WoltLab Community Framework

Technical Documentation

WoltLab GmbH
http://www.woltlab.com

Contents

Part I.

The WoltLab Community Framework

1. Introduction

1.1. About the WoltLab Community Framework

The foundation of the WoltLab Burning Board 3 (WBB) is created by the WoltLab Community Framework (WCF). It will be the basis for all future WoltLab products in the sphere of Community development. It is completely object–oriented, programmed in PHP 5 and provides support for module–style packages. Burning Board 3 is the first application to use this system. The new forum software consists of multiple smaller packages, which can be shared with other applications.

The Community Framework also provides functions for the automatic update of installed packages. For this, the manufacturer's (e.g. WoltLab's) package database is checked for updated versions of these packages. If required, the updates are automatically downloaded and executed. It is also possible to manually search the package database, in order to install add-ons or completely new applications. A selected package and all other packages required additionally will be automatically downloaded and installed. Unless it is being used by another package, an installed package can be removed with a single mouse click.

1.2. Terminology

In this documentation, the following terms will be used:

Package a module for the WCF

Style the visual appearance of the standalone application (comparable with "skin" or "theme"). A style consists of value definitions of colors, fonts and sizes (CSS) as well as the matching templates.

Template XHTML–pattern with placeholders (a.k.a. template variables).

1.3. License

The WoltLab Community Framework as such is covered by the "GNU Lesser General Public License" (LGPL). The complete license terms can be found at the following address:

http://www.gnu.org/licenses/lgpl.html

Put simply, you can use all WCF packages that are covered by the LGPL for your own free or fee-based programs.

Some packages are covered by the Burning Board License. These can only be used if you have purchased a respective license from WoltLab for every installation of your own application. Alternatively, individual license agreements are also negotiable.

2. Installation

2.1. System requirements

The following requirements must be fulfilled for the successful installation of the WoltLab Community Framework:

- A webserver with PHP 5 support
- Apache 2 with PHP 5.2.x-Module recommended
- A MySQL-Database version 4.1.2 or higher
- Approx. 6 MB hard-disc storage space
- An FTP-Program in order to load the program files onto the Webserver/Webspace

2.2. Download

Please begin by using the WCF which is included in the download file of the forum software WoltLab Burning Board. We are still working on a separate download of the WCF.

2.3. Performing the installation

For the installation of the WCF you need two files that can be found in the downloaded zip-archives:

install.php The installation script, which can be conveniently executed via the browser.

WCFSetup.tar.gz Contains all files and packages of the Community Framework.

Please load both of these files onto your webserver and execute the installation file `install.php`. The minimum steps to be completed are as follows:

1. Choosing the language of the installation assistant
2. Accepting the license agreement

3. Checking the system requirements
4. Selecting the installation directory of the WCF
5. Selecting character encoding and language
6. Entering the database access data
7. Creating an administrator account
8. Optionally installing other standalone applications

3. The package system

3.1. The fundamentals

The elementary constituent of the WCF is the package system. Packages are single components that encapsulate a certain functionality and that can be used by other packages. Through the WCF, a multiplicity of packages can be installed, automatically updated or managed.

A package is an archive of files. The most important file is the `package.xml`. This includes all important details of the package such as its name, which other packages it depends upon and which tasks need to be completed during the installation of the package.

1. **simple package** Within a simple package, functions are merely provided without being actively used. A package can deliver PHP classes, own database tables, language files, graphics and templates. Packages can depend upon other packages.
2. **standalone application** The standalone application package uses the functions of the other (simple) packages and provides its own interfaces. Only standalone applications have their own graphic user interface.
3. **plug-in** A plug-in is an extension of a standalone application containing new functions. Interfaces that have previously been defined by other packages are used for that purpose. This means the plug-in is optional, it is not a prerequisite of a package.

3.2. Included basic packages

The Community Framework provides the most essential basic functions of web applications. Further features are included as free basic packages. The following systems are permanently integrated into the WCF:

Package administration This package systems manages the installed packages, installs, updates and uninstalls packages.

