

TechKnowMe Code Convention Guidelines

General

We realize that every programmer has his or her own way of formatting their code, and for the most part we respect that. However, in order to maintain consistent looking code that anyone with the requisite experience can pick up and begin using, certain conventions must apply. We ask that you adhere to the following code formatting guidelines as much as reasonably possible in order to create consistent-looking maintainable code.

Philosophy

In all cases, we strive for code that is functional, maintainable and performs well, in that order of importance. Where trade-offs must occur, we prefer code that functions according to specifications above all else, with readability and maintainability coming in a close second. While we hate to sacrifice performance, we understand there is occasionally a tradeoff. Code that is well written and commented typically already performs well, but where there needs to be a compromise, we'd rather the code be easy to understand and enhance where necessary.

PHP allows for a tight coupling of display code and business logic. It's very easy to get lazy in this manner and start intermingling the two. This must be avoided at all costs. All business code - functions, classes, etc. - should never contain any formatting information for the data they display. All formatting should be left up to the display code, which typically manifests as PHP pages. Similarly, all PHP page display code, which typically appears in PHP pages or include files, should strive to minimize the amount of business logic, using classes and included functions where such code is necessary. Clearly some PHP code will need to be in these pages - include functions, looping functions, if-then statements, etc. - but these should only exist to support the display of data. Database access and data manipulation should all be handled by classes and functions written specifically for these purposes.

The reason for this separation is to make it easy for other programmers to find what they need. It seems counter-intuitive at first as doing things this way typically creates a number of files that need to be included in order to perform a particular function. However, by parceling out the code and display in separate files, we can assure that each module can stand on its own and maximize its reusability. Further, we can more easily unit test each piece when it stands on its own. If there's a bug in the way, for example, a particular application pulls data from the database, we can usually pinpoint the location of the bug by analyzing which part of the system is experiencing the problem. Once we know that, we can ignore the files that have nothing to do with that specific part of the system and fix and test it without affecting the rest of the code.

Brackets

Brackets should begin at the end of the line which declares the beginning of the block the brackets define and should end on a line by themselves.

Example:

```
function functionName() {  
    //Some code
```

```
}
```

PHP Code Block Declarations

PHP Code Block Declarations should begin and end on their own lines with no code following or preceding them. Code required to be embedded within HTML should follow the format of simple PHP output blocks (i.e. `<?= expression ?>`).

Example:

```
<?php
    //This is where some PHP code will go
?>
<p>
    <b>This variable is declared in PHP: <?= $someVar?></b><br />
<?php
    //Some more PHP code would go here
?>
    <i>This will be outputted at HTML</i>
</p>
```

Tabs

All code defined within the same block should be tabbed in at the same amount – one tab per block. Tabs are used instead of spaces as tab spacing can usually be controlled by text editing programs, where spaces can not be easily changed.

Example:

```
<?php
    //The PHP declaration counts as a block
    function someFunction() {
        if(true) {
            for($n = 0; $n < 10; $n++) {
                //some code
            }
        }
    }
?>
```

HTML

All PHP pages using HTML should strive to use as little actual PHP scripting on these pages as possible. Any substantial business logic should be extracted from these pages and placed in appropriate include files containing strictly PHP code, preferably encapsulating such code in classes where possible.

When PHP code and HTML must co-exist on a page, the tab blocks for each shall be separate so that, when the source of the interpreted HTML code is viewed, it is seen to follow the same tabbing conventions (one tab per block) described above.

Example:

```
<?php
    //This is where some PHP code will go
?>
<p>
    <b>This variable is declared in PHP: <?= $someVar?></b><br />
</p>
<?php
    //Some more PHP code would go here
?>
    <i>This will be outputted at HTML</i>
</p>
```

Documentation

All code should be peppered with comments wherever reasonable, particularly in areas where the purpose of the code may not be immediately obvious. Comments should also be used to describe any special algorithms or functions to act as both documentation as well as a “sanity check” to help testers ensure the code is working as described.

In addition, all functions and publicly accessible variables should be documented using comments in the PHPDoc style in order to help automatically create API documentation. This documentation should include descriptions for all parameters and return values as well as describe exactly what type of data each function expects to receive and what type of values, if any, it will return.

Example:

```
/**
 * Checks the provided username and password against entries in the database to confirm that the user
 * exists and that the provided password matches what is kept in the records. Function returns true
 * on a successful log in.
 *
 * @global Set the $_SESSION variable [“username”] on successful login.
 * @param string $username the user's username
 * @param string $password the user's plaintext password
 * @return boolean
 */
function loginUser ($username, $password) {
    //Some code
}
```

Variables

All variables should be named to describe as accurately as possible the type and context of the data to be used. All variable names should start with a lowercase letter. Each subsequent word in a multi-word variable should begin with a capitalized letter. Variables intended to be used as constants should either be in all uppercase letters or, preferably, defined using PHP's define() function.

Examples:

A string variable containing the user's full name: \$fullName
An integer variable storing the bitwise roles of a user: \$roles
A constant defining the name of the site: \$SITE_NAME

Functions

All functions should be named to describe as accurately as possible its return type and context. Names should start with lowercase letters. Functions with multi-word names should use capital letters for each word after the first. Parameter names should adhere to the same rules as standard variable names.

Example:

```
function getResultsArray($dbRef) {  
    //Some code  
}
```

Classes

All classes must be contained in their own include file, which should be named in the format ClassName.class.php. All classes should be named to describe as accurately as possible the type of object they represent. Each word in a class name should be capitalized. All methods and variables in the class should adhere to their appropriate style guidelines.

Where possible, class variables should only be accessed through methods designed to “set” and “get” their values rather than be directly accessed.

One special instance in classes that act as Data Access Objects is that the fields that map to fields in a data store should be handled through an associative data array. This allows easier maintenance for large numbers of fields such as appears in these datastores.

Example:

```
Class SampleClass {  
    var $myVar;  
  
    function SampleClass() {  
        $dataArray = array (  
            'firstname' => "  
            'lastname' => "  
        );  
    }  
  
    function myMethod() {  
        //Some code  
    }  
}
```