The Joomla! Platform Coding Standards

The Joomla! Platform Coding Standards

Table of Contents

List of Examples

Preface

One of the things that sets good software apart from great software is not the features or the actual function the software performs, but the quality of the source code. In order to perform in the highly competitive Open Source and propriety software industries, the source code not only needs to be beautifully designed, it also needs to be beautiful and elegant to look at.

Readable code is maintainable code and the compass that guides us in achieving that goal is a set of well thought out coding standards for the different software languages that are employed in our software project. Joomla has a solid heritage of striving to support a great looking product with great looking code. This document compiles the collective wisdom of past and present contributors to the project to form the definitive standard for coding in Joomla, whether that is for the core Joomla Platform or an extension that forms part of the stack of the Joomla CMS. It also serves as a lighthouse to the Joomla developer community, to safely guide developers around the pitfalls of becoming lackadaisical with respect to writing clean, beautiful code.

The Joomla Coding Standards borrows heavily from the PEAR coding standard for PHP files, augmenting and diverging where it is deemed sensible to do so

1

Source Code Management

1.1.

Before we start talking about what code should look like, it is appropriate to look at how and where the source code is stored. All serious software projects, whether driven by an Open Source community or developed within a company for proprietary purposes will manage the source code is some sort of source or version management system. Joomla currently employs two different systems. One system is for the files that form the product distributed as the Joomla CMS, and the other is for the files that are distributed as the Joomla Platform. Each system is described below.

1.2. The Joomla Platform

In April 2011 the Joomla project decided to formally split off the core engine that drives the Joomla CMS into a separate project with a separate development path called the Joomla Platform. The Joomla Platform is a PHP framework that is designed to serve as a foundation for not only web applications (like a CMS) but other types of software such as command line applications. The files that form the Joomla Platform are stored in a Distributed Version Control System (DVCS) called Git hosted at github.org

You can learn about how to get the Joomla Platform source code from the Git repository from the following page: <permalink to developer.joomla.org staging page>

Because Git treats the concepts of file revision numbers differently than Subversion, the repository revision number is not required in files (that is, the @version tag is not necessary).

1.3. Compliance Tool

TODO Mention something about CodeSniffer and the custom Joomla sniff standard for PHP files. The Sniff is based on the standard outlined in this document. For more information about how code standards are enforced see the analysis appendix of the manual.

2

Basic Guidelines

2.1.

This chapter outlines the basic guidelines that cover and files.

2.2. File Format

All files contributed to Joomla must be stored as ASCII text, use UTF-8 character encoding and ne Unix formatted. Lines must end only with a line feed (LF). Line feeds are represented as ordinal 10, octal 012 and hex 0A. Do not use carriage returns (CR) like Macintosh computers do or the carriage return/line feed combination (CRLF) like Windows computers do.

2.3. Spelling

The spelling of words and terms used in code comments and in the naming of class, functions, variables and constant should generally be in accordance with British English rules (en_GB). However, some exceptions are permitted, for example where common programming names are used that align with the PHP API or other established conventions such as for “color” where is it common practice to maintain US English spelling.

2.4. Indenting

Tabs are used for indenting code (not spaces as required by the PEAR standard). Source code editors or Integrated Development Environments (IDE’s) such as Eclipse must have the tab-stops for indenting measuring four (4) spaces in length.

2.5. Line Length

There is no maximum limit for line lengths in files, however, a notional value of about 150 characters is recommend to achieve a good level of readability without horizontal scrolling. Longer lines are permitted if the nature of the code for individual lines requires it and line breaks would have an adverse affect on the final output (such as for mixed PHP/HTML layout files).

2.6. Best Practices

TODO Any words of wisdom about PHP best practice, maybe even references for design patterns? Can/should we link out to the JRD, Doc wiki and Developer site for additional resources?

3
PHP

This chapter is about ...

3.1. Language Constructs

3.1.1. PHP Code Tags

Always use the full <?php ?> to delimit PHP code, not the <? ?> shorthand. This is the most portable way to include PHP code on differing operating systems and setups.

For files that contain only PHP code, the closing tag (?>) should not be included. It is not required by PHP. Leaving this out prevents trailing white space from being accidentally injected into the output that can introduce errors in the Joomla session (see the PHP manual on instruction separation at http://php.net/basic-syntax.instruction-separation).

TODO Example

3.1.2. Including Code

Anywhere you are unconditionally including a file, use require_once. Anywhere you are conditionally including a file (for example, factory methods), use include_once. Either of these will ensure that 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.

Note

include_once and require_once are PHP language statements, not functions. The correct formatting is:

require_once JPATH_COMPONENT.'/helpers/helper.php';

You should not enclose the filename in parentheses.

3.1.3. E_STRICT Compatible PHP Code

As of Joomla version 1.6 and for all versions of the Joomla Platform, adhering to object oriented programming practice as supported by PHP 5.2 is required. Joomla is committed to progressively making the source code E_STRICT.

3.2. Global Variables

TODO Usage should be kept to a minimum. Use OOP and factory patterns instead.

