PHP Coding-Guidelines
1. Ausgabe Oktober 2012

Der Guideline beschreibt den verwendeten Coding-Stil von PHP. Dieser Stil wird umgesetzt, falls in einem Projekt vor Ort kein anderen Vereinbarungen über vorhandene Guidelines getroffen werden.

Hinweis:
Die Regeln orientieren sich an den offiziellen Coding-Guidelines des Zend-Frameworks, der Zend Framework Coding Standard für PHP wird unter der Adresse http://framework.zend.com/manual/1.12/de/coding-standard.html beschrieben.

1. Allgemeine Richtlinien

1.1 PHP Tag-Kennzeichnung

Der Eröffnungs-Tag ist in der Form „<?php“ zu schreiben.

Das Schließen von PHP-Dokumenten mit dem -?>-Tag ist nicht erlaubt. Die Angabe wird für PHP nicht benötigt und es wird das Risiko ausgeschlossen, dass nach dem Schließen-Tag noch weitere Literale mitgespeichert werden. Nach dem Schließen-Tag verwendete Literale werden von PHP als Ausgabe betrachtet, dadurch können Fehler entstehen. PHP startet dann die Ausgabe, so können anschließend keine Header mehr gesendet werden.

Permitted: ?>

1.2. Anzahl der Leerzeichen

Es werden 4 Leerzeichen als Einrückung verwendet. Tabbs dürfen nicht zur Formatierung eingesetzt werden.

1.3. Zeichen-Codierung

Alle Datei-Inhalte werden mit UTF-8 gelesen, verarbeitet und gespeichert. Es wird zu keinem Zeitpunkt der Verarbeitung das Byte-Order-Mark verwendet. Weitere Infos zu BOM bietet z.B. der Wikipedia-Text unter http://de.wikipedia.org/wiki/Byte_Order_Mark.

1.4. Maximale Zeilen-Länge

Die maximale Zeichen-Länge darf 120 Zeichen nicht überschreiten, nach Möglichkeit werden im Höchstfall 80 Zeichen verwendet.

1.5. Zeilen-Abschluß

Es wird dabei die Original-Zend-Vorgabe umgesetzt (Quelle: http://framework.zend.com/manual/1.12/de/coding-standard.php-file-formatting.html, 27.12.2012):

„Die Zeilenbegrenzung folgt der Konvention für Unix-Textdateien. Zeilen müssen mit einem einzelnen Zeilenvorschubzeichen (LF) enden. Zeilenvorschubzeichen werden durch eine ordinale 10, oder durch 0x0A (hexadezimal) dargestellt.

Beachte: Benutze nicht den Wagenrücklauf (CR) wie in den Konventionen von Apples OS (0x0D) oder die Kombination aus Wagenrücklauf und Zeilenvorschub (CRLF) wie im Standard für das Windows OS (0x0D, 0x0A)“.

2. Namens-Konventionen

2.1 Klassen und Interfaces

Klassen-Dateien werden mit Dateiname.class.php bezeichnet, Interfaces in der Form Interface.interface.class und andere PHP-Dateien mit Dateiname.php. Da in vielen Fällen Autoloader in Verbindung mit Namespaces verwendet werden, ist der Name der Interface- und Klassen-Datei identisch mit dem Namen der jeweiligen Klasse bzw. dem Interface:

```
namespace Foo\Bar

class Test {
…
}

Führt zum Dateinamen: /Foo/Bar/Test.class.php
```

Alle Datei-, Klassen- oder Interface-Namen dürfen nur alphanumerische Zeichen, Unterstrich und den Punkt enthalten, Leerzeichen sind verboten.

2.2. Funktionen und Methoden

Der Funktions- bzw. Methoden-Name darf nur alphanumerische Zeichen enthalten und ist in der Camel-Case-Schreibweise zu realisieren:

```
public function testOfFoo () {
....
}
```

Setters und Getters sind mit mit setMethodenName bzw. getMethodenName zu benennen.

2.3. Variablen

Die Bezeichnung von Variablen darf alphanumerische Zeichen enthalten, Nummern, Unterstiche oder andere speziellere Zeichen sind nicht erlaubt. Es wird ebenfalls die Camel-Case-Schreibweise verwendet.

2.4 Konstanten

Kontanten dürfen wie Variablen nur alphanumerische Zeichen ohne Nummern enthalten. Da alle Zeichen groß zu schreiben sind, wird der Unterstrich als Trenner verwendet. Eine globale Definition mit Konstanten mit define ist verboten.

```
namespace Foo\Bar
class Test {
    const NAME_KONSTANTE = “foo“;
}
```

Alle Bezeichnungen wie z.B. für Variablen, Methoden oder Klassen besitzen einen möglichst aussagekräftiegn Namen, der seine Aufgabe möglichst gut beschreibt.

3. Code-Stil

Die Konsistenz des Code-Stiles ist durchgängig zu verwenden, diese Vorgabe hat die höchste Priorität.

