

TYPO3 Coding Guidelines

Document key: **doc_cgl**

Copyright © 2008, Dmitry Dulepov, <dmitry@typo3.org>

This document is published under the Open Content License
available from <http://www.opencontent.org/opl.shtml>

The content of this document is related to TYPO3
- a GNU/GPL CMS/Framework available from www.typo3.org

Table of Contents

TYPO3 Coding Guidelines.....	1	General requirements to PHP files.....	7
About this document.....	3	File structure.....	7
Conventions used in this document.....	3	PHP syntax formatting.....	9
File system conventions.....	4	Using phpDoc.....	12
TYPO3 directory structure.....	4	Coding: best practices.....	14
TYPO3 files and user files.....	4	Accessing the database.....	14
Extension directory structure.....	4	Setting up your editor.....	15
File names.....	5	Eclipse.....	15
Namespaces.....	5	Komodo IDE.....	15
PHP file formatting.....	7	ChangeLog.....	16

About this document

This document defines coding guidelines for the TYPO3 project. Following these guidelines is mandatory for TYPO3 core developers. No change to core can be made if the change does not follow the guidelines.

Extension authors are strongly encouraged to follow these guidelines in their development. Following common guidelines makes it easier to read the code, analyze it for learning or make code reviews. These guidelines also help to prefer typical errors in TYPO3 code.

The original document was written by Kasper Skårhøj. This document is written from scratch and based on the discussion between developers held at TYPO3 Developer Days in May 2008.

The information for this document was collected by Ingo Renner (TYPO3 core team member, release manager for TYPO3 4.2). The initial draft was produced by Dmitry Dulepov (TYPO3 core team member).

This document defines how TYPO3 code, files and directories should look like. It does not teach how to program for TYPO3 and does not provide technical information about TYPO3.

Conventions used in this document

Monospace font is used for:

- File names and directories. Directories have slash (/) appended to the directory name.
- Code examples
- TYPO3 module names
- Extension keys
- TYPO3 namespaces (see page 5)

File system conventions

TYPO3 has certain conventions about file names for TYPO3 core and extensions. Some of them are historical and do not follow other formal rules. They will be described separately. New Core classes and extensions are required to follow formal rules outlined below.

TYPO3 directory structure

Each TYPO3 installation consists from the following directories:

Directory	Description
fileadmin/	This is a directory where users can store files. Typically images or HTML files go here. Often this directory is used for downloadable files. This directory is the only one accessible using TYPO3 Web>File module.
t3lib/	TYPO3 library directory
typo3/	TYPO3 Backend directory
typo3conf/	TYPO3 configuration directory
typo3conf/ext/	Directory for TYPO3 extensions
typo3temp/	Directory for TYPO3 temporary files. It contains subdirectories for temporary files of extensions and TYPO3 components.
uploads/	Default upload directory. All images uploaded with "Image with text" content element will be in this directory

This structure is default for TYPO3 installation. Other non-TYPO3 applications can add their own directories.

TYPO3 files and user files

All files in the TYPO3 web site directory hierarchy are divided to TYPO3 files and user files. TYPO3 files are files that come with official TYPO3 source package released by TYPO3 core team. This includes files inside `t3lib/` and `typo3/` directories and file named `index.php` in the root of TYPO3 installation

All other files are user files. This includes extensions, files in `fileadmin/` or files generated by TYPO3 (like thumbnails or temporary CSS files).

