PHP 5.1.4+
* It follows PEAR coding standards that apply to PHP5-only packages
* It has only E\_STRICT-compatible required dependencies

## Implementation of Design Pattern

Design Patterns are simply defined solutions to common problems.  These are proven solutions that are implemented over and over again through different projects.  This re-use of the solution itself becomes a pattern.

Design and Design Patterns are beneficial in regards of:

* Maintenance
* Documentation
* Readability
* Easy when developing in large development teams
* Developing Code to be used by other then yourself

Please check the PHP\_TECHNOLOGIES\_DEVELOPMENT.doc for more details

# PHP Code Demarcation

PHP code must always be delimited by the full-form, standard PHP tags, If it is standalone php script then done use end/close tag. But never start with short form tag.

**<?php**  
  
**?>**

Short tags are never allowed. For files containing only PHP code, the closing tag must always be omitted (See [Section 2.1, “General”](#_General)).

## Strings

### String Literals

When a string is literal (contains no variable substitutions), the apostrophe or "single quote" must always used to demarcate the string:

**$a = 'Example String';**

### 

### String Literals Containing Apostrophes

When a literal string itself contains apostrophes, it is permitted to demarcate the string with quotation marks or "double quotes". This is especially encouraged for SQL statements:

**$sql = "SELECT `id`, `name` from `people` WHERE `name`='Fred' OR `name`='Susan'";**

The above syntax is preferred over escaping apostrophes.

### Variable Substitution

Variable substitution is permitted using either of these two forms:

**$greeting = "Hello $name, welcome back!";**  
  
**$greeting = "Hello {$name}, welcome back!";**

For consistency, this form is not permitted:

**$greeting = "Hello ${name}, welcome back!";**

### 

### String Concatenation

Strings may be concatenated using the "." operator. A space must always be added before and after the "." operator to improve readability:

**$company = 'Advanstar' . ' ' . 'Inc';**

When concatenating strings with the "." operator, it is permitted to break the statement into multiple lines to improve readability. In these cases, each successive line should be padded with whitespace such that the "."; operator is aligned under the "=" operator:

**$sql = "SELECT `id`, `name` FROM `people` "**  
**. "WHERE `name` = 'Susan' "**  
**. "ORDER BY `name` ASC ";**

## Arrays

### Numerically Indexed Arrays

Negative numbers are not permitted as indices.

An indexed array may be started with any non-negative number, however this is discouraged and it is recommended that all arrays have a base index of 0.

When declaring indexed arrays with the array construct, a trailing space must be added after each comma delimiter to improve readability:

**$sampleArray = array(1, 2, 3, 'Hcl', 'Technologies');**

It is also permitted to declare multi line indexed arrays using the "array" construct. In this case, each successive line must be padded with spaces such that beginning of each line aligns as shown below:

**$sampleArray = array(1, 2, 3, 'Hcl', 'Technologies',**  
**$a, $b, $c,**  
**56.44, $d, 500);**

### Associative Arrays

When declaring associative arrays with the array construct, it is encouraged to break the statement into multiple lines. In this case, each successive line must be padded with whitespace such that both the keys and the values are aligned:

**$sampleArray = array('firstKey'  => 'firstValue',**  
**'secondKey' => 'secondValue');**

Don’t use array\_merge for associative arrays instead ‘+’ operator can be used to union of two arrays.

## Classes

### Class Declaration

Classes must be named by following the naming conventions.

The brace is always written on the line underneath the class name ("one true brace" form).

Every class must have a documentation block that conforms to the PHPDocumentor standard.

Any code within a class must be indented four spaces.

Only one class is permitted per PHP file.

Placing additional code in a class file is permitted but discouraged. In these files, two blank lines must separate the class from any additional PHP code in the file.

This is an example of an acceptable class declaration:

**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**class SampleClass**  
**{**  
**// entire content of class**  
**// must be indented four spaces**  
**}**

### Class Member Variables

Member variables must be named by following the variable naming conventions.

Any variables declared in a class must be listed at the top of the class, prior to declaring any methods.

The var construct is not permitted. Member variables always declare their visibility by using one of the private, protected, or public constructs. Accessing member variables directly by making them public is permitted but discouraged in favor of accessor methods (set/get).

## Functions and Methods

### Function and Method Declaration

Functions must be named by following the naming conventions.

Methods inside classes must always declare their visibility by using one of the private, protected, or public constructs.

