PHP_CodeSniffer - Eigenen Coding Standard erstellen

In diesem Beitrag möchte ich auf PHP_CodeSniffer eingehen und wie man einen eigenen Coding Standard mit selbst definierten Regeln erstellen kann.

== Installation ==
Die Installation bezieht sich auf Ubuntu (UNIX) und kann bei anderen System abweichen. Zur Installation benötigt es eigentlich nur einem Befehl, den wir in unserer Konsole eingeben.
(Voraussetzung ist, dass PEAR auf eurem System schon installiert ist.)

Quellcode

```
1. sudo pear install PHP_CodeSniffer
```

== Anwendung ==
Der PHP_CodeSniffer sollte nun systemweit über den Befehl

Quellcode

```
1. phpcs
```

aufrufbar sein.

Es werden auch schon einige Coding Standards "out of the box" mitgeliefert. Wenn wir herausfinden wollen, welche Standards das nun sind, nutzen wir einfach folgenden Befehl:

Quellcode

```
1. phpcs -i
```

Die Ausgabe sollte in etwa so aussehen:

Quellcode

```
1. The installed coding standards are PEAR, Zend, Squiz, PHPCS and MySource
```

Wir wollen nun einfach mal eine Beispielklasse gegen den Zend Coding Standard validieren lassen.

Quellcode

```
1. # in welchem Verzeichnis wir die Datei erstellen ist egal - ich habe sie einfach mal im www-Verzeichnis erstellt
2. cd /var/www/
3. sudo gedit exampleClass.php
```

Code unserer Beispielklasse:

Quellcode

```
1. <?php
2. class ExampleClass
3. {
4. const exampleConstant = '';
6. private $exampleVar = '';
```

```
8.  public function getExampleVar()
9.  {
10. return $this->$exampleVar;
11. }
12. }
13. ?>
```

Alles anzeigen

Jetzt tippen wir in unsere Konsole:

Quellcode

```
1. phpcs --standard=zend exampleClass.php
```

Ausgabe:

Brainfuck-Quellcode

```
1.  FILE: /var/www/exampleClass.php
2.  ----------------------------------------------------------------------
3.  FOUND 8 ERROR(S) AFFECTING 7 LINE(S)
4.  ----------------------------------------------------------------------
5.  4 | ERROR | Spaces must be used to indent lines; tabs are not allowed
6.  6 | ERROR | Spaces must be used to indent lines; tabs are not allowed
7.  6 | ERROR | Private member variable "exampleVar" must contain a leading
8.  | | underscore
9.  8 | ERROR | Spaces must be used to indent lines; tabs are not allowed
10. 9 | ERROR | Spaces must be used to indent lines; tabs are not allowed
11. 10 | ERROR | Spaces must be used to indent lines; tabs are not allowed
12. 11 | ERROR | Spaces must be used to indent lines; tabs are not allowed
13. 13 | ERROR | A closing tag is not permitted at the end of a PHP file
14. ----------------------------------------------------------------------
```

Alles anzeigen

== Eigenen Coding Standard ersellen ==
Als erstes müssen wir ins PHP_CodeSniffer Verzeichnis und dort folgende Ordner und Dateien anlegen:

Quellcode

```
1. cd /usr/share/php/PHP/CodeSniffer/Standards/
2. sudo mkdir MyPhpCodingStandard
3. sudo mkdir MyPhpCodingStandard/Sniffs
4. cd MyPhpCodingStandard/
5. sudo gedit ruleset.xml
```

In der ruleset.xml Datei können wir nun festlegen, welche Regeln für unseren Standard verwendet werden:

Quellcode

```
1. <?xml version="1.0"?>
2. <ruleset name="MyPhpCodingStandard">
3. <description>MyPhpCodingStandard fuer PHP_CodeSniffer</description>
4. <rule ref="Generic.NamingConventions.UpperCaseConstantName"/>
5. <rule ref="Generic.Files.LineLength">
6. <properties>
```

```
7.  <property name="lineLimit" value="10"/>
8.  <property name="absoluteLineLimit" value="0"/>
9.  </properties>
10. </rule>
11. </ruleset>
```