Database connection The WCF provides a so-called "Database Abstraction Layer" that facilitates the use of a MySQL database.

User management and authentication The user management creates, edits and removes user accounts. User can authenticate themselves (e.g. to LDAP[1]).

Group management The group management carries out the classification of users into groups.

Sessions The WCF manages sessions for you.

Language administration Allows multilingual user interfaces.

Caching The caching system allows the intermediate storage of data that needs to be rapidly accessible in the data system.

Events The event system permits the execution of functions at previously defined events.

Template system The WCF provides an own template system for the distribution of the sites.

com.woltlab.wcf

The packages as described above are provided through the installation of the WCF. Additionally, all important Package Installation Plugins (see chapter 13 on page 66) are also included.

com.woltlab.wcf.data.cronjobs

Cronjobs are time-controlled tasks known from the world of Unix. Real cronjobs are not available most of the time as they require their own server. That is why the functionality of Cronjobs is imitated by AJAX and PHP. Through this simulation, regular tasks can still be executed. These tasks could include the refreshment of readouts. Often, certain readouts are buffered (caching), because they are frequently required by users to relieve the database. With the help of a Cronjob, a buffer like that can for instance be updated once a day.

com.woltlab.wcf.data.feed.reader

Information on websites is nowadays often proviced through an RSS-feed. This plugin permits the regular import of this information, for the purpose of displaying it within your own application.

[1] http://en.wikipedia.org/wiki/Ldap

com.woltlab.wcf.data.help

This package delivers the simple facility of implementing an end user help for your own application. It is designed in a way so that your own help topics can be defined through an XML-file. A search function is available to the end user and in addition a referrer[2] can be defined for every topic. If a user is on a site and accesses the help function, he can be referred directly to the corresponding help page.

com.woltlab.wcf.data.image

Within this module help functions that can be used for the processing of images are provided. This includes for instance the provision of a `thumbnail`-class to create small preview images.

com.woltlab.wcf.data.image.captcha

Captcha-images are an important instrument to preserve the safety of web applications. Whenever data can be entered into forms by unregistered users, this creates points of attack for automated spam robots. This can be precluded by using these security images. The drawback of this solution is that these pictures are no barrier-free and blind or visually impaired people cannot decipher the letters and numbers.

com.woltlab.wcf.data.message

The message package is to be used when a typical message form is available. In general, this would consist of a subject and a text.

com.woltlab.wcf.data.message.attachment

When composing messages, other media are often embedded as attachment files. This package facilitates this.

com.woltlab.wcf.data.message.bbcode

When composing messages on websites, these can be formatted with the help of BB-Codes. They are a kind of HTML substitute, because the direct use of HTML is often not recommendable due to safety reasons.

[2]The Referrer is the Internet address, the current page was called on this one.

com.woltlab.wcf.data.message.censorship

In order to censor certains words in a message, this module is required. For this, a list of words that are prohibited must be compiled. On sending, the message is then checked for the words on the list and if applicable there will be an error message.

com.woltlab.wcf.data.message.poll

This package can be used to compose a message containing a poll.

com.woltlab.wcf.data.message.search

The search function is the central component of every system that stores messages. The concrete implementation of this search uses the MySQL-Fulltext-Index.

com.woltlab.wcf.data.page

The package contains templates for the outer frame of a page. In addition, it also brings the PageLocation-PIP (see chapter 13.2.11 on page 82).

com.woltlab.wcf.data.page.headerMenu

This module provides the functionality needed to display a main menu on the page. Through an XML-file, any number of entries can be defined. During the installation these are inserted into the database.

com.woltlab.wcf.form.message

This package contains all the functionalities required to compose or edit a message.

com.woltlab.wcf.form.message.wysiwyg

The WYSIWYG-Editor is offered for the composition of messages. This enhances usability because the user can see directly how formatting affects the text when creating a message, with no knowledge of the corresponding codes required.

com.woltlab.wcf.form.user

All user actions concerned with the user account, such as registering, signing in, changing e-mail address or requesting a lost password are covered by this package.

com.woltlab.wcf.page.user.profile

With the help of this component, the users' public profiles can be displayed.

com.woltlab.wcf.system.style

The style package already offers the corresponding formatting for many of the application's user elements. It provides CSS classes that amongst others govern the displays of Tab-Menus, tabular lists or buttons.

com.woltlab.wcf.system.template.pack

Use this package if different template groups are to be supported within the application. That way, the community users can choose between several styles based on different templates.

com.woltlab.wcf.system.template.plugin.includePHP

This plugin facilitates the use of PHP-Code within the templates.

