TYPO3-PHP Mode

An Emacs major mode for TYPO3-PHP files

Joachim Mathes (joachim_mathes@web.de)

This manual introduces a TYPO3-PHP major mode for Emacs, which is specialized on editing PHP code in the TYPO3 world.

Table of Contents

List of figures

1 Introduction

This manual describes T3PHP mode, which extends Emacs with functionality for editing TYPO3-PHP files. Of course, it is also useful for PHP coding in general.

GNU Emacs is an extensible, customizable text editor and can be obtained for free from `http://ftp.gnu.org/pub/gnu/emacs/`. See Section "Preface" in *GNU Emacs manual*. The code is written and tested on GNU Emacs version 23.1.1.

2 Installation

To install T3PHP Mode just drop file '`t3php-mode.el`' into a directory on your load-path. If you are the administrator of the system you are working on, a possible directory could be '`/usr/share/emacs/site-lisp/t3php-mode`', for example. You might byte-compile it for better performance. See Section "Byte Compilation" in *Emacs Lisp*.

To set up Emacs to automatically edit files ending in '`.php`' using T3PHP Mode, add the following lines to your '`~/.emacs`' file (GNU Emacs) or '`~/.xemacs/init.el`' file (XEmacs):

```
(setq auto-mode-alist (cons '("\\.php$" . t3php-mode) auto-mode-alist))
(autoload 't3php-mode "t3php-mode" "TYPO3-PHP input file editing mode." t)
```

If you just want to test T3PHP Mode or are not eager to modify your '`~/.emacs`' file, type *`M-x load-file`*, load the '`t3php-mode.el`' file and finally apply the mode to the current buffer by typing *`M-x t3php-mode`*.

3 Using T3PHP mode

TYPO3 Mode supports different display and editing features, which will be described in the following sections.

3.1 Customization

T3PHP Mode defines a customization group in Emacs, which is a member of the Emacs standard customization group `Languages`, whose parent Emacs group is `Programming`. Use *`M-x customize`* to browse through the full list of customization groups or *`M-x customize-group`* with parameter '`t3php`' to enter the T3PHP Mode customization group directly

The following user-customizable variables are provided.

`string t3php-developer` [Variable]
The current developer or author of the PHP code. The value of this variable is used in the php-doc comments.

Default: '`Lisa Fremont <lisa@fremont.de>`'

`string t3php-typo3-extension-directory` [Variable]
The path to the TYPO3 extensions' directory relative to the TYPO3 root directory.

Default: '`ext/`'

`string t3php-path-to-typo3-extension-directory` [Variable]
The path to the TYPO3 extensions directory relative to the TYPO3 root directory.

Default: '`typo3conf/`'

`string t3php-date-format` [Variable]
The date format which is used for php-doc comments.

Default: '`%Y-%m-%d`'

`string t3php-year-format` [Variable]
The year format which is used in php-doc comments.
Default: '`%Y`'

`integer t3php-block-indentation` [Variable]
The indentation relative to a predecessing line which begins a new code block.
Default: '`4`'

`choice t3php-newline-function` [Variable]
This variable decides which function to call upon pressing RET. Depending on the chosen option, line indentation will be processed automatically. The following options are available for *choice*:

`newline` This command just inserts newlines into the current buffer before point.

`newline-and-indent`
This function inserts a newline[1], then indents the new line (the one following the newline just inserted) according to T3PHP Mode's internal indentation strategies.

`reindent-then-newline-and-indent`
This command reindents the current line, inserts a newline at point, and then indents the new line (the one following the newline just inserted).

Default: '`newline`'.