Alles anzeigen

Im Prinzip haben wir damit festgelegt, dass wir zwei Sniffs für unseren Standard verwenden wollen. Zum einen nutzen wir den "UpperCaseConstantName"-Sniff, der nichts weiter macht, als zu prüfen, ob die Namen der Konstanten groß geschrieben wurden. Zum anderen nutzen wir den "LineLength"-Sniff und legen zusätzlich die Eigenschaften des Sniffs fest bzw. passen sie an. Dabei legen wir fest, dass eine Codezeile maximal 10 Zeichen enthalten darf (die 10 Zeichen sind natürlich nur exemplarisch =).
Im Prinzip haben wir damit einen vollständigen Coding Standard erstellt und können dieses wieder mit dem Befehl überprüfen, der uns die bestehenden Coding Standards anzeigt.

Quellcode

```
1. phpcs -i
```

Die Ausgabe sollte in etwa so aussehen:

Quellcode

```
1. The installed coding standards are MyPhpCodingStandard, PEAR, Zend, Squiz, PHPCS and MySource
```

Nun wollen wir natürlich unseren Standard in Aktion sehen - zuerst legen wir aber noch fest, dass unser Coding Standard als Standard genommen wird, wenn der PHP_CodeSniffer aufgerufen wird:

Quellcode

```
1. sudo phpcs --config-set default_standard MyPhpCodingStandard
2. cd /var/www/
3. phpcs exampleClass.php
```

Ausgabe:

Brainfuck-Quellcode

```
1.  FILE: /var/www/exampleClass.php
2.  ----------------------------------------------------------------------
3.  FOUND 1 ERROR(S) AFFECTING 5 LINE(S)
4.  ----------------------------------------------------------------------
5.  2 | WARNING | Line exceeds 10 characters; contains 20 characters
6.  4 | ERROR | Class constants must be uppercase; expected EXAMPLECONSTANT but
7.  | | found exampleConstant
8.  4 | WARNING | Line exceeds 10 characters; contains 28 characters
9.  6 | WARNING | Line exceeds 10 characters; contains 26 characters
10. 8 | WARNING | Line exceeds 10 characters; contains 32 characters
11. 10 | WARNING | Line exceeds 10 characters; contains 28 characters
12. ----------------------------------------------------------------------
```

Alles anzeigen

Man kann auch für seinen Standard definieren, dass dieser alle Einstellungen bzw. Sniffs eines anderen Coding

Standards als Basis verwendet:

Quellcode

```
1. <?xml version="1.0"?>
2. <ruleset name="MyPhpCodingStandard">
3. <description>MyPhpCodingStandard fuer PHP_CodeSniffer</description>
4. <rule ref=”PEAR”>
5. <exclude name=”PEAR.WhiteSpace.ScopeIndent”/>
6. </rule>
7. <rule ref="Zend.Files.ClosingTag"/>
8. </ruleset>
```

Im obigen Beispiel nehmen wie also den PEAR Coding Standard als Basis und legen fest, dass der "ScopeIntend"-Sniff nicht verwendet wird, dafür aber der Zend "ClosingTag"-Sniff. Die bereits vorhandenen Sniffs können wir uns wie folgt anzeigen lassen:

Quellcode

```
1. cd /usr/share/php/PHP/CodeSniffer/Standards/
2. sudo find ./ -name '*Sniff.php'
```

== Eigenen Sniff erstellen ==
Für unseren neuen Coding Standard wollen wir aber auch eine eigene Regel bzw. einen eigenen Sniff erstellen. Unser Sniff soll einfach nur überprüfen bzw. sicherstellen, dass der öffnende PHP-Tag (<?php) am Anfang der Datei gesetzt ist und nichts davor steht.

