Programming style guide

2 December 2016

Aim of this document

Following a style guide will increase the readability of the code when other people look at them.

Some of these guidelines ensure that code is easily edited in more than one environment, from a large screen to a tiny terminal; other guidelines set a standard format so that all programmers know what to expect when looking at each other's code. In some cases, these requirements are no better and no worse than some of the alternatives, but mixed styles will reduce readability and reliability.

Contents

1. Project

Folder structure

/db	Logic Database manipulation and logic (knowledge of the data model) per table in one separate php-file. Never mingle with scripts producing output.
/inc	Framework, cq function libraries
/config	Configuration files
/_logs	Log files, for verification and debugging.
/_cache	Cached files

Names starting with ~ should be checked in into SVN, but will not be copied to the production server.

Names starting with _ should not be checked in into SVN and not be synchronized with other machines. They should be created on the production server, preferable by program code.

Files

Use lower-case in all file names.

All files with PHP code must have .php as extension.

Character set

All files and all output should be UTF-8, unless there is a specific reason to choose another character set.

Use three-tier model (software architecture pattern).

Separate code in:

the user interface
functional process logic ("business rules")
framework: computer data storage and data access

Separate structure (html), styling (css) and behavior (JavaScript):

easier to read
easier to maintain
easier to find bugs
easier to re-use code

So instead of:

```
<select name="sid" onchange="window.location = 'index.php?sid=' +
this.value;" style="width:10em"></select>
```

Do so:

```
<style>
.text-xl{
  width:10em;
}
</style>

<select class="text-xl" name="sid" id="sid"></select>

<script>
document.getElementById('sid').onchange = function(){
  window.location = 'index.php?sid=' + this.value;
};
</script>
```

2. Programming

Indentation

Use tabs, not spaces, for indentation. Always indent blocks of code one tab (and only one tab) beyond their parent structure.

Never combine multiple statements on the same line.

Opening braces go on the same line as their control structure. A closing brace on its own should mean that this block has ended and there are no more conditions to test. For example:

```
foreach( $arr as $value ){
    printf("Value: %s<br />\n", $value);
}
```

Function Calls

No spaces between function names and the opening parentheses of the parameter list, and use spaces after commas in parameter lists. Use spaces around operators.

```
$var = foo($bar, $baz, $quux);
```

Operators

Use spaces around operators (=+-*/%). In the case of a block of related assignments, more space may be inserted to promote readability:

```
$short         = foo($bar);
$long_variable = foo($baz);
$combined      = $short . $longvariable;
$combined     .= ' (c)2009';
```

Statements

Use parentheses (although PHP doesn't require them).

```
require_once('inc/sql.php');
```

Line endings

Use Unix line endings (\n). The easiest way to do this is to set your text editor to use Unix line endings.

Arrays

Arrays should be formatted with a space separating each element and assignment operator:

```
$some_array = array('hello', 'world', 'foo' => 'bar');
```

Multi-line arrays

It is usually a good idea to keep lines short, which means that sometimes you'll want your arrays to span multiple lines. When you do this, put all of the data pieces on their own, indented, line. For example:

```
fileExtensions = (
   ("php", "Dynamic PHP page"),
   ("html", "Static HTML page"),
   ("pdf", "Adobe PDF document"),
);
```

If the scripting language allows it*, put a comma (or appropriate item delimiter) on the last line. This makes it easier to add new items and adjust the position of existing items.

* Note: not allowed in the JavaScript version implemented by IE 8-.

Control structure syntax (if, for, foreach, while, switch, etc.)

Always put conditional statements on their own line.

```
if( $options['no-red'] ){
   $ball = str_replace('red', 'grey');
}elseif( $options['use-rubber'] ){
   $ball = str_replace('plastic', 'rubber');
   $ball = str_replace('vinyl', 'rubber');
}else{
   $ball .= ' standard';
}
```

But also very short structures. Use

```
if( empty($product) ){
   $product = new Product();
}
```

instead of

```
if (empty($product)) $product = new Product;
```

Variable and function names

Variable names and function/method names must be self- explanatory, and English based.

Variables must be consistently named. Use the same style throughout the code.

Give variables a prefix to indicate the variable type:

```
$aArray
$bBoolean
$iInteger or $nNumber
$oObject
$sString
```

Do not create obscure abbreviations. If referring to an IP address, $ip might be okay. But using $qr to refer to a query result is not okay.

Both lowerCamelCase and underscore-style improve readability by marking where words begin and end. If you do not have a preference, use lowerCamelCase.

```
//lowerCamelCase (preferred)
$newsTitle = $this->shortenTitle();

//underscore style
$news_title = $this->shorten_title();
```

Functions which return booleans should be named as a question (has, is, can ...).

```
isValid();
```

Use specific names, use

```
$aUsers = getUsers();
foreach( $aUsers as $oUser ){
   //
}
```

instead of

```
$result = getUsers();
foreach($result as $item){
   //
}
```

Class names

Use UpperCamelCase for classes.

```
class InvoiceRecords{
}
```

Constant names

Constant names must be self- explanatory and English based.