4. Quick start

To receive fast results, this chapter provides a short overview of the `WCF` class and the object-oriented approach of the WoltLab Community Framework.

4.1. The class `WCF`

The class `WCF` (in the file `wcf/lib/system/WCF.class.php`) is the central class of WoltLab Community Framework. It facilitates amongst others access to the database, the template or language system.

The WCF is programmed object-orientated, so the database, template system, etc. have their own objects (class entities). The WCF uses objects according to the so-called *Singleton Pattern*[1], which ensures that exactly one instance of these classes exists throughout the whole application. Access is carried out through the `WCF` class, in which the entities are saved as static member variables.

4.1.1. Database access

With `WCF::getDB()` the instance of the database object is obtained, so that queries to the database server can be sent and processed, e.g.

```
// send the SQL query $sql to server
$result = WCF::getDB()->sendQuery($sql);

// receive result and scan every row
while ($row = WCF::getDB()->fetchArray($result)) {
  // ..
}
```

Further information on the use of the database system can be found in chapter 5 on page 21.

[1] http://en.wikipedia.org/wiki/Singleton_pattern

4.1.2. Template system

With `WCF::getTPL()` you obtain the instance of the template system.

```
// Assignment of variables
WCF::getTPL()->assign('action', $action);

// Display the template $templateName
WCF::getTPL()->display($templateName);
```

You will find more detailed information in chapter 6 on page 25.

4.1.3. Signed-in user

`WCF::getUser()` delivers the object which represents the current visitor of the site. Normally, this is an instance of the `UserSession` class or a class inherited from this (e.g. in the standalone application WoltLab Burning Board these can be the inherited classes `WBBUserSession` or `WBBGuestSession`).

The `UserSession` class is in turn inherited from the class `User`.

```
WCF::getUser()->userID
WCF::getUser()->username
```

4.1.4. Session

`WCF::getSession()` delivers the instance of the class `Session` (or a class inherited from this), e.g. for the registration or the readout of session variables.

```
// get all Session Variables
$sessionVars = WCF::getSession()->getVars();

// get only the Session Variable 'test'
$test = WCF::getSession()->getVar('test');

// register the Variable 'blub'
WCF::getSession()->register('blub', $blub);
```

In chapter 9 on page 53, you will learn more about this topic.

4.1.5. Language system

`WCF::getLanguage()` delivers the instance of the class `Language`, e.g. to load the name of the language variable:

```
WCF::getLanguage()->get("name.der.variablen");
```

4.1.6. Cache

The cache is used to buffer frequently used data in the file system. `WCF::getCache()` delivers the instance of the class `CacheHandler`, which manages the cache. Find out more in chapter 10 on page 55.

```
// Read data
$data = WCF::getCache()->get('spiders');
```

4.1.7. Request

`WCF::getRequest()` delivers the instance of the class `RequestHandler`. The RequestHandler can be used to establish which page has been access. See chapter 11 on page 57.

4.2. The class `DatabaseObject`

All classes representing a dataset are inherited from the abstract class `DatabaseObject` (also referred to as `data`-classes from hereon). The data that is to be managed is buffered in the class variable `$data`. In practice, `$data` is often an associative array.

The arguably most important and most useful feature of the `DatabaseObject`-class is the use of the magic method `__get()` of PHP5[2].

This allows access to the variable `$dataObject->myVar` through the notation `$dataObject->data[` – provided that `$dataObject->myVar` does not exist as a class variable. Only this magic method allows accesses such as `WCF::getUser()->username`.