Quellcode

```
1. cd /usr/share/php/PHP/CodeSniffer/Standards/MyPhpCodingStandard/Sniffs/
2. sudo mkdir PHP/
3. sudo gedit PHP/PhpOpenTagAtBeginningSniff.php
```

Inhalt unseres Sniffs:

Quellcode

```
1. <?php
2. // Sniff-Klassen müssen immer das PHP_CodeSniffer_Sniff Interface implementieren
3. class MyPhpCodingStandard_Sniffs_PHP_PhpOpenTagAtBeginningSniff implements PHP_CodeSniffer_Sniff
4. {
5. // hier definieren wir, auf welche Tokens wir reagieren wollen
6. public function register()
7. {
8. return array(T_OPEN_TAG);
9. }
10. // die eigentliche Regel
12. public function process(PHP_CodeSniffer_File $phpcsFile, $stackPtr)
13. {
14. if ($stackPtr != 0)
15. {
16. $phpcsFile->addWarning('Nothing should precede the PHP open tag.', $stackPtr);
17. // wir könnten anstatt einer Warnung auch einen Fehler ausgeben lassen
18. // $phpcsFile->addError('Only full PHP opening tags are allowed.', $stackPtr);
19. }
20. }
```

```
21. }
22. ?>
```

Alles anzeigen

Die register()-Methode dient dazu festzulegen, auf welche Tokens wir reagieren wollen. Einen Überblick über die PHP-Tokens gibt es hier: PHP Token-Liste. Wir wollen also reagieren, wenn der "<?php"-Tag gefunden wird. Wenn das der Fall ist, wird die process()-Methode aufgerufen, die noch zwei Argumente übergeben bekommt - mehr dazu unter: PEAR Manual. Da die Variable "$stackPtr" der aktuellen Position des Tokens enthält, können wir damit überprüfen, ob die Position des "<?php"-Tags ungleich der Position 0 ist und geben dann eine Warnung aus, falls dass der Fall sein sollte. (Anmerkung: Diese Regel ist noch nicht sehr ausgereift, denn was ist, wenn wir nun mehrere öffnende PHP-Tags in unserer Datei nutzen?)

Als nächsten und letztes müssen wir nur noch unsere "ruleset.xml" Datei aktualisieren:

Quellcode

```
1. cd ../
2. sudo gedit ruleset.xml
```

Aktualisierter Inhalt:

Quellcode

```
1. <?xml version="1.0"?>
2. <ruleset name="MyPhpCodingStandard">
3. <description>MyPhpCodingStandard fuer PHP_CodeSniffer</description>
4. <rule ref="Generic.NamingConventions.UpperCaseConstantName"/>
5. <rule ref="MyPhpCodingStandard.PHP.PhpOpenTagAtBeginning"/>
6. <rule ref="Generic.Files.LineLength">
7. <properties>
8. <property name="lineLimit" value="10"/>
9. <property name="absoluteLineLimit" value="0"/>
10. </properties>
11. </rule>
12. </ruleset>
```

Alles anzeigen

Jetzt wird unsere neue Regel in unserem Coding Standard verwendet. Bevor den PHP_CodeSniffer aufrufen, werden wir unsere Beispielklasse anpassen:

Quellcode

```
1. cd /var/www/
2. sudo gedit exampleClass.php
```

Aktualisierter Code unserer Beispielklasse:

Quellcode

```
1. <!-- HTML-Kommentar -->
2. <?php
3. class ExampleClass()
4. {
5. const exampleConstant = '';
6. private $exampleVar = '';
```

```
9.  public function getExampleVar()
10. {
11. return $this->$exampleVar;
12. }
13. }
14. ?>
```

Alles anzeigen

Jetzt tippen wir in unsere Konsole:

Quellcode

```
1. phpcs exampleClass.php
```

Ausgabe:

Brainfuck-Quellcode

```
1.  FILE: /var/www/exampleClass.php
2.  ----------------------------------------------------------------------
3.  FOUND 1 ERROR(S) AFFECTING 7 LINE(S)
4.  ----------------------------------------------------------------------
5.  1 | WARNING | Line exceeds 10 characters; contains 23 characters
6.  2 | WARNING | Nothing should precede the PHP open tag.
7.  2 | WARNING | Nothing should precede the PHP open tag.
8.  3 | WARNING | Line exceeds 10 characters; contains 20 characters
9.  5 | ERROR | Class constants must be uppercase; expected EXAMPLECONSTANT but
10. | | found exampleConstant
11. 5 | WARNING | Line exceeds 10 characters; contains 28 characters
12. 7 | WARNING | Line exceeds 10 characters; contains 26 characters
13. 9 | WARNING | Line exceeds 10 characters; contains 32 characters
14. 11 | WARNING | Line exceeds 10 characters; contains 28 characters
15. ----------------------------------------------------------------------
```

Alles anzeigen