Constants must be consistently named. Use the same style throughout your code. When working on someone else's code, use their style.

Use all uppercase for constants.

```
define('SESSION_NAME', 'shoppingcart');
```

Code comments

Every file must have a comment at the top describing the file's purpose. Write them so that people can understand them on their own.

Use comments to describe the intent of a function or block. Comments should explain why this code is here; the code itself should explain how it does what it does.

First write the comments, then the code.

Comments should be on a separate line immediately before the code line or block they reference.

```
if( $bAuthenticated ){
   //create session and redirect to homepage
   session_start();
   header('location: /');
}else{
   //show errormessage on login screen
}
```

When changing code maintained by more than one person or when changing someone else's code, always comment the changes you make directly above the line or function where you made the change. Include the full date (yyyy-mm-dd), and your email address. Explain the purpose of the change.

Do the same for any bug fixes you add to any code, even your own, after that code has been put into production.

Magic numbers

Every variable you use should have a known purpose. Bare constants don't display their purpose, so don't use them. For example, don't do this:

```
$newsfeed->display(3, 6);
```

but code like this:

```
//display six news items of interest to faculty
define('ROLE_CUSTOMERS', 3)
$maxItems = 6;
$newsfeed->display(ROLE_CUSTOMERS, $maxItems);
```

Quoting strings

PHP uses single quotes and double quotes to define a text string. Single quote strings are known to be faster because the parser doesn't have to look for in-line variables. Their use is preferred except in two cases:

1. If the text contains single quotes

```
print("He's an administrator.");
```

2. If you need to have a combination of text and variables

```
print("Dear $name,");
```

But in the latter case, you should consider using

```
printf('Dear %s,', htmlentities($name));
```

Type declarations (PHP)

You are encouraged to use type hinting that was introduced with PHP 5. Type hinting allows you to request function parameters passed to be of a certain type. If a wrong type is passed, PHP throws a runtime error. Type hinting only works with objects or arrays, i.e. you can specify the exact class name of which passed objects need to be an instance or you use the type array.

```
public function myFunc(MyClass $oObject, array $aValues) {
   //now we can safely assume an array
   foreach( $aValues as $sValue ){
      //...
   }
}
```

Note: `callable` is available as type declarations as of PHP 5.4.0.

Note: `bool`, `float`, `int` and `string` are available as type declarations as of PHP 7.0.0.

Be fail safe

Make sure you see all errors and warnings in the development environment

```
ini_set('display_errors', '1');
ini_set('display_startup_errors', '1');
error_reporting(E_ALL);
```

Test results before using them

```
if( $aEmployers = listEmployers() ){
   foreach( $aEmployers as $oEmployer ){
      //
   }
}
```

instead of

```
$aEmployers = listEmployers();
foreach( $aEmployers as $oEmployer ){
   //
}
```

Remove dead code

Easier to maintain the remaining code, better performance, less security risks.

3. Security

Never trust variables

Especially if it is a user contributed variable (_GET, _POST, _COOKIE, _SESSION, _SERVER).

htmlentities()

When displaying user contributed content in html, always use htmlentities() to escape special characters. This content can come from a form ($_GET or $_POST), from a url, from a cookie or sessions, or from the database.

When interpreting/parsing a url, apply urldecode() before htmlentities().

Validate user input

When possible force a variable in to the type you need before using it in SQL or in functions like inlude().

- Convert booleans to 0 or 1.
- Use intval() for all number values and the numeric parameters of LIMIT.

```
if( isset($_GET['id']) && is_numeric(($_GET['id']) ){
  $id = intval($_GET['id']);
  $sql = "SELECT * FROM `tablename` WHERE `id`='$id'";
}
```

- If identifiers (columns, tables or databases) or keywords (such as ASC and DESC) are referenced in the script parameters, make sure (and force) their values are chosen as one of an explicit set of options.

```
if( in_array( $_GET['type'], array('date', 'name') ) ){
  $sField = $_GET['type'];
}else{
  $sField = 'date';
}
if( isset($_GET['sort']) && $_GET['sort'] == 'DESC' ){
  $sOrderDirection = 'DESC';
}else{
  $sOrderDirection = 'ASC';
}
$sql = "SELECT * FROM `tablename` ORDER BY `$sField` $sOrderDirection";
```

Write properly quoted SQL

- Single quotes around values (string literals **and** numbers).
- Backtick quotes around identifiers (databases, tables, **columns, aliases**).

