

# Joomla! Coding Standards

Donnerstag, 28. September 2006

Original: [http://dev.joomla.org/component/.../joomla\\_coding\\_standards](http://dev.joomla.org/component/.../joomla_coding_standards)

Gute Codingstandards sind wichtig in jedem Entwicklungsprojekt, aber besonders wenn mehrere Entwickler an demselben Projekt arbeiten. Codingstandards helfen sicherzustellen, dass der Code von hoher Qualität ist, weniger Bugs hat und leicht zu verwalten ist.

{tab=Format}

Joomla! wird an den PEAR coding standards festhalten. Du kannst sie Dir auf [pear.php.net](http://pear.php.net) ansehen, oder sie hier lesen:

Version: September 10, 2005, 01:57:59 PM

Pear Coding Standards

Zeileneinzug und Zeilenlänge

Benutze einen Zeileneinzug von 4 Leerzeichen, ohne Tabs. Wenn Du Emacs zum Bearbeiten von PEAR Code nutzt, solltest Du set indent-tabs-mode auf nil setzen. Hier ist ein Beispiel mode hook, der Emacs auf diese Richtlinie einstellt (Du solltest sicher stellen, dass dies aufgerufen wird, wenn Du PHP Dateien bearbeitest):

Code:

```
(defun php-mode-hook ()
  (setq tab-width 4
        c-basic-offset 4
        c-hanging-comment-ender-p nil
        indent-tabs-mode
        (not
         (and (string-match "\\(PEAR\\|pear\\)/" (buffer-file-name))
              (string-match "\\.php$" (buffer-file-name))))))
```

Hier sind die vim Regeln dafür:

Code:

```
set expandtab
set shiftwidth=4
set softtabstop=4
set tabstop=4
```

Es wird empfohlen Zeilen nach maximal 75-85 Zeichen umzubrechen. Es gibt keine Standardregeln für das Umbrechen von Zeilen (Unter Windows haben sich 80 Zeichen/Zeile bewährt), nutze Dein Beurteilungsvermögen, und wenn du unsicher bist, frage in der PEAR Quality Assurance Mailingliste nach.

{tab=Kontrollstrukturen}

Kontrollstrukturen

Diese beinhalten if, for, while, switch, etc. Hier ist ein Beispiel für eine if Abfrage, da sie die Komplexeste ist:

Code:

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

Anweisungen sollten ein Leerzeichen zwischen dem Kontrollschlüsselwort und der sich öffnenden runden Klammer haben, um sich von den Funktionsaufrufen zu unterscheiden.

Es wird stark empfohlen geschweifte Klammern zu nutzen, auch wo sie technisch gesehen nur optional sind. Sie einzusetzen erhöht die Lesbarkeit und verhindert gleichzeitig logische Fehler, die sich einschleichen können, wenn neue Zeilen hinzugefügt werden.

Für die Switch Anweisung:

```
Code:
<?php
switch (condition) {
case 1:
    action1;
    break;
case 2:
    action2;
    break;
default:
    defaultaction;
    break;
}
?>
```

{tab=Funktionen}

Funktionsaufrufe

Funktionen sollten ohne Leerzeichen zwischen dem Funktionsnamen und der sich öffnenden runden Klammer, und dem ersten Parameter aufgerufen werden; Leerzeichen zwischen Kommas und jedem Parameter, und keine Leerzeichen zwischen dem letzten Parameter und der sich schliessenden Klammer, und dem Semikolon Hier ein Beispiel:

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

Wie oben gezeigt, sollte ein Leerzeichen auf jeder Seite der Zuweisung des Rückgabewertes an eine Variable sein. Im Falle, dass ein Block von Anweisungen kommt, können mehr Leerzeichen eingefügt werden, um die Lesbarkeit des Codes zu verbessern:

```
Code:
<?php
$short      = foo ( $bar );
$long_variable = foo ( $baz );
?>
```

Funktionen Definieren

Funktionsdeklarationen folgen der "one true brace" Konvention:

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

Funktionsparameter mit Standardwerten werden ans Ende der Parameterliste gesetzt. Es sollte immer versucht werden einen aussagekräftigen Wert aus einer Funktion zurückgeben zu lassen, wenn angebracht. Hier ein etwas größeres Beispiel:

```
Code:
<?php
function connect(&$dsn, $persistent = false)
{
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
?>
```

{tab=Code, Tags, URLs}

Code hinzufügen

Überall wo du ohne Vorbehalt eine Klassendatei einfügst, nutze `require_once()`. Überall wo du unter Vorbehalt eine Klassendatei einfügst (z.B. Factory Methoden), nutze `include_once()`. Beide stellen sicher, dass die Klassendatei nur einmal eingefügt wird. Sie benutzen die gleiche Dateiliste, daher musst Du Dir keine Sorgen über die gemischte Verwendung beider machen - eine Datei die mit `require_once()` eingefügt wird, wird nicht noch einmal mit `include_once()` eingefügt.

Anmerkung: `include_once()` und `require_once()` sind Anweisungen, keine Funktionen. Es werden keine runden Klammern um die Dateinamen der Klassendateien benötigt.

PHP Code Tags

Nutze immer `<?php ?>`, um PHP code auszuweisen, nicht die `<? ?>` Kurzform. Dies wird für die PEAR Kompatibilität benötigt und ist der sicherste Weg PHP Code auch auf anderen Betriebssystemen und Konfigurationen zu portieren.

Beispiel URLs

Benutze "beispiel.com", "beispiel.org" und "beispiel.net" für alle Beispiel URLs und E-Mail Adresen, nach RFC 2606.