Normally, for every `Data`-Class you will be able to find a corresponding `DataEditor`-class. The `DataEditor`-Class frequently contains static methods such as `create()`, `delete()` and `insert()`. While the `Data`-Classes ensure that the data is read out, the `DataEditor`-classes are employed to create new datasets or to edit or delete existing ones.

[2] see http://www.php.net/manual/en/language.oop5.magic.php
and http://www.php.net/manual/en/language.oop5.overloading.php

5. Database interface

Although the WoltLab Community Framework provides the rudiments of a database abstraction layer, the current version [1] so far only contains implemented support for the database system MySQL. Implements for other database systems have to be inherited from the abstract class `Database`.

5.1. Methods of the `Database` class

sendQuery

resource **sendQuery** (*string* $query)

The method `sendQuery()` sends an SQL-query to the database server and returns the result identifier. If the SQL query fails, a `DatabaseException` is thrown.

Note: The SQL query should not close with a semicolon.

sendUnbufferedQuery

resource **sendUnbufferedQuery** (*string* $query)

`sendUnbufferedQuery()` sends an SQL query to the database server without immediately getting the datasets of the result[2].

Note: `countRows()` and `getAffectedRows()` cannot be applied to result identifiers of `sendUnbufferedQuery()`.

[1] State: WCF Version 1.0.1
[2] siehe auch http://www.php.net/manual/en/function.mysql-unbuffered-query.php

fetchArray

array **fetchArray** ([*resource* $queryID = null], [*int* $type = null])

`fetchArray()` delivers a dataset of an SQL-query in the form of an array. If the parameter `$queryID` has not been entered, the SQL-query last sent is assumed. The second optional parameter `$type` defines the type of the array. The standard value `Database::SQL_ASSOC` delivers an associative array, `Database::SQL_NUM` a numerical array and `Database::SQL_BOTH` delivers both.

getFirstRow

array **getFirstRow** (*string* $query, [*integer* $type = null])

`getFirstRow()` is used when only the first dataset of an SQL-query is needed. In this case, `getFirstRow()` is an abbreviated notation for the execution of `sendQuery()` and `fetchArray()`.

Like with `fetchArray()` the optional parameter `$type` defines the type of the array.

getResultList

array **getResultList** (*string* $sql)

`getResultList()` sends an SQL-query, collects the datasets and returns them in a multidimensional array. `getResultList()` is convenient for reading out all the datasets of an SQL-query if you do not wish to work in the script subsequently.

countRows

integer **countRows** ([*resource* $queryID = null])

`countRows()` delivers the number of datasets in result of an SQL-query. `CountRows()` can only be applied to SELECT-queries. If you want to determine the number of manipulated datasets from an UPDATE-, INSERT- or DELETE-query, please use `getAffectedRows()`.

If no `$queryID` is specified, the SQL-query last sent is assumed.

getAffectedRows

integer **getAffectedRows** ()

`getAffectedRows()` returns the number of datasets manipulated by the most recently executed INSERT-, DELETE or UPDATE-query.

getInsertID

int **getInsertID** ()

`getInsertID()` delivers the ID that was assigned for a field of the type AUTO_INCREMENT during the last INSERT-query. `getInsertID()` delivers 0, when the preceding INSERT-query has not generated an AUTO_INCREMENT value.

If you wish to save the value for later use, make sure to access `getInsertID()` directly after the INSERT-query which created the value.

seekResult

void **seekResult** ([*integer* $queryID = null], *integer* $offset)

`seekResult()` moves the reading pointer of a quantity of results to the `$offset` position. The pointer begins at position 0, i.e. position 0 is pointing at the first set of data.

If the parameter `$queryID` has not been entered, the SQL-query last sent is assumed.

registerShutdownUpdate

void **registerShutdownUpdate** (*string* $query, [*integer* $key = null])

`registerShutdownUpdate()` inserts an UPDATE-command in a waiting list of UPDATE-commands, that are executed at the end of the script.

escapeString

string **escapeString** (*string* $string)

`escapeString()` is a wrapper for the PHP-function `mysql_real_escape_string()` and has to be used for the masking of special characters on strings if these are going to be used in an SQL-query.