Extension directory structure

Extension directory contains the following files and directories:

Name	Description
<code>ext_emconf.php</code>	This is the only mandatory file in the extension. It describes extension to the rest of TYPO3.
<code>ext_icon.php</code>	This is extension icon. The name may not be changed.
<code>ext_localconf.php</code>	This file contains hook definitions and plugin configuration. The name may not be changed.
<code>ext_tables.php</code>	This file contains table declarations. For more information about table declarations and definitions see "TYPO3 Core API document". The name may not be changed.
<code>ext_tables.sql</code>	This file contains definitions for extension tables. It has special syntax and may not be valid SQL files. The name may not be changed.
<code>tca.php</code>	This file contains full table definitions for extension tables.
<code>locallang*.xml</code>	These files contain localizable labels. They can also appear in subdirectories.
<code>doc/</code>	This directory contains extension manual. The name may not be changed.
<code>doc/manual.sxw</code>	This file contains extension manual in OpenOffice 1.0 format. The name or file format may not be changed. See "Documentation template" document on the typo3.org for more information about extension manuals.
<code>piX/</code>	These directories commonly contain Frontend plugins. <i>x</i> is usually substituted with a number.
<code>modX/</code>	These directories commonly contain Backend modules. <i>x</i> is usually substituted with a number.
<code>modfuncX/</code>	These directories commonly contain Backend submodules (embedded into other modules). <i>x</i> is usually substituted with a number.

Name	Description
lib/	Directory for non-TYPO3 files supplied with extension. TYPO3 is licenses under GPL version or 2 or any later version. Any non-TYPO3 code must be compatible with GPL version 2 or any later version. Note: this name is not mandatory but recommended.

This directory structure is strongly *recommended*. *Extensions may create their own directories (for example, move all language files into other directories).*

File names

TYPO3 requires all PHP class files to start with `class.` prefix followed by namespace prefix, class name, underscore character and extension. For information on namespaces and namespace prefix, see the next section of this document. Extension for PHP files is always `.php`.

Non-class files must not start with `class.` prefix. It is recommended to use only PHP classes and avoid non-class files.

File names must be all lower case.

Namespaces

TYPO3 logically separates all files and directories into several namespaces. These namespaces server two purposes:

1. They show where file or directory belongs inside TYPO3 CMS
2. They restrict PHP execution only to files from a certain namespace.

There are several namespaces reserved exclusively to TYPO3. Others are for user code (extensions).

t3lib

`t3lib` namespace is reserved for common TYPO3 files. These files are used by both Frontend and Backend. Physically this namespace corresponds to `t3lib/` directory in TYPO3 directory hierarchy.

All PHP class files in `t3lib` name space start with `class.t3lib_` prefix.

User files are not allowed inside this namespace.

typo3

This namespace is reserved for TYPO3 Backend files. No user files are allowed here.

Historically files in this namespace have different prefixes and do not follow common naming rules.

tslib

`tslib` historically stands for "TypoScript library" (see original coding guidelines for details). This namespace is part of `cms` extension. Physically it is located in `typo3/sysextd/cms/tslib/` directory and contains Frontend page and content generation files.

File in this namespace historically do not follow common naming conventions.

User files are not allowed in this namespace.

tx_

This namespace is reserved for extensions. Extension PHP class files must start with `class.tx_` prefix, followed by extension key without underscores, another underscore and class name in lower case. File name ends with `.php` extension. For example, if extension key is `test_ext`, file name can be `class.tx_testext_myclass.php` and file name in the class will be `tx_testext_myClass`.

User files from this namespace commonly found in `typo3conf/ext/` directory.

user_

This namespace is reserved for PHP files without PHP classes. Non-class files can be called by TYPO3 only if they have `user_` prefix (the prefix can be changed by administrator in Install tool). All functions inside such files must have `user_` prefix as well.

Files from this namespace typically placed inside `fileadmin/` directory or its subdirectory.

UX_

This names space is reserved for `XCLASS` files. These files are usually included into extensions.

PHP file formatting

General requirements to PHP files

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 closing tag. Empty lines after closing tags may break output compression in PHP and produce AJAX errors.

Line breaks

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

Line length

Line 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 indents. Example:

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

Whitespace and indents

TYPO3 uses tabs for indenting source code. One indenting level is one tab.

There must be no white spaces in the end of the line. Configure your editor to remove such white spaces (see section "Setting up your editor" on page 15) or do it manually. If you use terminal, you can use `sed` to strip spaces from the end of the line:

```
sed 's/[ \t]*$//g' filename
```

Character set

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

XML language files use UTF-8 character set.