`boolean t3php-php-doc-align` [Variable]
Determines whether the PHPDoc parameters are aligned after a method or class is inserted with `t3php-insert-method` or `t3php-insert-class` respectively. See Section 3.4.1 [Inserting methods], page 5 and Section 3.4.2 [Inserting classes], page 6, for more information.
Aligned parameters:

```
1  /**
2   * Define foobar
3   *
4   * @param   $foo
5   * @param   $bar
6   * @return
7   * @author  Lisa Fremont <lisa@fremont.de>
8   * @since   2009-12-02
9   */
10 public function foobar($foo, $bar) {
11
12 }
```

Unaligned parameters:

```
1  /**
2   * Define foobar
3   *
4   * @param $foo
5   * @param $bar
6   * @return
7   * @author Lisa Fremont <lisa@fremont.de>
8   * @since 2009-12-02
9   */
10 public function foobar($foo, $bar) {
11
12 }
```

Default: '`t`'

[1] Note the different meanings of *newline* and *new line*, that goes with their different spellings. *newline* refers to the code representation of a new line, while *new line* means its interpretation as a visible new line in the editor after pressing RET.

`string t3php-php-manual-url` [Variable]
URL at which to find PHP manual. You can replace '`en`' with your ISO language code.

Default: '`http://www.php.net/manual/en/`'

`string t3php-php-search-url` [Variable]
URL at which to search for documentation on a word, e.g. a PHP function.

Default: '`http://www.php.net/manual/en/`'

`boolean t3php-outline-keep-other-windows` [Variable]
When the outline is viewed, the boolean value of this variable decides if the currently selected T3PHP Mode buffer will be deleted. If the variable is set to true ('`t`') the current window is split and the outline is displayed in the newly created window. When `nil`, all other windows except the selected one will be deleted, so that the outline window fills an adjustable (see [t3php-outline-split-windows-fraction], page 4) amount of the frame.

Default: '`t`'.

`boolean t3php-outline-split-windows-horizontally` [Variable]
This variable determines the splitting mode, if the outline is displayed. If true, split the current window horizontally. Otherwise split the window vertically.

Default: '`nil`'.

I must admit, that sometimes I get confused. So if it is not for your orientation, at least it is for mine. :-)

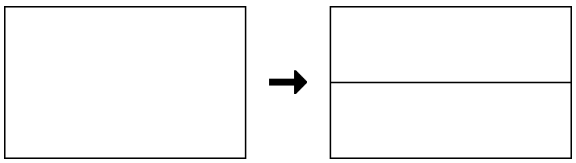

Figure 3.1: Splitting a window *vertically*.

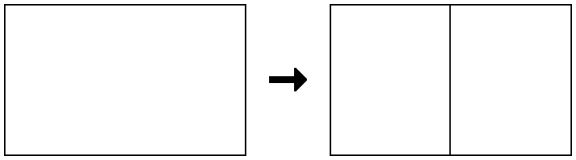

Figure 3.2: Splitting a window *horizontally*.

`number t3php-outline-split-windows-fraction` [Variable]
This variable describes the fraction of the width or height of the window, which is used for the outline.

Default: '`.3`'.

`boolean t3php-outline-follow-mode` [Variable]
If true, point in outline buffer will make T3PHP window to follow. It will show the corresponding part of the document. This flag can be toggled from within the outline buffer with the `f` key.

Default: '`nil`'.

`color t3php-outline-hl-line-color` [Variable]
The color used to highlight the background of the horizontal line which marks the current line of the outline buffer and the corresponding line in the T3PHP buffer when follow mode is active.

Default: '`DarkSlateBlue`'.

`color t3php-outline-method-name-color` [Variable]
The color used to highlight the lines in the outline, which refer to the corresponding method.

Default: '`sea green`'.

3.2 Syntax highlighting

Syntax highlighting is a convenient feature of an editor to improve the appearance hence the readability of code. TYPO3 Mode supports the highlighting of single line and multiline comments, keywords and several syntactic structures.

In XEmacs syntax highlighting should be enabled automatically. In GNU Emacs you may have to add these lines to your '`~/.emacs`' file:

```
(global-font-lock-mode t)
(setq font-lock-maximum-decoration t)
```

3.3 Line indentation

A PHP code line is indented automatically when the TAB key is pressed while point is on the according line. The indentation takes place with respect to the indentation of previous lines. See Section 3.1 [Customization], page 2, for details. The default width is `4`.

Depending on the current value of the customizable variable *t3php-newline-function*, line indentation might be executed automatically after pressing RET. See Section 3.1 [Customization], page 2, for details.

3.4 Inserting code templates

The following sections describe functions and their respective key bindings to insert templates for methods and classes.