Note: Instead of the call WCF `WCF::getDB()->escapeString($string)`, you can also type `escapeString($string)`.

getErrorDesc

string **getErrorDesc** ()

Delivers the error message of a previously executed database query.

getErrorNumber

int **getErrorNumber** ()

Delivers the number of an error message of a previously executed database query.

getVersion

string **getVersion** ()

Delivers the version number of the database system.

getDBType

string **getDBType** ()

Returns the database type in use.

getDatabaseName

string **getDatabaseName** ()

Returns the name of the database in use.

getCharset

string **getCharset** ()

Returns the used character set.

getTableNames

array **getTableNames** ([*mixed* $database = ''])

Returns an array with all tables existent in the database.

getTableStatus

array **getTableStatus** ()

Returns an array with detailled information on every table in the database.

6. The template system

The template system of the WoltLab Community Framework in syntax and operation mode is geared to Smarty[1], the compiling template system, but without reaching its massive functional range.

The template system reads the template files (file ending `.tpl`) and converts them directly into PHP-scripts. This means templates only need to be parsed after being changed. For the following requests, the generated PHP-script is simply executed.

In addition, the template system can be expanded by custom functions, more on this later in chapter 6.6 on page 42.

6.1. Basic syntax for template designers

All template commands are surrounded by curly brackets, e.g. `{include file="boardList"}`.

6.1.1. Variables

Before being able to use a template variable in the templates, it has to be declared in the corresponding PHP-script and a value has to be assigned to the variable.

This works with the `assign()` method of the class `Template`, which is described in more detail in 6.2.1 on page 27.

Template variables within the templates are, like PHP-variables, activated through the dollar sign $. Additionally, PHP-constants can be accessed as well. Program 6.1 on the next page on the following page gives a few examples.

[1] http://smarty.php.net

@ and

When putting out variables, HTML special characters are masked automatically by the template system. If this is not desired in certain cases, this can be disabled with the @-sign: `{@$variable}`.

Warning: Do not use @ unless you would like the HTML special characters to be displayed as such.

For the output of numerical values, the pound key # is very useful. The pound key effects the automatical formatting of the numbers, i.e. decimal numbers are rounded to two decimal places and separated by the decimal divider depending on the language. If applicable, a divider for the thousands is inserted: `{#$number}`

Programm 6.1 Access of variables within the templates

```
{$variable}
{@$variable}
{#$variable}
{CONST}
{$array.0} (for $array[0])
{$array.key} (for $array['key'])
{$array.$key} (for $array[$key])
{$object->var}
{$object->method($foo)}
{WCF::getUser()->userID}
```

6.1.2. Commentaries

Single sections in templates can be exluded from parsing through commentaries. A note is opened by `{*` and closed by `*}`.

```
{* This is a commentary *}
```

Commentaries can also be extended over several lines.

Commentaries are not sent to the client during the XHTML-output.

6.1.3. Functions

Template functions are accessed through the syntax `{name of function}`. A distinction is made between normal functions and blocking functions. Blocking functions are functions of the form `{block} ... {/block}` that surround a text, similar to XML-tags. These functions treat the surrounded text as an input.

Parameters

With help of the syntax `{name of function param1=$val1 param2="value"}`, parameters can be forwarded to the function `nameoffunction`. With blocking functions, the paramaters are inserted into the opening function request. `{block param1=$val1} ... {/block}`

6.1.4. Modifiers

Modifiers are used to apply a function to a variable. Both your own modifiers or the php-functions can be accessed. The syntax is short and simple: Add `|modifier` the variable to open the modifier, e.g. `{$var|empty}`.

Parameters

A modifier is applied to a template variable. It is however also possible to pass on further parameters to the modifier. For this, add the parameter values to the modifier name separated through colons :
`{$var|modifier:"val1":"val2":$foo}`

Program 6.2 shows some typical use cases for modifiers:

Programm 6.2 Application of modifiers to template variables

```
{@TIME_NOW|fulldate}
{$title|truncate:40:"..."}
{function param1=$val1|modifier}
{function param1=$val1|modifier:$val2:"val3"}
```

6.2. The template system for programmers

In this section, the most important methods of the class `Template` are presented.