File structure

TYPO3 file structure is the following:

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
9. Closing PHP tag

Next sections in this chapter discuss each of these parts.

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) 2008 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.

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

In the example above there is also SVN \$Id\$ meta keyword. SVN will expand it to the version string when file is committed.

File information block can also contain optional function index. This index is created and updated but `extdeveval` extension.

Included files

Files are usually included with `require_once` function. There are two ways to obtain path for included file:

1. Use one of predefined TYPO3 constants: `PATH_tslib`, `PATH_t3lib`, `PATH_typo3`, `PATH_site`. The first two contain paths to corresponding TYPO3 directories. The last contains 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 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 current directory in the PHP include path and `require_once` without a proper path will result in PHP fatal error.

Class information block

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

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

PHP class

PHP class follows the Class information block. PHP code must be formatted as described in chapter “PHP syntax formatting” on page 9.

XCLASS declaration

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

XCLASS declaration must include proper path to the current class file. Example:

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

Here extension has `myext` extension key. The file is located in `pil/class.tx_myext_pil.php` file.

Optional module execution code

Module execution code exists only for Backend modules. Here is how it typically looks like:

```
$SOBE = t3lib_div::makeInstance('tx_myext_module1');
$SOBE->init();
$SOBE->main();
$SOBE->printContent();
```

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

PHP syntax formatting

Identifiers

All identifiers must use `camelCase`. Underscore characters are not allowed. Abbreviations should be avoided. Examples of good identifiers:

```
$goodName
$anotherGoodName
```

Examples of bad identifiers:

```
$BAD_name
$unreasobalyLongNamesAreBadToo
$noAbbrAlwd
```

Identifier names must be descriptive. However it is allowed to use traditional integer variables like `$i`, `$j`, `$k` for `for` loops. If such variables are used, their meaning must be absolutely clear from context.

Same rules apply to functions. 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;
```

Comments

Comments in the code are highly welcome and recommended. Inline comments must precede the commented line and be indented on the same level. Example:

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

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

```
/** Number of images submitted by user */
protected $numberOfImages;
/**
 * Local instance of t3lib_cObj class
 *
 * @var t3lib_cObj
 */
```

```
protected $localCobj;
```

Single line comments are allowed when there is no type declaration for the variable or constant.

If variable can hold value of different types, use `mixed` as type.

Debug output

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

Braces

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

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

Closing brace must start on a new line and be indented to the same level as the construct with 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
    }
}
```

Conditions

Conditions consist from `if`, `elseif` and `else` keywords. 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 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
}
```

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

```
$result = ($extendedUser ? $this->extendedUse() : $this->basicUse());
```

Wrong usage of ternary conditional operator:

```
$result = ($extendedUser ? $this->extendedUse() : $basicUse ? $this->basicUse() : $this->errorUse());
```

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

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

The following is allowed for clarity but not recommended:

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

The following is not allowed (missing brackets):

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

Switch

case statements are indented with a single indent inside switch statement. Code inside case statements is further indented with a single indent. break statement is aligned with code. Only one break statement is allowed per case.

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

If one case has to pass control inside another case 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();
}
```

Loops

The following loops can be used:

- do
- while
- for
- foreach

The use of each is not allowed.

