IonExpress 5.0

PHP Coding Standards

Version: 5.0
Date: Jan 11, 2011

Document Information

Author	Kasturi K Murthy, Jayanth P
Reviewer	Shaily Goel, Varun K
Approver	**Manohar Joshi, SEPG**

Change History

Version #	Release Date	Sections Changed	Author	Change Description
0.1	16-Aug-10	NA	Kasturi / Jayanth	Initial Draft Version
0.2	20 Dec 2010	2.3, 2.4, 2,2, 9.4, 12,1, 12.3	Kasturi	Shaily's Review comments implemented
0.3	21 Dec 2010	11.1	Kasturi	Varun's review comments added
1.0	21 Dec 2010	NA	Kasturi	Approved by Manohar
5.0	11 Jan 2011	Formatted the document	Saswata	Baselined to IonExpress 5.0

TABLE OF CONTENTS

1 Purpose

This document provides the coding standards and guidelines for php developers and teams. Code conventions can help in reducing the brittleness of programs.

2 Naming Conventions

9.1 Global Variables and Functions

Global names should start with a single underscore followed by the package name and another underscore. Global functions should use camel case. In addition, they should have the package name as a prefix, to avoid name collisions between packages. The initial letter of the name (after the prefix) is lowercase, and each letter that starts a new "word" is capitalized. An example: XML_RPC_serializeData()

9.1 Classes

Classes should be given descriptive names. Class names should always begin with an uppercase letter.
Examples of good class names are: HTML_Upload_Error, Log etc
Example

```
<?
class human{
var $legs=2;
var $arms=2;
}
//instantiate the class
$jude = new human();
echo "Jude has " .$jude->legs." legs";
?>
```

Class variables and Methods should be named using camel case. Private class members are preceded by a single underscore.

Example:

```
Class Customer{
private  _first_name;
private _last_name;
        function getCustomerdData(){
                //some code goes here
        }
}
```

9.1 Constants

Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercased name of the package they are used in. Some examples:

```
define("REG_MAX_SIZE", 100);

echo REG_MAX_SIZE;
echo constant("REG_MAX_SIZE");
```

9.1 Variables

Variable names should be all lowercase, with words separated by underscores. Variable names should be short yet meaningful. One-character variable names should be avoided except for temporary variables and loop indices. –

Examples:

```
$current_user=””;
$setEmail=””;
$count_rows=0;
```

Common names for temporary variables are i, j, k, m, and n for integers; c, d, e for strings.

- use all lower case letters
- use '_' as the word separator
- do not use 'l' (lowercase 'L') as a temporary variable
- do not use '-' as the word separator
- Also define Method Argument names : Method arguments should be separated by spaces, both when the function is defined and when it is called. However, there should not be any spaces between the arguments and the opening/closing parentheses

E.g. set_password($email, $password)

3 Formatting

PHP Tags are used for delimit PHP from HTML in a file. There are several ways to do this. <?php ... ?>, <? ... ?>, <script language="php"> ... </script>, <% ... %>, and <?=$name?>. Some of these may be turned off in your PHP settings. Always use <?php ... ?>.

9.1 Indenting and Line Length

Use an indent of 4 spaces, with no tabs. This helps to avoid problems with diffs, patches, SVN history and annotations. – usually in every editor we use tabs rather than spaces this . I prefer tab to be used because it is easy to align everything with tabs. Most editors can set how many "spaces" are displayed for each tab, so you can set your editor up to display tabs however you like (2 spaces, 4 spaces, whatever), and the other programmers on the team may do the same.

9.1 One Statement per line

There should be only one statement per line.

Parenthesis(..)

As a general rule, parentheses should always be used where the possibility of ambiguity exists. Additionally, there should not be any added spacer between a parenthesis and its contents. Control statements, such as if, for, and while, should have one space after the keyword prior to the opening parenthesis. Methods should follow the rules laid out already, i.e. no space between the method name and the opening parenthesis, and no space between the parentheses and the arguments, but one space between each argument; generally speaking, this is the only time an opening parenthesis will not be preceded by a space. Example

```
if ((condition1 || condition2) || (value1 < value2))
{
		...
}
while (condition)
{
		...
}
strcmp($s, $s1);
return 1;
```

9.1 Operators and tokens

There should not be space between a unary operator and the thing on which is operates.