Like classes, the brace is always written on the line underneath the function name ("one true brace" form). There is no space between the function name and the opening parenthesis for the arguments.

Functions in the global scope are strongly discouraged.

This is an example of an acceptable function declaration in a class:

**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**class Foo**  
**{**  
**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**public function bar()**  
**{**  
**// entire content of function**  
**// must be indented four spaces**  
**}**  
**}**

NOTE: Pass-by-reference is permitted in the function declaration only:

**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**class Foo**  
**{**  
**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**public function bar(&$baz)**  
**{}**  
**}**

Call-time pass-by-reference is prohibited.

The return value must not be enclosed in parentheses. This can hinder readability and can also break code if a method is later changed to return by reference.

**/\*\***  
**\* Documentation Block Here**  
**\*/**  
**class Foo**  
**{**  
**/\*\***  
**\* WRONG**  
**\*/**  
**public function bar()**  
**{**  
**return($this->bar);**  
**}**  
  
**/\*\***  
**\* RIGHT**  
**\*/**  
**public function bar()**  
**{**  
**return $this->bar;**  
**}**  
**}**

### Split function definitions onto several lines

Functions with many parameters need to be split onto several lines to keep the 80 chars/line limit. The first parameters may be put onto the same line as the function name if there is enough space. Subsequent parameters on following lines are to be indented 4 spaces. The closing parenthesis and the opening brace are to be put onto the next line, on the same indentation level as the "function" keyword.

**<?php**

**function someFunction ($firstParameter = 'something', $secondParameter = 'booooo',**

**$third = null, $fourthParameter = false, $fifthParameter = 123.12,**

**$sixthParam = true**

**) {**

**//....**

**?>**

### Split function call on several lines

The CS require lines to have a maximum length of 80 chars. Calling functions or methods with many parameters while adhering to CS is impossible in that cases. It should be allowed to split parameters in function calls onto several lines.

Example:

**<?php**

**$this->someObject->subObject->callThisFunctionWithALongName(**

**$parameterOne, $parameterTwo,**

**$aVeryLongParameterThree**

**);**

**?>**

Several parameters per line should be allowed. Parameters need to be indented 4 spaces compared to the level of the function call. The opening parenthesis is to be put at the end of the function call line, the closing parenthesis gets its own line at the end of the parameters. This shows a visual end to the parameter indentations and follows the opening/closing brace rules for functions and conditionals.

### Function and Method Usage

Function arguments are separated by a single trailing space after the comma delimiter. This is an example of an acceptable function call for a function that takes three arguments:

**threeArguments(1, 2, 3);**

Call-time pass-by-reference is prohibited. See the function declarations section for the proper way to pass function arguments by-reference.

For functions whose arguments permit arrays, the function call may include the "array" construct and can be split into multiple lines to improve readability. In these cases, the standards for writing arrays still apply:

**threeArguments(array(1, 2, 3), 2, 3);**  
  
**threeArguments(array(1, 2, 3, 'Hcl', 'Technologies',**  
**$a, $b, $c,**  
**56.44, $d, 500), 2, 3);**

#### Function Argument

PHP can accept value by reference, in that case it instead of using data pointer, it use named table and same variable location can be referred by more than one name. This will help to optimize memory usage in our code. But any changes in the parameter variable it also change value in main variable.

We should use this feature, when we have to call a function by passing array or object as parameter.

## Control Statements

### If / Else / Elseif

Control statements based on the if and elseif constructs must have a single space before the opening parenthesis of the conditional and a single space after the closing parenthesis.

Within the conditional statements between the parentheses, operators must be separated by spaces for readability. Inner parentheses are encouraged to improve logical grouping of larger conditionals.

The opening brace is written on the same line as the conditional statement. The closing brace is always written on its own line. Any content within the braces must be indented four spaces.

**if ($a != 2) {**  
**$a = 2;**  
**}**

For "if" statements that include "elseif" or "else", the formatting conventions are as shown in the following examples:

**if ($a != 2) {**  
**$a = 2;**  
**} else {**  
**$a = 7;**  
**}**  
  
**if ($a != 2) {**  
**$a = 2;**  
**} elseif ($a == 3) {**  
**$a = 4;**  
**} else {**  
**$a = 7;**  
**}**

PHP allows for these statements to be written without braces in some circumstances. The coding standard makes no differentiation and all "if", "elseif" or "else" statements must use braces.

Use of the "elseif" construct is permitted but highly discouraged in favor of the "else if" combination.