3.4.1 Inserting methods

`t3php-insert-method` [Interactive Command]
Insert the rudimentary method framework code with respect to TYPO3 coding standards. The function asks interactively for the method name, an auto-completed modifier (`public`, `protected`, `private`) and method arguments until an empty string is provided. The inserted code consists of

- a class header comment in PHPdoc style which contains
  - empty lines for method descriptions
  - `@param` and variable name
  - `@return`
  - `@author` (default: '`t3php-developer`')
  - `@since` (default: '`t3php-date-format`')
- the method signature

Key binding: *C-c C-if*.

The following example was inserted with parameters

- Method name: `foobar`
- Modifier: `public`
- Parameter: `foo`
- Parameter: `bar`

```
1  /**
2   *
3   *
4   * @param $foo
5   * @param $bar
6   * @return
7   * @author Lisa Fremont <lisa@fremont.de>
8   * @since 2009-12-02
9   */
10 public function foobar($foo, $bar) {
11
12 }
```

3.4.2 Inserting classes

`t3php-insert-class` [Interactive Command]

Insert the rudimentary class framework code with respect to TYPO3 coding standards. The inserted code consists of

- PHP tags
- a copyright notice where date and author are inserted automatically with respect to customizable variables '`t3php-year-format`' and '`t3php-developer`'. See [t3php-year-format], page 3 and [x-t3php-developer], page 2, for more information.
- a marker comment for the class and function index which can be inserted by TYPO3 extension *extdeval*. The following empty lines might be filled with required files.
- a class header comment in PHPdoc style which contains
  - `@author` (default: '`t3php-developer`')
  - `@package` (default: '`TYPO3`')
  - `@subpackage` (default: empty)
  - `@since` (default: '`t3php-date-format`')
- the class definition
- a TYPO3 XCLASS inclusion code snippet for class extension and/or overriding. The associative array `TYPO3_CONF_VARS` has a key which refers to the current class path and file name, which is inserted dynamically. The information is extracted from the customizable variables '`t3php-path-to-typo3-extension-directory`' and the buffer file name. See [t3php-typo3-extension-directory], page 2, for more information.

Key binding: *C-c C-ic*.

The following code example was inserted in an empty file named '`class.bar.php`':

```
1  <?php
2  /***************************************************************
3   *  Copyright notice
4   *
5   *  (c) 2009 Lisa Fremont <lisa@fremont.de>
6   *  All rights reserved
7   *
8   *  This script is part of the TYPO3 project. The TYPO3 project is
9   *  free software; you can redistribute it and/or modify
10  *  it under the terms of the GNU General Public License as published by
11  *  the Free Software Foundation; either version 2 of the License, or
12  *  (at your option) any later version.
13  *
14  *  The GNU General Public License can be found at
15  *  http://www.gnu.org/copyleft/gpl.html.
16  *
17  *  This script is distributed in the hope that it will be useful,
18  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
19  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
20  *  GNU General Public License for more details.
21  *
22  *  This copyright notice MUST APPEAR in all copies of the script!
23  ************************************************************/
24 /**
25  * [CLASS/FUNCTION INDEX of SCRIPT]
26  *
27  * Hint: use extdeveval to insert/update function index above.
28  */
29
30
31
32
33
34 /**
35  *
36  *
37  * @author     Lisa Fremont <lisa@fremont.de>
38  * @package    TYPO3
39  * @subpackage
40  * @since      2009-12-02
41  */
42 class tslib_content {
43
44 }
45
46
47 /*****************************************************************************
48  * TYPO3 XCLASS INCLUSION (for class extension/overriding)
49  *****************************************************************************/
50 if (defined('TYPO3_MODE') && $TYPO3_CONF_VARS[TYPO3_MODE]['XCLASS']['ext/foo/class.bar.php']) {
51     include_once($TYPO3_CONF_VARS[TYPO3_MODE]['XCLASS']['ext/foo/class.bar.php']);
52 }
53 ?>
```

3.5 Buffer outline

The outline buffer summarizes the methods of the PHP buffer and provides comfortable navigation through the PHP buffer. The first three lines are intangible and show the file name of the current PHP input file and the shortcuts of the most important commands.