There should always be one space on either side of a token or binary operator. The only exceptions are commas and semicolons, which should have one space after, but none before (just like in English).

5.4.1 If ..then …else

Always use braces.

3.1.1 Example

```
if (condition)
{
	// comment
```

```
        ...
}
else if (condition)
{
        // comment
        ...
}
else
{
        // comment
        ...
}
```

5.4.2 Switch

- Falling through a case statement into the next case statement shall be permitted as long as a comment is included.
- The *default* case should always be present and trigger an error if it should not be reached, yet is reached.
- Although case blocks do not use braces, indent them as if they did

3.2.1 Example

```
switch (...)
{
        case 1:
                ...
                // FALL THROUGH
        case 2:
                $v = get_week_number();
                ...
                break;
        default:
}
```

4 Comments

Consider your comments a story describing the system. Class comments are one part of the story, method signature comments are another part of the story, method arguments another part, and method implementation yet another part. All these parts should weave together and inform someone else at another point of time just exactly what you did and why. Comments should document decisions. At every point where you have a choice of what to do, place a comment describing which choice you made and why.

9.1 Comment Layout

Each part of the project has a specific comment layout.

C style comments (/* */) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.

5 Function Calls

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
<?php
  $var = foo($bar, $baz, $quux);
?>
```

6 Class and Function Definitions

Class declarations have their opening brace on a new line:

```
<?php
class Foo_Bar
{

   //... code goes here

}
?>
```

Function declarations are as follows:

```
<?php
function fooFunction($arg1, $arg2 = '')
{
   if (condition) {
      statement;
   }
   return $val;
}
?>
```

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate

Split function definitions onto several lines

Functions with many parameters may need to be split onto several lines to keep the 80 characters/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 someFunctionWithAVeryLongName($firstParameter = 'something', $secondPar
ameter = 'booooo',
            $third = null, $fourthParameter = false, $fifthParameter = 123.12,
          $sixthParam = true
) {
    //....
?>
```

7 Arrays

Assignments in arrays may be aligned. When splitting array definitions onto several lines, the last value may also have a trailing comma. This is valid PHP syntax and helps to keep code diffs minimal:

```
<?php

$some_array = array(
    'foo'  => 'bar',
    'spam' => 'ham',
);
?>
```

8 Including Code

Anywhere you are unconditionally including a class file, use **require_once**. Anywhere you are conditionally including a class file (for example, factory methods), use **include_once**. Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with **require_once** will not be included again by **include_once**.

9 Control Structures

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

9.1 Control statements

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

```
<?php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    default action;
}
?>
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.
You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

9.1 For Switch Statements

```
<?php
switch (condition) {
    case 1:
          action1;
          break;

     case 2:
           action2;
           break;

     default:
         default action;
         break;
}
?>
```

9.1 Split long if statements onto several lines

Long if statements may be split into several lines, when the character/line limit will be exceeded. The conditions have to be positioned onto the following line, and indented 4 characters. The logical operators (&&, ||, etc.) should be at the beginning of the line to make it easier to comment (and exclude) the condition. The closing parenthesis and opening brace get their own line at the end of the conditions.

Keeping the operators at the beginning of the line has two advantages: It is trivial to comment out a particular line during development while keeping syntactically correct code (except of course the first line). Further is the logic kept at the front where it's not forgotten. Scanning such conditions is very easy since they are aligned below each other.

```
<?php

    if (($condition1
        || $condition2)
          && $condition3
         && $condition4
     ) {
         //code here
        }
?>
```

The first condition may be aligned to the others.