### Switch

Control statements written with the "switch" construct must have a single space before the opening parenthesis of the conditional statement, and also a single space after the closing parenthesis.

All content within the "switch" statement must be indented four spaces. Content under each "case" statement must be indented an additional four spaces. Although switch statement is using loose comparison still its good practice to use default case inside switch, to handle unknown value.

**switch ($numPeople) {**  
**case 1:**  
**break;**  
  
**case 2:**  
**break;**  
  
**default:**  
**break;**  
**}**

The construct default may never be omitted from a switch statement.

NOTE: It is sometimes useful to write a case statement which falls through to the next case by not including a break or return in that case. To distinguish these cases from bugs, any case statement where break or return are omitted must contain the comment "// break intentionally omitted".

## Iteration

PHP Iteration can be used in many ways, for, foreach, while, do-while, there is no any limitation of using any of these iteration, but always comment if anywhere break statement used.

Do not use variable assignment in iteration expression; rather make it outside (before) of the expression,

**$size = count($data);**

**$print = 0;**

**for ($count =0; $count < $size; $count++) {**

**if ($data[$count][‘price’] > 100) {**

**echo ‘Item: ’ . $data[$count][‘name’];**

**$print++;**

**if ($print == 10 ) {**

**/\* First 10 Item having price more than 100 unit will be displayed.\*/**

**break;**

**}**

**}**

**}**

## All expression should be separated by single blank space.

## Script Termination

Script can be terminated by calling exit() function The exit function should use the exit status range between 0 to 254. But dont leave it empty. Use 0 for successful program termination.

## Redirection

## The redirection of script execution can be done by using header(). As per the HTTP/1.1 standard The attribute text should contain “Location: “ followed by the absolute URI.

## Always exit(0) should follow all header redirection statement.

**header (“Location: signout.php”);**

**exit(0);**

## Inline Documentation

### Documentation Format

All documentation blocks ("docblocks") must be compatible with the phpDocumentor format. Describing the phpDocumentor format is beyond the scope of this document. For more information, visit: <http://phpdoc.org/>

All source code files must contain a "file-level" docblock at the top of each file and a "class-level" docblock immediately above each class. Below are examples of such docblocks.

### Files

Every file that contains PHP code must have a header block at the top of the file that contains these phpDocumentor tags at a minimum:

**/\*\***

**\* File short description**

**\* File detailed description (if any)**

**\***

**\* Mention the Dependencies and requirements in this file**

**\***

**\* PHP version 5.2.3**

**\***

**\* LICENSE:-**

**\***

**\*** @category **Entity**

**\*** @package **Classes**

**\*** @author **HCL**

**\*** @copyright **Copyright (c) 2010-present Advanstar Inc.**

**\* All rights reserved.**

**\*** @license **http://www.dvm360.com DVM360 License**

**\*** @version **CVS: $Id: Venues.class.php,v 1.15 2009/02/13 02:35:44 $**

**\*** @link[**http://www.dvm360.com**](http://www.dvm360.com)

**\* @see parserDocBlock, parserInclude, parserPage, parserClass**

**\* @see parserDefine, parserFunction, parserMethod, parserVar**

**\*** @since **File available since Release 0.1**

**\*/**

### \*/Classes

Every class must have a docblock that contains these phpDocumentor tags at a minimum:

**/\*\***  
**\* Short description for class**  
**\***  
**\* Long description for class (if any)...**  
**\***  
**\* @copyright  2010 Advanstar**  
**\* @license    License information**  
**\* @version    Release: @package\_version@**  
**\* @since      Class available since Release 1.2.0**

**\*/**

### Documenting Exception

Because PHP, unlike Java, does not require you to explicitly state which Exceptions a method throws in the method signature, it is critical that Exceptions be thoroughly documented in your method headers.

Exceptions should be documented using the **@throws** phpdoc keyword:

**<?php**

**/\*\***

**\* This method searches for aliens.**

**\***

**\* @return array Array of Aliens objects.**

**\* @throws AntennaBrokenException If the impedence readings indicate**

**\* that the antenna is broken.**

**\***

**\* @throws AntennaInUseException If another process is using the**

**\* antenna already.**

**\*/**

**public function findAliens($color = 'green');**

**?>**

### Functions

Every function, including object methods, must have a docblock that contains at a minimum:

* A description of the function
* All of the arguments
* All of the possible return values

It is not necessary to use the "@access" tag because the access level is already known from the "public", "private", or "protected" construct used to declare the function.