Each method in the current PHP buffer is referred to by its name. The maximally displayed line width of a method name is limited to 40 characters. Thus, a longer name is chopped to a length of 40 characters and ellipse characters (...) are concatenated. When you move the mouse cursor over a method name the complete name is displayed nevertheless. On graphical displays, it is displayed as a *tooltip* or *balloon help* as it is called sometimes. On some systems, it is shown in the echo area. On text-only terminals, Emacs can't follow the mouse, and therefore is not able to show the complete name on mouse-over.

The character and color of the leftmost column indicates the type of method modifier:

`private` `[-]` (color: '`firebrick`')

`protected`
`[#]` (color: '`goldenrod`')

`public` `[+]` (color: '`forest green`')

The rightmost column presents the line numbers, which delimit the region of the respective measurement block.

```
TABLE OF CONTENTS of ~/public_html/typo3/t3lib/class.t3lib_extmgm.php
SPC=view TAB=goto RET=goto+kill [f]ollow [r]escan  [k]ill [?]Help
```

```
------------------------------------------------------------------------
[P] isLoaded                                    [  131-136  ]
[P] extPath                                     [  147-155  ]
[P] extRelPath                                  [  165-172  ]
[P] siteRelPath                                 [  182-185  ]
[P] getCN                                       [  194-197  ]
[P] addTCAcolumns                               [  227-235  ]
[P] addToAllTCAtypes                            [  251-304  ]
[P] allowTableOnStandardPages                   [  315-320  ]
[P] addModule                                   [  332-378  ]
[P] addModulePath                               [  390-395  ]
[P] insertModuleFunction                        [  412-421  ]
[P] addPageTSConfig                             [  431-435  ]
[P] addUserTSConfig                             [  445-449  ]
[P] addLLrefForTCAdescr                         [  460-472  ]
```

`t3php-outline` *rescan* [Interactive Command]

This command shows the outline of the current PHP buffer. If *rescan* is true, the PHP buffer will be rescanned before the ouline buffer is displayed.

Key binding *C-ct*.

The outline buffer is a read-only buffer with few dedicated key bindings:

n	Move to the next selectable item.
RIGHT	Move to the next selectable item.
DOWN	Move to the next selectable item.
p	Move to the previous selectable item.
LEFT	Move to the previous selectable item.
UP	Move to the previous selectable item.
C-HOME	Move to the beginning of the outline buffer.
C-END	Move to the end of the outline buffer.
SPC	Show the corresponding location of the TYPO3 buffer.
TAB	Go to the corresponding location of the TYPO3 buffer and keep the outline window.
RET	Go to the corresponding location of the TYPO3 buffer and hide the outline window.
f	Toggle follow mode.
r	Re-parse the TYPO3 buffer.
k	Kill the outline buffer.
?	Show a help buffer.

Appendix A TYPO3 Coding Guidelines

This appendix refers to the section *PHP file formatting* of the manual *TYPO3 Coding Guidelines* by the *TYPO3 Core Development Team*. I decided to append this documentation to have a reference nearby, whenever I think of extending any T3PHP Mode features.

A.1 General requirements

A.1.1 PHP Tags

Each PHP file in TYPO3 must use full PHP tags. There must be exactly one pair of opening and closing tags (no closing and opening tags in the middle of the file). Example:

```
<?php
    // File content goes here
?>
```

There must be no empty lines after the closing PHP tag. Empty lines after closing tags break output compression in PHP and/or result in AJAX errors.

A.1.2 Line breaks

TYPO3 uses Unix line endings (`\n`, PHP `chr(10)`). If a developer uses Windows or Mac OS X platform, the editor must be configured to use Unix line endings.

A.1.3 Line length

Line length is limited to 80 characters. Longer lines should be split to several lines. Each line fragment starting from the second must be indented with one or more tab characters. Example:

```
$rows = $GLOBALS['TYPO3_DB']->exec_SELECTgetRows('uid, title', 'pages',
            'pid=' . $this->fullQuoteStr($this->pid, 'pages') .
            $this->cObj->enableFields('pages'), '', 'title');
```

In certain cases splitting lines exactly at 80 characters is not feasible. In this case lines can be longer but such practice is discouraged.