6.2.1. assign

void **assign** (*mixed* $variable, [*mixed* $value = ”]) The `assign()`-method declares and initiates one or more template variables, so they subsequently be used in the templates. The `assign()`-methog accepts a name- or value pair or alternately an associative array with name or value pairs (see program 6.3 on the next page).

Programm 6.3 Variable allocation with `assign`

```
// ..
$variable = 'value';
$object = new myObject();
$array = array(1,2,3,4);

WCF::getTPL()->assign('variable', $variable);
WCF::getTPL()->assign(array(
 'object' => $object,
 'array' => $array
));
```

6.2.2. append

void **append** (*mixed* $variable, [*mixed* $value = ''])

The `append()`-method attaches contents to an existing template variable (concatenation). Like the `assign()`-method, the `append()`-method too accepts a name or value pair or an associative array with name or value pairs.

6.2.3. assignByRef

void **assignByRef** (*string* $variable, *mixed* &$value)

`assignByRef()` creates a template variable with reference to a PHP-variable. For the argument, the name of the template variable and of the PHP-variable are wanted.

6.2.4. clearAssign

void **clearAssign** (*mixed* $variable)

The `clearAssign()`-method deletes one or several template variables from memory. The method expects as argument the name of the template variables to be deleted or an array with the names of the template variables to be deleted.

6.2.5. clearAllAssign

void **clearAllAssign** ()

`clearAllAssign()` deletes all template variables. This method does not possess any arguments.

6.2.6. display

void **display** (*string* $templateName, [*boolean* $sendHeaders = true])

To display a template send the output to the client, the `display()`-method is used.

The method wants the name of the template. The second (boolean) argument `$sendHeaders` is optional and determines if the HTTP HEADERs of the site are to be sent[2].

The `display()`-method automatically takes care of the compiling of the template.

When the sending of the HTTP HEADERs is activated, the events `shouldDisplay` and `didDisplay` are also triggered before and after the output of the template.

6.2.7. fetch

string **fetch** (*string* $templateName)

If you wish to display a template without immediately sending the output to the client, e.g. continue processing it instead, use the `fetch()`-method to return the output of the template.

The method expects the name of the template as argument.

6.2.8. registerPrefilter

void **registerPrefilter** (*mixed* $name)

A prefilter is a program that can be applied to the template contents before the compilation of a template. To use a prefilter, it needs to be registered first by using `registerPrefilter()`.

`registerPrefilter()` expects as an argument the name of the prefilter or an array with the names of multiple prefilters.

How to create your own prefilters you will find out in 6.6.4 on page 43.

[2] see class `HeaderUtil`

6.3. Fixed integrated functions

6.3.1. `if`,`else`,`elseif` – Case differentiation

The function `{if} ... {/if}` is used for conditions in templates. This provides (almost) the same possibilities as in PHP-scripts. As in PHP, the expressions `{else}` and `{elseif}` (or alternatively `{else if}`) exist as well.

Programm 6.4 Syntax of the {if}-function

```
{if CONDITION1}
 CONDITION1 is true
{elseif CONDITION2}
 CONDITION2 is true
{elseif CONDITION3}
{else}
 none of the CONDITIONs are true
{/if}
```

The syntax of the `{if}`-function can be seen in program 6.4. Within this, `CONDITION1`, `CONDITION2` and `CONDITION3` are expressions which, besides variables and modifiers, also allow the operators `||`,`&&`,`>`, `>=`,`<`,`<=`,`==`,`===`,`!=`,`!===`[3] known from PHP.

Warning: In the WoltLab Community Framework 1.0.0., bracketed expression are not possible in CONDITIONs of the `if`-function[4].