If a function/method may throw an exception, use @throws:

**/\*\***

**\* Connect to database**

**\***

**\* @param string $host Host to connect to**

**\* @param string $user Userid to login**

**\* @param string $password Associated password**

**\* @param string $database database**

**\* @param bool $forceConn force new connection**

**\***

**\* @return true or false**

**\***

**\* @throws exceptionclass [description]**

**\***

**\* @todo cleanup the code,**

**\*/**

# References

* Zend Framework Coding standards
* PEAR Coding standard guideline

# Best Practices

1. **Don’t store any user level sensitive data (If it is not required and requirement not verified and confirmed), from clients input. The sensitive data may be in any form like credit card number.**
2. **Don’t log any system related information like database or ftp credential.**
3. While framing sql queries, never enclose integers within single quotes. These will slowdown the queries.

Keep the all used SQL in separate file with heredoc format and assign to variable

$sql[‘select\_user’] =<<<SQL

SELECT \* FROM `tbl\_user` WHERE `uid` = %d

SQL;

Include the SQL file and access the SQL by variable, parse it to substitute the %d with value.

1. Use mysql\_escape\_string for handling SQL injection attacks. (This is very important).
2. Make sure the same set of files is not included more than once.
3. Doesn’t use any OS dependent functions in common files keep it separate.
4. Always keep HTTP server (Apache or IIS) configuration or constants in single commonly accessed file
5. Eliminate unnecessary cookies
6. Keep cookie sizes as low as possible to minimize the impact on the user response time.
7. Be mindful of setting cookies at the appropriate domain level so other sub-domains are not affected
8. Comment code wherever required for better understanding.
9. Avoid Double Quotes around Strings that Contain HTML
10. Echo is faster than print. Use echo over print.
11. Use echo’s multiple parameters instead of string concatenation.
12. Set the max value for your for-loops before and not in the loop.
13. Unset your variables to free memory, especially large arrays.
14. require\_once () is expensive
15. Use full paths in includes and requires, less time spent on resolving the OS paths.
16. If you need to find out the time when the script started executing, $\_SERVER[’REQUEST\_TIME’] is preferred to time()
17. See if you can use strncasecmp, strpbrk and stripos instead of regex
18. str\_replace is faster than preg\_replace, but strstr is faster than str\_replace
19. Error suppression with @ is very slow.
20. Turn on apache’s mod\_deflate
21. Close your database connections when you’re done with them
22. $row[’id’] is 7 times faster than $row[id]
23. Incrementing a global variable is 2 times slower than a local var.
24. Incrementing an object property (eg. $this->prop++) is 3 times slower than a local variable.
25. Incrementing an undefined local variable is 9-10 times slower than a pre-initialized done.
26. Methods in derived classes run faster than ones defined in the base class.
27. Surrounding your string by ‘instead of’ will make things interpret a little faster since php looks for variables inside “…” but not inside ‘…’. Of course you can only do this when you don’t need to have variables in the string.
28. A PHP script will be served at least 2-10 times slower than a static HTML page by Apache. Try to use more static HTML pages and fewer scripts.
29. Your PHP scripts are recompiled every time unless the scripts are cached. Install a PHP caching product to typically increase performance by 25-100% by removing compile times.
30. Cache as much as possible. Use memcached - memcached is a high-performance memory object caching system intended to speed up dynamic web applications by alleviating database load. OP code caches are useful so that your script does not have to be compiled on every request .
31. You can improve the speed of your code by using an isset() trick.
32. if (!isset($foo{5})) { echo "Foo is too short"; }
33. Calling isset() happens to be faster then strlen() because unlike strlen(), isset() is a language construct and not a function meaning that it’s execution does not require function lookups and lowercase. This means you have virtually no overhead on top of the actual code that determines the string’s length.
34. Do use POST over GET for all values that will wind up in the database for TCP/IP packet performance reasons.
35. Do use ctype\_alnum,ctype\_alpha and ctype\_digit over regular expression to test form value types for performance reasons.
36. Do use require/include over require\_once/include\_once to ensure proper opcode caching.
37. Do set Apache allowoverride to “none” to improve Apache performance in accessing files/directories.
38. Do not rename the file to new name for iterating same set of code. Doing so will create redundancy in code.
39. Because PHP is not 100% object oriented and that the fact HTML page are static pages you may want to consider using more of a hybrid approach where you use design patterns where it makes sense.