A.1.4 Whitespace and indentation

TYPO3 uses tab characters to indent source code. One indentation level is one tab. There must be no white spaces in the end of a line. This can be done manually or using a text editor that takes care of this.

Spaces must be added:

- on both sides of string, arithmetic, assignment and other similar operators (for example, `.`, `=`, `+`, `-` , `?`, `:`, `*`, etc)
- after commas
- in single line comments after the comment sign (double slash)
- after asterisks in multiline comments

A.1.5 Character set

TYPO3 PHP files use iso-8859-1 character set.

All XML files use UTF-8 character set.

A.2 File structure

TYPO3 files use the following structure:

1. Opening PHP tag
2. Copyright notice

3. File information block (with optional function index) in phpDoc format
4. Included files
5. Class information block in phpDoc format
6. PHP class
7. XCLASS declaration
8. Optional module execution code (for example, in eID classes)

The following sections discuss each of these parts.

A.2.1 Copyright notice

TYPO3 is released under the terms of GNU General Public License version 2 or any later version. The copyright notice with a reference to the GPL must be included at the top of every TYPO3 PHP class file. `user_` files must have this copyright notice as well. Example:

```
<?php
/***************************************************************
 *  Copyright notice
 *
 *  (c) YYYY Your name here (your@email.here)
 *  All rights reserved
 *
 *  This script is part of the TYPO3 project. The TYPO3 project is
 *  free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  The GNU General Public License can be found at
 *  http://www.gnu.org/copyleft/gpl.html.
 *  A copy is found in the textfile GPL.txt and important notices to the license
 *  from the author is found in LICENSE.txt distributed with these scripts.
 *
 *
 *  This script is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  This copyright notice MUST APPEAR in all copies of the script!
 ***************************************************************/
```

This notice may not be altered except for the year, author name and author e-mail.

A.2.2 File information block

File information block follows the copyright statement and provides basic information about the file. It should include file name, description of the file and information about the author (or authors). Example:

```
/**
 * class.tx_myext_pi1.php
 *
 * Provides XYZ plugin implementation.
 *
 * $Id: class.tx_myext_pi1.php 9514 2008-07-23 17:06:12Z john_doe $
 *
 * @author John Doe <john.doe@example.com>
 */
```

The example above uses SVN Id meta keyword. SVN will expand it to a full version string when the file is committed. The file information block can also contain the optional function index. This index is created and updated by the extdeveval extension.

A.2.3 Included files

Files are included using `require_once` function. All TYPO3 files must use absolute paths in calls to `require_once`. There are two ways to obtain the path to the included file:

1. Use one of the predefined TYPO3 constants: `PATH_tslib`, `PATH_t3lib`, `PATH_typo3`, `PATH_site`. The first three contain absolute paths to the corresponding TYPO3 directories. The last constant contains absolute path to the TYPO3 root directory. Example:

   ```
   require_once(PATH_tslib . 'class.tslib_pibase.php');
   ```

2. Use `t3lib_extMgm::extPath()` function. This function accepts two arguments: extension key and path to the included file. The second argument is optional but recommended to use. Examples:

   ```
   require_once(t3lib_extMgm::extPath('lang', 'lang.php'));
   require_once(t3lib_extMgm::extPath('lang') . 'lang.php');
   ```

Always use one of these two ways to include files. This is required to include files even from the current directory. Some installations do not have the current directory in the PHP include path and `require_once` without a proper path will result in fatal PHP error.

A.2.4 Class information block

Class information block is similar to the file information block and describes the class in the file. Example:

```
/**
 * This class provides XYZ plugin implementation.
 *
 * @author John Doe <john.doe@example.com>
 * @author Jane Doe <jane.doe@example.com>
 */
```

A.2.5 PHP class

PHP class follows the Class information block. PHP code must be formatted as described in Section A.3 [PHP syntax formatting], page 12. The class name is expected to follow some conventions. The namespace and path parts are all lowercase and separated by underscores ('_'). At the end comes the "true" class name which must be written in upper camel case. Taking again the example of file '`class.t3lib_cache_backend_abstractbackend.php`', the PHP class declaration will look like:

```
class t3lib_cache_backend_AbstractBackend {
    ...
}
```

A.2.6 XCLASS declaration

The XCLASS declaration must follow the PHP class. The format of the XCLASS is very important. No spaces can be added or removed, no reformatting can be done to the declaration. If the formatting is changed, the TYPO3 Extension Manager will complain about a missing XCLASS declaration.