Programm 6.5 Examples of the {if}-function

```
{if $userMessages|isset}{@$userMessages}{/if}

{if $boards|count > 0}
 ...
 ...

 {if $hasChildren}<ul id="category{@$boardID}">{else}</li>{/if}
 {if $openParents > 0}{@"</ul></li>"|str_repeat:$openParents}{/if}
{/if}

{if $this->user->userID}
 {if $this->user->activationCode && REGISTER_ACTIVATION_METHOD == 1}
  ....
 {/if}
 ...
{elseif !$this->session->spiderID}
 ...
{/if}
```

[3] see http://www.php.net/manual/en/language.operators.logical.php
and http://www.php.net/manual/en/language.operators.comparison.php
[4] http://www.woltlab.com/forum/index.php?page=Thread&threadID=122382

6.3.2. include

With the `{include}`-function, a different template can be loaded into the current template. This way, frequently used sections can for instance be outsourced into their own template and then be integrated into other templates.

The `{include}`-function expects the parameter `file`, which defines the template that is to be integrated.

Programm 6.6 Syntax of the {include}-function

```
{include file='headerMenu'}
{include file='header' sandbox=false}
{include file='template' assign='var' append=true}
```

The integrated template can also be assigned additional template variables via parameter: `{include file='header' var1='foo' var2=$foo}`.

Integrated templates are run in a "sandbox", i.e. changes to the template variables within the integrated templates do not have any effect on the variables in the current templates. These properties can be managed through an optional `sandbox`-parameter.

Through the optional parameter `assign`, the content of an integrated template can be allocated to a template variable. When passing the optional parameter `append=true`, the content is attached the to template variable designated in `assign`.

Parameter	Required	Default	Imapct
`file`	yes	–	Name of the integrated template
`assign`	no	–	Variable to which the content of the template is to be assigned
`append`	no	false	Attach content to the template variable?
`sandbox`	no	true	activate/deactivate "Sandbox"
`beliebig`	no	–	Variable passed to the integrated template.

Table 6.1.: Parameters of the {include}-function

6.3.3. foreach

The function `{foreach from=$array item='val'} ... {/foreach}` is the equivalent to the `foreach`-loop[5] `foreach($array as $val) { ... }` in PHP.

[5] http://www.php.net/manual/en/control-structures.foreach.php

The required parameter `from` specifies the array that is to be passed through in the loop. For the required parameter `item`, specify the name of the variable that the current active element of the array is assigned to.

When using the optional parameter `key`, the at any one time current key is saved in the variable specified by key.

Programm 6.7 Examples for the {foreach}-loop

```
<?php
$arr = array("one", "two", "three");
$arr2 = array(
 array("a", "b"),
 array("y", "z")
);

WCF::getTPL()->assign(array(
 'arr' => $arr,
 'arr2' => $arr2
));
?>

{* foreach-loop *}
{foreach from=$arr item=value key=key}
 Key: {$key}<br />
 Value: {$value}<br />
{/foreach}

{* multidimensionals arrays can be passed with nested foreach-loops too *}
{foreach from=$arr2 item=v1}
 {foreach from=$v1 item=v2}
  {$v2}<br />
 {/foreach}
{/foreach}
```

6.3.4. section

The `{section}`-function is used for loop-runthroughs that require the functionality of a `for`-loop from PHP. The syntax is more complex compared to `{foreach}`, but in return `{section}` offers several additional functions.

The required parameter `name` includes the name of the counter variables, for which the iteration will be used, e.g. `name=i` or `name=j` etc.

The requires parameter `loop` determines the number of iterations. `loop` can also be an array. In this case, the number of elements is the number of iterations. You however can also pass a whole number.

With the parameters `start`, `step` and `max`, the iteration can be manipulated. Normally, the counter variable defined by `name` will begin at 0. With the help of `start`, a different starting value can be forced.

Parameter	Required	Default	Impact
`loop`	yes	–	Determines the number of iterations, can be a whole number of an array.
`name`	yes	–	Name of the counter variable.
`start`	no	0	Starting position of the counter variable (whole number).
`step`	no	1	Increment of the iteration (whole number).
`max`	no	–	Maximum number of iterations (natural number).
`show`	no	true	show or hide output of the `{section}`.

