PHP Source Code Formatting Guidelines

Developers Handbook

Contents

Introduction

Properly written source code is one of the major factors from which the quality of software stems. In its turn, the crucial factors of the quality source code are readability and comprehensibility. To make your source code readable to other developers, a set or formal rules is required.

The source code formatting rules must be adhered to throughout the whole project files. If there are multiple projects that may possibly have interconnections, the rules must apply to these projects with no exception.

Chapter 1.
Source Code Formatting

Text Structure Rules

Line Length

Avoid typing lines whose length exceeds 120 characters. If a line spans beyond that limit, use the line wrapping rules described below.

Line Wrapping Rules

If a line length exceeds 120 characters, the following wrapping rules apply:

- wrap lines after the comma or before the operator;
- the wrapped line must be indented by one tab;
- use UNIX line ends.

Example 1

The code

```
$arAuthResult = $USER->ChangePassword($USER_LOGIN,
$USER_CHECKWORD, $USER_PASSWORD, $USER_CONFIRM_PASSWORD,
$USER_LID);
```

needs to be wrapped as follows:

```
$arAuthResult = $USER->ChangePassword($USER_LOGIN,
$USER_CHECKWORD,
  $USER_PASSWORD, $USER_CONFIRM_PASSWORD, $USER_LID);
```

Example 2

The code

```
if(COption::GetOptionString("main", "new_user_registration",
"N")=="Y" && $_SERVER['REQUEST_METHOD']=='POST' &&

$TYPE=="REGISTRATION" && (!defined("ADMIN_SECTION") ||
ADMIN_SECTION!==true))
```

needs to be wrapped as follows

```
if (COption::GetOptionString("main", "new_user_registration", 
"N") == "Y"
 && $_SERVER['REQUEST_METHOD'] == 'POST' && $TYPE == 
"REGISTRATION"
 && (!defined("ADMIN_SECTION") || ADMIN_SECTION !== true))
{
    ...
}
```

Spaces And Tabs

Use tabs for indentation. Using spaces for indentation is forbidden for the following reasons:

- with tabs, any developer can configure his or her text editor to show the desired tab length;
- using tabs makes file size smaller;
- if both tabs and spaces are used for indentation, the originally intended text formatting is likely to be damaged if a different tab size is used in a text editor.

Scopes And Code Blocks

The code block contents must be indented by one tab. The code block contents must not be on the same line as the controlling statement.

Example

The format of this statement is invalid:

```
if ($a == 0) {$a = 10;
$b = 1;}
```

It should be reformatted like this:

```
if ($a == 0)
{
    $a = 10;
    $b = 1;
}
```

Expressions And Statements

Expressions

One line must contain only one expression.

Example

This expression is formatted incorrectly:

```
$a = $b; $b = $c; $c = $a;
```

Rewrite it like this:

```
$a = $b;
$b = $c;
$c = $a;
```

The statements if, else, while etc.

Use one of the following formatting rules depending on the statement length.

- if a controlled code block contains only one statement, use the following form:

```
if (expression)
    statement 1;
else
    statement 2;
```

- if at least one controlled code block contains multiple statements, use braces:

```
if (expression)
{
    statement 1;
}
else
{
    statement 2;
    statement 3;
}
```

The rule “Scopes And Code Blocks” must be obeyed when writing multiple statements: they must be indented by one tab off the controlling statement. The braces must exist on new lines on the same level as the controlling statement.

Example

This code

```
if ($a == 0) $a = 10;
else{
$a = 5;
$b = 10;}
```

must be reformatted like this:

```
if ($a == 0)
{
    $a = 10;
}
else
{
    $a = 5;
    $b = 10;
}
```

Compound Expressions

Compound expressions must be split in multiple lines according to rules described in “The statements if, else, while etc.”.

For example, consider the following code.

```
if(COption::GetOptionString("main", "new_user_registration",
"N")=="Y" && $_SERVER['REQUEST_METHOD']=='POST' &&
$TYPE=="REGISTRATION" && (!defined("ADMIN_SECTION") ||
ADMIN_SECTION!==true))
```

Make it readable by sticking to the formatting rules:

```
if (COption::GetOptionString("main", "new_user_registration",
"N") == "Y"
 && $_SERVER['REQUEST_METHOD'] == 'POST' && $TYPE ==
"REGISTRATION"
 && (!defined("ADMIN_SECTION") || ADMIN_SECTION !== true))
{
}
```

It is recommended that you split an extremely complex expression into several simple lines of code.

For example, the code

```
if((!(defined("STATISTIC_ONLY") && STATISTIC_ONLY &&
substr($APPLICATION->GetCurPage(), 0,
strlen(BX_ROOT."/admin/"))!=BX_ROOT."/admin/")) &&
COption::GetOptionString("main", "include_charset", "Y")=="Y"
&& strlen(LANG_CHARSET)>0)
```

is definitely more readable when written like this:

```
$publicStatisticOnly = False;
if (defined("STATISTIC_ONLY")
 && STATISTIC_ONLY
 && substr($APPLICATION->GetCurPage(), 0,
strlen(BX_ROOT."/admin/")) != BX_ROOT."/admin/")
{
 $publicStatisticOnly = True;
}
if (!$publicStatisticOnly && strlen(LANG_CHARSET) > 0
 && COption::GetOptionString("main", "include_charset", "Y") ==
"Y")
{
}
```

or like this:

```
if (!defined("STATISTIC_ONLY") || ! STATISTIC_ONLY
 || substr($APPLICATION->GetCurPage(), 0,
strlen(BX_ROOT."/admin/")) == BX_ROOT."/admin/")
{
 if (strlen(LANG_CHARSET) > 0 &&
COption::GetOptionString("main",       "include_charset", "Y")
== "Y")
{
}
}
```

Arrays

The arrays consisting of multiple lines of code should be formatted like this:

```
$arFilter = array(
    "key1" => "value1",
    "key2" => array(
         "key21" => "value21",
         "key22" => "value22",
    )
);
```

Empty Lines And Spaces

Empty Lines

Use empty lines to logically divide your source code. Use multiple empty lines to divide the source code into logical or functional sections in one file. Use a single empty line to separate methods, as well as expressions and statements within a method for better readability.