The XCLASS declaration must include proper path to the current class file. The following example assumes that extension key is `myext`, file name is '`class.tx_myext_pi1.php`' and file is located in the '`pi1`' subdirectory of the extension:

```
if (defined('TYPO3_MODE') && $TYPO3_CONF_VARS[TYPO3_MODE]['XCLASS']['ext/myext/pi1/class.tx_myext_pi1.php'])
    include_once($TYPO3_CONF_VARS[TYPO3_MODE]['XCLASS']['ext/myext/pi1/class.tx_myext_pi1.php']);
}
```

A.2.7 Optional module execution code

Module execution code instantiates the class and runs its method(s). Typically this code can be found in eID scripts and old Backend modules. Here is how it may look like:

```
$controller = t3lib_div::makeInstance('tx_myext_ajaxcontroller');
$controller->main();
```

This code must appear *after* the XCLASS declaration. `$SOBE` is traditional but not required name.

A.3 PHP syntax formatting

A.3.1 Identifiers

All identifiers must use camelCase and start with a lower case letter. Underscore characters are not allowed. Abbreviations should be avoided. Examples of good identifiers:

```
$goodName
$anotherGoodName
```

Examples of bad identifiers:

```
$BAD_name
$unreasonablyLongNamesAreBadToo
$noAbbrAlwd
```

Identifier names must be descriptive. However it is allowed to use traditional integer variables like `$i`, `$j`, `$k` in for loops. If such variables are used, their meaning must be absolutely clear from the context where they are used. The same rules apply to functions and class methods. Examples:

```
protected function getFeedbackForm()
public function processSubmission()
```

Class constants should be clear about what they define. Correct:

```
const USERLEVEL_MEMBER = 1;
```

Incorrect:

```
const UL_MEMBER = 1;
```

Variables on the global scope may use upper case and underscore characters. Examples:

```
$TYPO3_CONF_VARS
$TYPO3_DB
```

A.3.2 Comments

Comments in the code are highly welcome and recommended. Inline comments must precede the commented line and be indented by one tab. Example:

```
protected function processSubmission() {
        // Check if user is logged in
    if ($GLOBALS['TSFE']->fe_user->user['uid']) {
        ...
    }
}
```

Class constants and variable comments should follow PHP doc style and precede the variable. Variable type must be specified for nontrivial types and optional for trivial types. Example:

```
    /** Number of images submitted by user */
    protected $numberOfImages;
    /**
      * Local instance of tslib_cObj class
```

```
         *
         * @var tslib_cObj
         */
        protected $localCobj;
```

Single line comments are allowed when there is no type declaration for the class variable or constant. If a variable can hold values of different types, use `mixed` as type.

A.3.3 Debug output

During development it is allowed to use `debug()` or `t3lib_div::debug()` function calls to produce debug output. However all debug statements must be removed (removed, not commented!) before committing the code to the Subversion repository.

A.3.4 Curly braces

Usage of opening and closing curly braces is mandatory in all cases where they can be used according to PHP syntax (except `case` statements).

The opening curly brace is always on the same line as the preceding construction. There must be a space (not a tab!) before the opening brace. The opening brace is always followed by a new line.