```
<?php

if (   $condition1
    || $condition2
    || $condition3
) {
    //code here
}
?>
```

The best case is of course when the line does not need to be split. When the if clause is really long enough to be split, it might be better to simplify it. In such cases, you could express conditions as variables and compare them in the if() condition. This has the benefit of "naming" and splitting the condition sets into smaller, better understandable chunks:

```
<?php

$is_foo = ($condition1 || $condition2);
$is_bar = ($condition3 && $condtion4);
if ($is_foo && $is_bar) {
    // ....
}
?>
```

9.1 Ternary Operators

The same rule as for if clauses also applies for the ternary operator: It may be split onto several lines, keeping the question mark and the colon at the front.

Example:

```
<?php
   $agestr = ($age < 16) ? 'child' : 'adult';
?>
```

First there is a condition ($age < 16), then there is a question mark, and then a true result, a colon, and a false result. If $age is less than 16, $agestr will be set to 'child', otherwise it will be set to 'adult'. That one-liner ternary statement can be expressed in a normal if statements like this:

```
<?php
   if ($age < 16) {
      $agestr = 'child';
   } else {
      $agestr = 'adult';
   }
?>
```

10 General guidelines

This part of the Coding Standards describes how errors are handled in PHP programming.

9.1 Quoting Strings

Strings in PHP can either be quoted with single quotes ('') or double quotes (""). The difference between the two is that the parser will use variable-interpolation in double-quoted strings, but not with single-quoted strings. So if your string contains no variables, use single quotes and save the parser the trouble of attempting to interpolate the string for variables, like so:

```
$str = "Avoid this - it just makes more work for the parser.";
```

$str = 'This is much better.'

Likewise, if you are passing a variable to a function, there is no need to use double quotes:

foo("$bar"); // No need to use double quotes

foo($bar); // Much better

Finally, when using associative arrays, you should include the key within single quotes to prevent any ambiguities, especially with constants

$foo = bar[example]; // Wrong: what happens if 'example' is defined as a constant elsewhere?

$foo = bar['example']; // Correct: no ambiguity as to the name of the key

However, if you are accessing an array with a key that is stored in a variable, you can simply use:

$foo = bar[$example];

11 Error guidelines

9.1 Definition of an Error

An error is defined as an unexpected, invalid program state from which it is impossible to recover. For the sake of definition, recovery scope is defined as the method scope. Incomplete recovery is considered a recovery

Example:

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}

// Continue execution
echo 'Hello World';
?>
```

What you need to remember as a developer is that the person who uses your script may not have exactly the same php.ini configuration as you so you aim for the lowest common denominator, i.e. all error reporting enabled. If your code works with E_ALL set, then it will also work with any other error reporting configuration, including when all error reporting is turned off (e.g. on sites where PHP errors are logged to a file instead).

Of course, on a production site you might want to turn off all errors, or at least redirect them to a file, to avoid admitting to users that your scripts are broken in some way. That's perfectly fine and in many cases the recommended action to take. So long as your scripts work with all error reporting turned on, it doesn't matter where they are deployed.

12 Best Practices

9.1 Readability of code blocks

Related lines of code should be grouped into blocks, separated from each other to keep readability as high as possible.

Example:

```
for($i=0;$i<count($arrcount);$i++){
        //manipulation of data result
        $data1=$arrcount[$i];
        $data2=$data1*2*100;


          //Display data result
          echo "<table>";
          echo "<tr>";
          echo "<td> Data Result </td>";
          echo "<td> ".$data2."</td>";
          echo "</tr>";
          echo "</table>";
}
```

9.1 Return early

To keep readability in functions and methods, it is wise to return early if simple conditions apply that can be checked at the beginning of a method:

```
<?php

function foo($bar, $baz)
{
    if ($foo) {
        //assume
        //that
        //here
        //is
        //the
        //whole
        //logic
        //of
        //this
        //method
        return $calculated_value;
    } else {
        return null;
    }
}
?>
```

9.1 Others

- Do not hardcode magic numbers rather define them.
- It's always better to define and initialize variable.
- Commenting a big piece of code which is used for testing use if(0) rather than comments.
- Avoid hardcoding, instead use DEFINE if needed.

```
<?php

function foo($bar, $baz)
{
    if (!$foo) {
        return null;
    }
```

```
   //assume
   //that
   //here
   //is
   //the
   //whole
   //logic
   //of
   //this
   //method
   return $calculated_value;
}
?>
```