3.3. Control Structures

For all control structures there is a space between the keyword and an opening parenthesis, then no space either after the opening parenthesis or before the closing bracket. This is done to distinguish control keywords from function names. All control structures must contain their logic within braces.

For all all control structures, such as if, else, do, for, foreach, try, catch, switch and while, both the keyword starts a newline and the opening and closing braces are each put on a new line.

3.3.1. An if-else Example

```
if ($test)
{
 echo 'True';
}
// Comments can go here.
else if ($test === false)
{
 echo 'Really false';
}
else
{
 echo 'A white lie';
}
```

3.3.2. A do-while Example

```
do
{
    $i++;
}
while ($i < 10);
```

3.3.3. A for Example

```
for ($i = 0; $i < $n; $i++)
{
    echo 'Increment = '.$i;
}
```

3.3.4. A foreach Example

```
foreach ($rows as $index => $row)
{
    echo 'Index = '.$id.', Value = '.$row;
}
```

3.3.5. A while Example

```
while (!$done)
{
    $done = true;
}
```

3.3.6. A switch example

When using a switch statement, the case keywords are indented. The break statement starts on a newline assuming the indent of the code within the case.

```
switch ($value)
{
    case 'a':
```

```
        echo 'A';
        break;

    default:
        echo 'I give up';
        break;
}
```

3.3.7. References

When using references, there should be a space before the reference operator and no space between it and the function or variable name.

For example:

```
<?php
$ref1  = &$this->sql;
```

Note

In PHP 5, reference operators are not required for objects. All objects are handled by reference.

3.3.8. Arrays

Assignments (the => operator) in arrays may be aligned with tabs. 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.

For example:

```
$options = array(
    'foo'  => 'foo',
    'spam' => 'spam',
);
```

3.4. Code Commenting

Inline comments to explain code follow the convention for C (/* … */) and C++ single line (// ...) comments. C-style blocks are generally restricted to documentation headers for files, classes and functions. The C++ style is generally used for making code notes. Code notes are strongly encouraged to help other people, including your future-self, follow the purpose of the code. Always provide notes where the code is performing particularly complex operations.

Perl/shell style comments (#) are not permitted in PHP files but are permitted in INI language files.

Blocks of code may, of course, be commented out for debugging purposes using any appropriate format, but should be removed before submitting patches for contribution back to the core code.

For example, do not include feature submissions like:

```
<?php
// Must fix this code up one day.
//$code = broken($fixme);
```

3.4.1. Comment Docblocks

Documentation headers for PHP and Javascript code in files, classes, class properties, methods and functions, called the docblocks, follow a convention similar to JavaDoc or phpDOC.

These "DocBlocks" borrow from the PEAR standard but have some variations specific for Joomla and the Joomla Platform.

Whereas normal code indenting uses real tabs, all whitespace in a Docblock uses real spaces. This provides better readability in source code browsers. The minimum whitespace between any text elements, such as tags, variable types, variable names and tag descriptions, is two real spaces. Variable types and tag descriptions should be aligned according to the longest Docblock tag and type-plus-variable respectively.

If the @package tag is used, it will be "Joomla.Platform".

If the @subpackage tag is used, it is the name of the top level folder under the /joomla/ folder. For example: Application, Database, Html, and so on.

Code contributed to the Joomla project that will become the copyright of the project is not allowed to include @author tags. You should update the contribution log in CREDITS.php. Joomla's philosophy is that the code is written "all together" and there is no notion of any one person "owning" any section of code. The @author tags are permitted in third-party libraries that are included in the core libraries.

Files included from third party sources must leave DocBlocks intact. Layout files use the same DocBlocks as other PHP files.

3.4.2. File DocBlock Headers

The file header DocBlock consists of the following required and optional elements in the following order:

- Short description (optional unless the file contains more than two classes or functions), followed by a blank line).
- Long description (optional, followed by a blank line).
- @category (optional and rarely used)
- @package (generally optional but required when files contain only procedural code)
- @subpackage (optional)
- @author (optional but only permitted in non-Joomla source files, for example, included third-party libraries like Geshi)
- @copyright (required)
- @license (required and must be compatible with the Joomla license)
- @deprecated (optional)
- @link (optional)
- @see (optional)
- @since (generally optional but required when files contain only procedural code)

```
<?php
/**
 * @package     Joomla.Platform
 * @subpackage  Database
 * @copyright   Copyright 2005 - 2010 Open Source Matters. All rights re-served.
 * @license     GNU General Public License version 2 or later; see LICENSE.txt
 */
```

Example 3.1. Example Joomla Platform file header

3.5. Function Calls

Functions should be called with no spaces between the function name and the opening parenthesis, and no space between this and the first parameter; a space after the comma between each parameter (if they are present), and no space between the last

parameter and the closing parenthesis. There should be space before and exactly one space after the equals sign. Tab alignment over multiple lines is permitted.

```
<?php
// An isolated function call.
$foo = bar($var1, $var2);

// Multiple aligned function calls.
$short   = bar('short');
$medium  = bar('medium');
$long    = bar('long');
```

Example 3.2. Example function call

3.6. Function Definitions

Function definitions start on a new line and the opening and closing braces are also placed on new lines. An empty line should precede lines specifying the return value.