3.1 Deklaration und Zugriff auf Variablen und Arrays

Variablen und Arrays müssen immer vor ihrer ersten Vernwendung deklariert werden. Beim Zugriff auf Eigenschaften ist bei Bedarf je nach Implementierung z.B. mit der isset-Methode auf Variablen oder der sizeof-Methode auf Arrays die Werte-Verfügbarkeit abzusichern.

3.2. Strings

Ein String wird immer in Single-Quotes eingefasst, wenn es sich um einen einfachen Sring handelt und keine Escape-Sequencen enthalten sind. SQL-Statements werden mit Double-Quotes eingefasst, dadurch wird die Kennzeichnung von Werten des Typs string innerhalb des Statements vereinfacht, die nicht escaped werden müssen.

Eine Variablen-Subsitution ist nicht gestattet, weil sie in Template-View-Ausgaben wie z.b. bei Magento-Templates für den Marker verwendet werden.

Bei der Verbindung von Strings wird vor und hinter dem Punkt-Operator ein Leerzeichen geschrieben. Bei Strings übner mehrere Zeilen beginnt der Teilstring mit einem Punkt-Operator, der direkt unter dem Gleichheitszeichen der Zuweisung eingerückt wird.

```
namespace Foo\Bar
Var strTest = 'Ich bin ein Beipspiel-Text, Hallo Welt'
              . $strVariable
              . 'hier geht der Text weiter';
```

Die Variable vom Typ string ist mit den Anfangs-Literalen “str” gekennzeichnet wie z.B. bei strNameStringVariable.

3.3 Numbers

Der Varibale vom Typ int ist mit „int” am Anfang zu benennen, wie z.B. bei intNameVariable.

Zwischen allen Opersatoren ist ein Leerzeichen einzufügen.

3.4 Arrays

Bei numerischen Indexes beginnt der erste Schlüsselwert mit null oder 1. Komma-separierte Werte enthalten jeweils nach dem Integer-Schlüsselwert oder dem in einfachen Anführungszeichen eingeschlossenen String-Wert ein Leerzeichen.

Mehrzeilige oder verschachtelte Arrays über mehrere Dimensionen sind einzurücken.

Beispiele:

```
$array = array('STR1', 'str2', 1, 2, 3);

$array = array(
    1, 2, 3,
    $a, $b, $c,
    200, $d, 400
);

$array = array(
    'first'  => 'firstValue',
    'second' => 'secondValue'
);
```

3.5 Klassen- und Konstanten-Eigenschaften

Getter- oder Setter-Methoden werden dann verwendet, wenn die entsprechende Injection sinnvoll ist oder keine Constructor-Methode verwendet wird. In anderen Fällen sind Klassen-Eigenschaften über den Konstruktor zu initialisieren. Bei der Deklaration ist die Sichtbarkeit der Eigenschaften anzugeben.

3.6. Methoden und Funktionen

Auch Methoden werden mit der Angabe der Sichbarkeit deklariert. Für den gesamten Code wird der K&R-Stil für die Klammerung verwendet. Vor einer sich öffnenden geschweiften Klammer wird ein Leerzeichen gesetzt.

```
class Foo {
    protected $myVar;
    const  FOO = 'undefined';
```

```
    public function getMyVar() {
        return $this->myVar;
    }

    public function helloWorld() {
        ....
    }
}
```

Für die Übergabe-Parameter von Ararys und Objekten sind Type-Hints zu verwenden.

3.7 Kontroll-Staements

Auch für Kontroll-Strukturen wird durchgängig der K&R-Stil verwendet. Ber der Verwendung von logischen Schritten inenrhalb einer Schleife oder Switch-Konstrukt ist mit -break- abzusichern. Tiefer verschachtelte Schleifen sind zu vermeiden.

4. Dokumentation

Die Dokumentation muss mit phpDocumentator kompatibel sein, für weitere Dokumentationen können zusätzlich mehrzeilige oder einzeilige Kommentare verwendet werden, wenn sie einen Mehrwert bieten.

4.1. Klassen

Die Klasse ist mit einem Doc-Block ausgezeichnet, der mindestens die Tags @author zur Autorenangabe, @package zur Paketangabe und @version zur Ausgabe der aktuellen Klassen-Version enthält. Vor der Angabe der Tags erfolgt eine Kurzbeschreibung über die Funktionalität der Klasse.

```
/*
* Ich bin ein Beschreibungs-Text
*
* @package Paketname
* @author Mustermann, Max
* @version v 1.2 2011-04-29 04:08:27
*/
```

```
class Foo {
    ....
}
```

4.2. Methoden und Funktionen

Methoden und Funktionen enthalten in ihrer Dokumentation mindestens:

- Alle Argumente
- Eine kurze Beschreibung darüber, was die Funktion macht
- Die Auflistung aller möglichen return-Werte

```
    /**
     * returns the  foo-object
     * @param string $type - the type to use
     * @param array $values - creating the object
     * @throws \Foo\Exception\Bar
     * @return \Foo\foo- the created foo-object
     */
    public function getFoo($type, $values) {
        try {
            $foo = new \Foo\fooObj($type, $values);
        } catch (Exception $e) {
            throw new \Foo\Exception\Bar($e->getMessage());
        }

        return $foo;
    }
```