Properly escape the strings and numbers in SQL statements

- $db->escape for all values (string literals).
- $db->escLike to escape wildcard/regexp metacharacters for LIKE.
- Better avoid REGEXP/RLIKE.
- Limit the result-set if possible to limit the effect of a dos-attack.
  LIMIT 1 if you only need one row.

Validation is not a substitute for escaping. No matter what validation steps you take when processing the user input in your scripts, always do the escaping steps before issuing the query.

4. Database

Engine

Make sure the type of engine is explicitly defined, don't rely on defaults.

Normally you'd use the `MyISAM` engine.

Charset and collation

Make sure charset and collation are explicitly defined, don't rely on defaults.

All tables and string fields should be UTF-8, unless there is a specific reason to choose another character set.

Normally you'd use the `utf8_general_ci` collation.

utf8_general_ci: compare strings using general language rules, case-insensitive;
utf8_general_cs: compare strings using general language rules, case-sensitive;
utf8_bin: compare strings by the binary value of each character in the string.

`WHERE `firstname` = 'Bob'`, the different collations would return matches for:

utf8_general_ci: Bob, Böb, BÖB
utf8_general_cs: Bob, Böb
utf8_bin: Bob

Minimize database effort:

only put the fields you need in the SELECT
only ask the rows you need
limit the number of JOINS
limit the amount of duplicate information
only query the database if needed
COUNT(id) is faster than COUNT(*)

Naming Conventions

- Table names are plural
- Table and fields names are in English, all lowercase.
- Names of boolean fields consist of verb_noun:
  `use_shiftedrates, eligiblefor_partnerpension`
- Names of foreign key fields must start with `fk_` followed with the name of the object (singular) it refers to.
  `fk_employee` refers to `employees.id`
- Names of fields used to store a set of flags should start with `set_`
  `set_permissions, set_usergroups`
- Names of timestamp fields end with `_on`
  `deleted_on, modified_on, oploaded-on`

Primary key

Each table starts with a field `id`

```
`id` smallint(5) unsigned NOT NULL auto_increment
```

Foreign keys

The MyIsam database engine doesn't support foreign keys.

The data type of the foreign key field must equal the data type of the id field it is referring to (`smallint(5) unsigned` in most cases).

Data types

Booleans	`tinyint(1) UNSIGNED`	
Sets of max 7 flags	`tinyint() UNSIGNED`	`Don't use Mysql's SET datatype`
Sets of max 15 flags	`smallint() UNSIGNED`	~
Sets of max 31 flags	`int() UNSIGNED`	~
IP-address	`int(10) unsigned`	`Store with INET_ATON()` `Read with SELECT INET_NTOA()`
Rates and percentages	`decimal(4,2)`	
Amounts	`decimal(7,2)`	
Abbreviations	`varchar(10)`	
Guids	`varchar(16)`	`use PHP function uniqid()`
Phone numbers	`varchar(20)`	
Passwords (md5 hashed)	`varchar(32)`	
Short names	`varchar(50)`	
Long names, email addresses, urls	`varchar(250)`	
Anything longer than 255 characters	`text`	

Choosing the right integer

	max int		bytes	bits	max bit flags
TinyInt	255		1	8	7
SmallInt	65.535	64K	2	16	15
MediumInt	16.777.215	16M	3	24	23
Int	4.294.967.295	4G	4	32	31
BigInt	18.446.744.073.709.600.000		8	64	63

5. HTML and CSS

Test if the output is valid HTML and valid CSS.

- HTML should be valid HTML5 in XHTML notation, meeting WCAG 2.0 Level AA criteria.
- CSS should be valid CSS 3

Supported browsers are Internet Explorer 9.0 and the latest versions of Edge, Chrome, FireFox, and Safari.

All JavaScript and CSS in external files. So no CSS en JavaScript in the HTML directly:

```
<div style="color:red;" onclick="showWarning()"></div>
```

Give HTML classes a meaningful name, not a description of the styling.

```
class="redborder"
```

clearer and better to maintain is:

```
class="error"
```

6. JavaScript framework

JQuery is the only framework used. Avoid plug-ins, especially when they include a user interface. Extensions from JQuery.UI are ok.

Including the framework

Use a protocol relative link to a specific version on Google's CDNs. Provide a local fallback in case the CDN is not available.

```
<script
src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="/yce/js/jquery-
1.8.3.min.js">\x3C/script>')</script>


<script src="//ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-
ui.min.js"></script>
<script>window.jQuery.ui || document.write('<script src="/yce/js/jquery-ui-
1.11.4.min.js">\x3C/script>')</script>
```

Aliasing the jQuery Namespace

```
(function($){
  //jquery is ready
  //---
  //---

  $(document).ready(function(){
    //document is ready
  });
})(jQuery);
```