Function definitions must include a documentation comment in accordance with the Commenting section of this document.

- Short description (required, followed by a blank line)
- Long description (optional, followed by a blank line)
- @param (required if there are method or function arguments, the last @param tag is followed by a blank line)
- @return (required, followed by a blank line)
- All other tags in alphabetical order, however @since is always required.

```
<?php
/**
 * A utility class.
 *
 * @package     Joomla.Framework
 * @subpackage  XBase
 *
 * @param       string  $path  The library path in dot notation.
 *
 * @return      void
 * @since       1.6
 */
public function jimport($path)
{
    // Body of method.
}
```

Example 3.3. Example function definition with docblock

3.7. Class Definitions

Class definitions start on a new line and the opening and closing braces are also placed on new lines. Class methods must follow the guidelines for Function Definitions. Properties and methods must follow OOP standards and be declared appropriately (using public, protected, private and static as applicable).

Class definitions, properties and methods must each be provided with a DocBlock in accordance with the following sections.

3.7.1. Class DocBlock Headers

The class Docblock consists of the following required and optional elements in the fol-lowing order.

- Short description (required, unless the file contains more than two classes or functions), followed by a blank line).
- Long description (optional, followed by a blank line).
- @category (optional and rarely used)
- @package (required)
- @subpackage (optional)
- @author (optional but only permitted in non-Joomla source files, for example, included third-party libraries like Geshi)
- @copyright (optional unless different from the file Docblock)
- @license (optional unless different from the file Docblock)
- @deprecated (optional)
- @link (optional)
- @see (optional)
- @since (required, being the version of the software the class was introduced)

3.7.2. Class Property DocBlocks

The class property Docblock consists of the following required and optional elements in the following order.

- Short description (required, followed by a blank line)
- @var (required, followed by the property type)
- @deprecated (optional)
- @since (required)

3.7.3. Class Method DocBlocks

The DocBlock for class methods follows the same convention as for PHP functions (see above).

```
<?php
/**
 * A utility class.
 *
 * @package     Joomla.Framework
 * @subpackage  XBase
 * @since       1.6
 */
class JClass extends JObject
{
    /**
     * Human readable name
     *
     * @var    string
     * @since  1.6
     */
```

```
    public $name;

    /**
     * Method to get the name of the class.
     *
     * @param      string  $case  Optionally return in upper/lower case.
     *
     * @return     boolean  True if successfully loaded, false otherwise.
     *
     * @since      1.6
     */
    public function getName($case = null)
    {
        // Body of method.

        return $this->name;
    }
}
```

Example 3.4. Example class definition

3.8. Naming Conventions

3.8.1. Classes

Classes should be given descriptive names. Avoid using abbreviations where possible. Class names should always begin with an uppercase letter and be written in CamelCase even if using traditionally uppercase acronyms (such as XML, HTML). One exception is for Joomla framework classes which must begin with an uppercase 'J' with the next letter also being uppercase.

For example:

- JHtmlHelper
- JXmlParser
- JModel

3.8.2. Functions and Methods

Functions and methods should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). The initial letter of the name is lowercase, and each let-ter that starts a new "word" is capitalized. Function in the Joomla framework must begin with a lowercase 'j'.

For example:

- connect();
- getData();
- buildSomeWidget();
- jImport();
- jDoSomething();

Private class members (meaning class members that are intended to be used only from within the same class in which they are declared) are preceded by a single underscore. Properties are to be written in underscore format (that is, logical words separated by underscores) and should be all lowercase.

For example:

```
class JFooHelper
{
    private $_status = null;

    protected $field_name = null;

    protected function sort()
    {
    }
}
```

Example 3.5. Example class with private properties (DocBlocks excluded for clarity)

Note

Private class methods, and properties, should be avoided unless absolutely necessary as they present difficulties when writing unit tests.

3.8.3. Constants

Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercase name of the class/package they are used in. For example, the constants used by the JError class all begin with "JERROR_".

3.8.4. Global Variables

Do not use global variables. Use static class properties or constants instead of globals.

3.8.5. Regular Variables and Class Properties

Regular variables, follow the same conventions as function.

Class variables should be set to null or some other appropriate default value.

3.9. Error Handling

Exceptions should be used for error handling.

3.10. SQL Queries

SQL keywords are to be written in uppercase, while all other identifiers (which the ex-ception of quoted text obviously) is to be in lowercase.

All table names should use the #__ prefix rather than jos_ to access Joomla contents and allow for the user defined database prefix to be applied. Queries should also use the JDatabaseQuery API.

```
// Get the database connector.
$db = JFactory::getDBO();

// Get the query from the database connector.
$query = $db->getQuery(true);

// Build the query programatically (using chaining if desired).
$query->select('u.*')
    // Use the qn alias for the quoteName method to quote table names.
```

```
    ->from($db->qn('#__users').' AS u'));

// Tell the database connector what query to run.
$db->setQuery($query);

// Invoke the query or data retrieval helper.
$users = $db->loadObjectList();
```

Example 3.6. Example query