{tab=Kommentare}

Kommentare

Inline Dokumentation für Klassen sollten der PHPDoc Konvention folgen, ähnlich der Javadoc. Mehr Informationen zu PHPDoc können hier gefunden werden: <http://www.phpdoc.org/>

Siehe auch PHPDocumentor Kommentare einfügen.

Nicht-Dokumentation Kommentare werden strengstens empfohlen. Eine allgemeine anwendbare Regel ist, sich einen Codebereich anzusehen und und zu denken "Wow, Ich habe keine Lust später darin rumzusuchen und beschreibe es", Du solltest es auskommentieren, bevor du vergessen hast wie der Code funktioniert.

C Style Kommentare (`/* */`) und Standard C++ Kommentare (`//`) sind beide gut. Von der Nutzung der Perl/shell Style Kommentare (`#`) wird abgeraten.

Header Kommentar Blocks

Alle Quellcode Dateien in der Core PEAR Distribution sollten folgenden Kommentarblock im Header enthalten:

Code:

```
<?php
```

```

/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */

/**
 * Short description for file
 *
 * Long description for file (if any)...
 *
 * PHP versions 4 and 5
 *
 * LICENSE: This source file is subject to version 3.0 of the PHP license
 * that is available through the world-wide-web at the following URI:
 * http://www.php.net/license/3_0.txt. If you did not receive a copy of
 * the PHP License and are unable to obtain it through the web, please
 * send a note to license [at] php.net so we can mail you a copy immediately.
 *
 * @category CategoryName
 * @package PackageName
 * @author Original Author <authorexample.com>
 * @author Another Author <anotherexample.com>
 * @copyright 1997-2005 The PHP Group
 * @license http://www.php.net/license/3_0.txt PHP License 3.0
 * @version CVS: $Id:$
 * @link http://pear.php.net/package/PackageName
 * @see NetOther, Net_Sample::Net_Sample()
 * @since File available since Release 1.2.0
 * @deprecated File deprecated in Release 2.0.0
 */

/*
 * Place includes, constant defines and $_GLOBAL settings here.
 * Make sure they have appropriate docblocks to avoid phpDocumentor
 * construing they are documented by the page-level docblock.
 */

/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 * @category CategoryName
 * @package PackageName
 * @author Original Author <authorexample.com>
 * @author Another Author <anotherexample.com>
 * @copyright 1997-2005 The PHP Group
 * @license http://www.php.net/license/3_0.txt PHP License 3.0
 * @version Release: @package_version@
 * @link http://pear.php.net/package/PackageName
 * @see NetOther, Net_Sample::Net_Sample()
 * @since Class available since Release 1.2.0
 * @deprecated Class deprecated in Release 2.0.0
 */

class foo
{

}

?>

```

Es gibt keine feste Regel, wann ein Autor neuen Codes in die Liste der Autoren für die Quelldatei eingetragen werden soll. Allgemein sollten die Änderungen in die Kategorie "Substantiell" fallen (in etwa um die 10% bis 20% der des Codes

verändert). Ausnahmen sollten bei Umschreibungen von Funktionen gemacht werden, die einer neuen verbesserten Logik zugrunde liegen.

Einfache Code Reorganisation oder Bug Fixing würden eine Aufnahme in die Liste der Autoren nicht rechtfertigen.

Dateien, die nicht aus dem Core PEAR Repository stammen sollten einen ähnlichen Block enthalten, mit Copyright, Lizenz und den Autoren. Alle Dateien sollten die modeline Kommentare enthalten, der Konsistenz wegen.

{tab=Benennung}

Namens Konventionen

## Klassen

Klassen sollten eine beschreibenden Charakter haben. Vermeide Abkürzungen wo möglich. Klassennamen sollten immer mit einem Großbuchstaben anfangen. Die PEAR Klassenhierarchie reflektiert sich auch in den Klassennamen, jede Ebene der Hierarchie wird mit einem einzelnen Unterstrich getrennt. Beispiele guter Klassennamen sind:

Log

Net\_Finger

HTML\_Upload\_Error

## Funktionen und Methoden

Funktionen und Methoden sollten per "studly caps" Style benannt werden (auch bekannt als "bumpy case" oder "camel caps"). Funktionen sollten zusätzlich den Paketnamen als Prefix tragen, um Namenskollisionen zwischen Paketen zu vermeiden. Der Anfangsbuchstabe des Namens (nach dem Prefix) wird klein geschrieben, und jeder Buchstabe, der ein neues "Wort" beginnt wird groß geschrieben. Einige Beispiele:

connect()

getData()

buildSomeWidget()

XML\_RPC\_serializeData()

Elemente privater Klassen (also Elemente die ausschliesslich dazu gedacht sind in derselben Klasse in der sie deklariert wurden eingesetzt zu werden; PHP unterstützt derzeit noch keine wirklich erzwingbaren privaten Namensräume) wird ein Unterstrich vorangestellt. Zum Beispiel:

\_sort()

\_initTree()

\$this->\_status

Konstanten Konstanten sollten immer komplett großgeschrieben werden, mit Unterstrich um die Wörter zu trennen. Prefix Konstantennamen mit der groß geschriebenen Klasse/Paket in der sie genutzt werden. Zum Beispiel, die Konstanten die von DB:: package genutzt werden beginnen alle mit "DB\_".

## Globale Variablen

Wenn dein Paket globale Variablen benötigt, sollten deren Namen mit einem einfachen Unterstrich beginnen, gefolgt vom Paketnamen und einem weiteren Unterstrich. Zum Beispiel, das PEAR Paket nutzt eine globale Variable namens `$_PEAR_destructor_object_list`.

{/tabs}