It is recommended to add a comment before a logical or functional section (see Comments).

Spaces

The comma must be followed by the space. The semicolon must be followed by the space unless it is the last character on the line (for example, in a complex “for” statement). No spaces before the comma or semicolon is allowed. Tabs must not be used instead of spaces in such cases.

Use Cases

The following example shows how a space is used after the commas, but not before the parenthesis:

```
TestMethod($a, $b, $c);
```

These two code fragments are formatted incorrectly:

```
TestMethod($a,$b,$c);
```

and

```
TestMethod( $a, $b, $c );
```

The following example shows to use spaces to properly separate the operators:

```
$a = $b * $c / $d;
```

as opposed to the same code formatted incorrectly:

```
$a=$b*$c/$d;
```

Use spaces to format the “for” statements:

```
for ($i = 0; $i < 10; $i++)
```

Do *not* merge operators and expressions like this:

```
for($i=0;$i<10;$i++)
```

Note that using tabs to format expressions inside statements is not allowed.

The following formatting should not be made a common practice:

```
$arArray = array(
 "key1" => "value1",
 "key2" => "value2",
);
```

There is no special rule regarding the use of the space, or lack thereof, after the “if” statement.

Other regulations

Use parentheses to group operators in complex expressions regardless of operator precedence for better readability.

For example:

```
$r = $a + ($b * $c);
```

Chapter 2.
Naming Conventions

General Provisions

Do not use underscores in the identifier names because it makes them longer and less readable. Use such names that can describe the identifier purpose and meaning in an unambiguous fashion.

At the same time, try to make your identifier shorter (but they still must be well readable).

Variable Names

Start with a lowercase character and use uppercase character as separators (camelCase).

For example:

```
$testCounter, $userPassword
```

Method Names

Start with a uppercase character and use uppercase character as separators (Pascal style).

Examples:

```
CountVariable, ChangeUserPassword
```

Variable Prefixes

PHP is a loosely typed language. It has only three type groups that are distinguished specifically: scalars, arrays and objects.

Array names should begin with the “ar” prefix and start other words with an uppercase character. For example: $arResult, $arModifiedUsers.

Object names should begin with the “ob” prefix and start other words with an uppercase character. For example: $obElement, $obUser.

Objects of the CDBResult type should begin with the “db” prefix and start other words with an uppercase character. For example: $dbResult.

Variables of scalar types may be prepended with a prefix only if their type is exact and fixed.

For example, in the following code:

```
$userID = $_REQUEST["var"];
$userID = IntVal($userID);
```

the variable $userID has no prefix because its type may change at runtime.

However, in the code:

```
$bFlag = (($aaa > 0)? True : False);
```

the variable $bFlag has the "b" prefix because its type is known and is unlikely to be changed.

Class Names

Class names must begin with the "C" character. If the class is a part of a module, the next parts of the name should uniquely identify the module. Use uppercase character as separators. For example:

```
CIBlockElement, CIBlockType, CSaleAffiliate
```

If a class is a base class that has specializations for different databases, the base class must begin with "CAll". For example:

```
Example CAllSaleAffiliate.
```

Class Member Access Control

Because PHP does not provide any class member access control mechanism, the developers should stick to the following rules.

- Private member variables and methods which must not be accessed by anyone (neither by the public section calls nor by the other modules), should begin with the two underscores. For example: __CheckEmail, __arData. Such members are never described in the documentation and may be changed or deleted disregarding backward compatibility.
- Internal member variables and methods which can be accessed by the other modules but must not be accessed by the public section, should begin with one underscore. For example: _CheckEmail, _arData. Such members are not described in the public documentation (but can be described internally), and may be changed or deleted disregarding backward compatibility with prior notification sent to all affected developers.
- Other methods and variables are public; they should be described in the public documentation and cannot be changed without providing backward compatibility.

Constants

Constants must be in uppercase and begin with the "BX_" prefix. For example:

```
BX_ROOT, BX_FILE_PERMISSIONS
```

Chapter 3. Comments

Put a comment before the class or method declaration to describe the purpose of the latter.

Avoid obvious comments like

```
$i = $i + 1; // increment i
```

Add comments to every public class and method.

If you divide the code into logical sections, add comments to each section and describe its purpose.

All comments must be in English.

Chapter 4. Idioms

General Provisions

Any programming language has idioms specific to that particular language. An idiom can be a commonly used expression, an iteration technique etc. For example, some of the PHP idioms are the array iteration which is usually written as:

```
for ($i = 0, $cnt = count($arArray); $i < $cnt; $i++)
{
}
```

or as:

```
foreach ($arArray as $key => $value)
{
}
```

The use of idioms helps other developers skip obvious patterns while concentrating on the important chunks of code, or find the required code fragments by typical patterns (idioms).

To summarize, use widely used patterns and idioms instead of reinventing the wheel.

For example, the following code:

```
reset($arHashLink);
while(list($hash, $arData)=each($arHashLink))
{
}
```

should be rewritten as:

```
foreach ($arHashLink as $hash => $arData)
{
}
```

Examples Of Idioms

Conditional operator idiom "?":

```
$res = ($bTrue ? "True" : "False");
```

Chapter 5.
SQL Queries

Each of the queries: SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING should begin at a new line.

Use the same new line rule as in PHP code: a new line and a tab.