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
}
```

Strings

All strings must use single quotes. Double quotes are allowed only for “\n”.

String concatenation operator must have spaces around it. Example:

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

However space after the concatenation operator must not be present if the operator is the last construction on the line. See section about white spaces on page 7 for more information.

Variables must not be embedded to strings. Example:

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

PHP5 features

The use of PHP5 features is strongly recommended for extensions and mandatory for TYPO3 core version 4.2 or newer.

Class functions must have access type specifier: `public`, `protected` or `private`. Notice that `private` may prevent XCLASSing of the class. Therefore use `private` 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 the certain class. Example:

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

Globals

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

Functions

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

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

Correct:

```
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 function can return during parameter validation before it starts its main logic. Example:

```
function extendedUse($enabled, tx_myext_useparams $useParameters) {
    // Validation
    if (count($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 must not be long. Long is not defined in terms of lines. General rule is that function should fit into 2/3 of the screen. 2/3'rds rule allow small changes in the function without splitting the function further.

Using phpDoc

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

- @author
- @access
- @coauthor
- @global
- @param
- @return
- @see
- @var

For more information on phpDoc see the phpDoc web site at <http://www.phpdoc.org/>

TYPO3 requires that each class and each function is documented with phpDoc. For information on phpDoc with use in classes see "Class information block" on page 8.

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 the value.

Coding: best practices

This section documents best practices when creating code for TYPO3.

Accessing the database

TYPO3 database should be always accessed through the use of `$GLOBALS['TYPO3_DB']`. This is the instance of `t3lib_db` class from `t3lib/class.t3lib_db.php`.

The same rule applies for accessing non-TYPO3 databases: they should be accessed through the same class. Failing this condition may corrupt TYPO3 database or prevent access to TYPO3 database for the rest of the script.

Setting up your editor

Eclipse

Open Eclipse preferences. In the "General", "Editors", "Text editors":

- clear "Insert spaces for tabs"
- check "Show print margin" and set "Print margin column" to 80

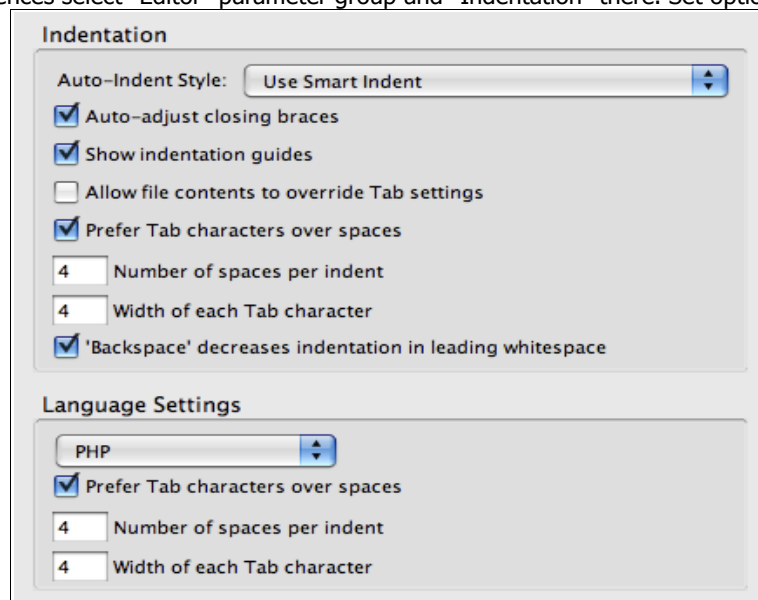
To clear trailing whitespace on save, install "AnyEdit" Eclipse plugin (search with Google). Then in "General", "Editors", "AnyEdit tools" check two options: "Remove trailing whitespace" and "Convert tabs <-> spaces". Also select "Spaces to tabs".

In the "General", "Workspace" set "New text file line delimiter" to "Other" and "Unix".

Komodo IDE

Komodo IDE is a product of ActiveState. It is available at <http://www.activestate.com/>

In the Komodo IDE preferences select "Editor" parameter group and "Indentation" there. Set options as follows:



In the "Smart editing" group set "Show edge line / Highlight characters beyond edge line" and enter "80" in the "Edge line column".

In the "Save options" group set "Clean trailing whitespace and EOL marker".

In the "New files" group set "Specify end-of-line (EOL) indicator for newly created files" to "Unix (\n)".

ChangeLog

Date	Modified by	Description
2008-08-03	Dmitry Dulepov	Initial draft