The closing curly brace must start on a new line and be indented to the same level as the construct with the opening brace. Example:

```
protected function getForm() {
        if ($this->extendedForm) {
                // generate extended form here
        } else {
                // generate simple form here
        }
}
The following is not allowed:
protected function getForm()
{
        if ($this->extendedForm) { // generate extended form here
        } else {
                // generate simple form here
        }
}
```

A.3.5 Conditions

Conditions consist from `if`, `elseif` and `else` keywords. TYPO3 code must not use the `else if` construct.

```
The following is the correct layout for conditions:
        if ($this->processSubmission) {
                // Process submission here
        } elseif ($this->internalError) {
                // Handle internal error
        } else {
                // Something else here
        }
```

Here is an example of the incorrect layout:

```
        if ($this->processSubmission) {
                // Process submission here
        }
        elseif ($this->internalError) {
                // Handle internal error
        } else { // Something else here
```

It is recommended to create conditions so that shortest block goes first. For example:

```
        if (!$this->processSubmission) {
              // Generate error message, 2 lines
        } else {
              // Process submission, 30 lines
        }
```

If the condition is long, it must be split into several lines. Each condition on the line starting from the second should be indented with a two or more indents relative to the first line of the condition:

```
        if ($this->getSomeCodition($this->getSomeValiarble()) &&
                    $this->getAnotherCondition()) {
              // Code follows here
        }
```

Ternary conditional operator must be used only if it has two outcomes. Example:

```
$result = ($useComma ? ',' : '.');
```

Wrong usage of ternary conditional operator:

```
$result = ($useComma ? ',' : $useDot ? '.' : ';');
```

Assignment in conditions should be avoided. However if it makes sense to do assignment in condition, it should be surrounded by the extra pair of brackets. Example:

```
        if (($fields = $GLOBALS['TYPO3_DB']->sql_fetch_result($res))) {
              // Do something
        }
```

The following is allowed but not recommended:

```
        if (false !== ($fields = $GLOBALS['TYPO3_DB']->sql_fetch_result($res))) {
              // Do something
        }
```

The following is not allowed (missing the extra pair of brackets):

```
        while ($fields = $GLOBALS['TYPO3_DB']->sql_fetch_result($res)) {
              // Do something
        }
```

A.3.6 Switch

`case` statements are indented with a single indent (tab) inside the `switch` statement. The code inside the `case` statements is further indented with a single indent. The `break` statement is aligned with the code. Only one `break` statement is allowed per `case`.

The `default` statement must be the last in the `switch` and must not have a `break` statement.

If one `case` block has to pass control into another `case` block without having a `break`, there must be a comment about it in the code.

Examples:

```
switch ($useType) {
      case 'extended':
            $content .= $this->extendedUse();
            // Fall through
      case 'basic':
            $content .= $this->basicUse();
            break;
      default:
            $content .= $this->errorUse();
}
```

A.3.7 Loops

The following loops can be used:

- `do`
- `while`

- `for`
- `foreach`

The use of each is not allowed in loops.

`for` loops must contain only variables inside (no function calls). The following is correct:

```
$size = count($dataArray);
for ($element = 0; $element < $size; $element++) {
      // Process element here
}
```

The following is not allowed:

```
for ($element = 0; $element < count($dataArray); $element++) {
      // Process element here
}
```

`do` and `while` loops must use extra brackets if assignment happens in the loop:

```
while (($fields = $GLOBALS['TYPO3_DB']->sql_fetch_result($res))) {
      // Do something
}
```

A.3.8 Strings

All strings must use single quotes. Double quotes are allowed only to create the new line character ("`\n`"). String concatenation operator must be surrounded by spaces. Example:

```
$content = 'Hello ' . 'world!';
```

However the space after the concatenation operator must not be present if the operator is the last construction on the line. See Section A.1.4 [Whitespace and indentation], page 9, for more information.

Variables must not be embedded into strings. Correct:

```
$content = 'Hello ' . $userName;
```

Incorrect:

```
$content = Hello $userName;
```

Multiline string concatenations are allowed. Line concatenation operator must be at the end of the line. Lines starting from the second must be indented relative to the first line. It is recommended to indent lines one level from the start of the string on the first level:

```
$content = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. ' .
                  'Donec varius libero non nisi. Proin eros.';
```

A.3.9 Booleans

Booleans must use PHP's language constructs and not explicit integer values like 0 or 1. Furthermore they should be written in uppercase, i.e. `TRUE` and `FALSE`.

A.3.10 PHP5 features

The use of PHP5 features is strongly recommended for extensions and mandatory for the TYPO3 core versions 4.2 or greater.

Class functions must have access type specifier: `public`, `protected` or `private`. Notice that private may prevent XCLASSing of the class. Therefore private can be used only if it is absolutely necessary.

Class variables must use access specifier instead of `var` keyword.

Type hinting must be used when function expects array or an instance of a certain class. Example:

```
protected function executeAction(tx_myext_action& $action, array $extraParameters) {
      // Do something
}
```

Static functions must use `static` keyword. This keyword must be the first keyword in the function definition:

```
static public function executeAction(tx_myext_action& $action, array $extraParameters) {
      // Do something
}
```

`abstract` keyword also must be on the first position in the function declaration:

```
abstract protected function render();
```

A.3.11 Global variables

Use of `global` is not recommended. Always use `$GLOBALS['variable']`.

A.3.12 Functions

If a function returns a value, it must always return it. The following is not allowed:

```
function extendedUse($enabled) {
       if ($enabled) {
              return 'Extended use';
       }
}
```

The following is the correct behavior:

```
function extendedUse($enabled) {
       $content = '';
       if ($enabled) {
              $content = 'Extended use';
       }
       return $content;
}
```

In general there should be a single return statement in the function (see the preceding example). However a function can return during parameter validation before it starts its main logic. Example:

```
function extendedUse($enabled, tx_myext_useparameters $useParameters) {
       // Validation
       if (count($useParameters->urlParts) < 5) {
              return 'Parameter validation failed';
       }
       // Main functionality
       $content = '';
       if ($enabled) {
              $content = 'Extended use';
       } else {
              $content = 'Only basic use is available to you!';
       }
       return $content;
}
```

Functions should not be long. "Long" is not defined in terms of lines. General rule is that function should fit into 2/3 of the screen. This rule allows small changes in the function without splitting the function further.

A.4 Using phpDoc

phpDoc is used for documenting source code. Typically TYPO3 code uses the following phpDoc keywords:

- @author
- @access
- @global
- @param

- @package
- @return
- @see
- @subpackage
- @var

For more information on phpDoc see the phpDoc web site at http://www.phpdoc.org/ TYPO3 requires that each class, function and method is documented with phpDoc. For information on phpDoc with use in classes see Section A.2.4 [Class information block], page 11. Note that the @author tag should **not** be used in function or method phpDoc comment blocks — only at class level — because it is too liable to change frequently and authors would accumulate indefinitely. `svn blame` is enough for tracking changes. Functions should have parameters and return type documented. Example:

```
/**
  * Initializes the plugin. Checks the configuration and substitutes defaults for missing values.
  * @param   array $conf Plugin configuration from TypoScript
  * @return void
  * @see     tx_myext_class:anotherFunc()
  */
protected function initialize(array $conf) {
       // Do something
}
```

Notice the use of `void` when function does not return a value.

A.5 The ChangeLog file

TYPO3 core has a '`ChangeLog`' file where all changes are recorded. This file is committed to the SVN together with the change.

Each entry in the ChangeLog file has the following format:

- Information about the date of the change (YYYY-mm-dd), author name and author's e-mail address. These three pieces of information are separated strictly by two spaces (no tabs!)
- Empty line
- One or more lines describing the change. Each such line is indented by a single tab
- Empty line
- Line that describes the change consists from the following parts:
- Asterisk character
- Type of the change ("Fixed bug" or "Added feature")
- Bug tracker issue id ("#12345")
- A colon
- Bug tracker issue title or, if needed, a more meaningful title that describes the change ("great feature for TYPO3")
- Optional thanks to message of the patch was submitted by the contributor

In some cases the line can have free format. This applies to nonfunctional changes, like "Formatting change" or "Fixed copyright dates").

Here is an example of the ChangeLog entry:

```
2009-01-06   John Doe   <john@doe.com>
      * Fixed bug #12345: typolink does not take XYZ into account
      * Added feature #12345: Google sitemap in the TYPO3 core (thanks to Jane Doe)
```

`ChangeLog` file updates are mandatory for every change in the TYPO3 core. Extension authors are strongly recommended to use such file with the same format in their extensions.

Key Index

Variable Index

Function Index