Table 6.2.: parameters of the `{section}`-function

`step` prescribes the increment of the iteration. The standard value is 1. In that way `start=1` and `step=2` will iterate over the set {1, 3, 5, 7, ...}.

`max` limits the number of iterations to a certain value.

Through `show=false` the loop run through can be deactivated.

Note: When nesting `{section}`-loop make sure to choose a custom, unique `name`-parameter for each loop.

Table 6.2 offers an overview of the parameters and Program 6.8 offers an example.

Programm 6.8 Example for the use of the {section}-loop

```
{* displays 0,2,4,6,8 *}
{section name=i loop=10 step=2}
 {$i}<br />
{section}
```

6.3.5. capture

The `{capture} ... {/capture}`-function activates the output buffering[6] and saves the output of the template codes, which is enclosed by the function, in a template variable. In the template code, variables and other template functions are analyzed as normal.

The name of the template variables is defined by the parameter `assign="var"`. The content between `{capture assign="var"}` and `{/capture}` will then be saved in the template variable `$var`.

[6] to the PHP-function `ob_start` and `ob_get_contents`

If you wish to attach the template code to the existing content of a template variable (instead of overwriting the contents of the template variable), you can use the parameter `append`: `{capture append="var"} ... {/capture}`.

6.4. Additional provided functions

6.4.1. append

The `{append}`-function allows for the `append()`-method of the current template object to be accessed in the template. Also see 6.2.2 on page 28 for this.

```
{append var=name value="foo"}
```

Technically, this is a `TemplateCompilerPlugin` - i.e. the method is already called during the compiling of the template.

6.4.2. assign

Like with the `{append}`-function, `{assign}` accesses the method of the same name of the current template object. Also see 6.2.1 on page 27 for this.

```
{assign var=name value="foo"}
```

This is technically also a `TemplateCompilerPlugin`.

6.4.3. counter

The `{counter}`-function is used to create a series of numbers. On first call, a counter variable is initialized that is increased with every further access. Both the starting value (`start`) and the increment of the counter (`skip`) are freely selectable (default values are 1).

You can also detemine if the value of counter is to be output in the template (`print`, default ist `true`) und if the value is to be assigned to a template variable (`assign`).

Through the parameter `direction="down"`, you can make the counter count in reversed order. This has the same effect as a negative value for the `skip` parameter.

Note: If you require several counters that are independent of each other, you need to assign these unique names with help of the `name`-parameter.

Parameter	Required	Default	Impact
`name`	no	default	internal name of the counter
`start`	no	1	start value of the counter (whole number)
`skip`	no	1	increment of the counter (whole number)
`direction`	no	up	direction of the counter (“up” or “down”)
`print`	no	true	output counter
`assign`	no	–	assign counter to a template variable

Table 6.3.: Parameter of the {counter}-function

Programm 6.9 Example of the use of the {counter}-function

```
{foreach from=$boards item=child}
 ...
 {counter assign=boardNo print=false}

 ...
 {$boardNo}
 ...

{/foreach}
```

6.4.4. cycle

With `{cycle}`, you can alternate between two values in turns. This is required frequently, e.g. to format the rows alternately in the output of a table.

Parameter	Required	Default	Impact
`values`	yes	-	two comma-separated values, between which the circulation takes place
`name`	no	default	internal name of the cycle
`print`	no	1	Werte ausgeben
`advance`	no	1	automatically etrieve next value
`reset`	no	0	reset

Table 6.4.: Parameter of the {cycle}-function

6.4.5. fetch

The `{fetch}`-function facilitates the outputting of file content. These can be stored in the local data system or be retrieved through HTTP or FTP. Internally, the php-function `file_get_contents()` is used.

Programm 6.10 Example of the use of the {cycle}-function

```
{foreach from=$boards item=child}
 ...
 <tr style="background-color: {cycle values="#eee,#fff"}">
 ...
 </tr>
 ...
{/foreach}
```

The parameter `assign` is optional in this case. With this, you can save the file contents in a template variable and only output at a later point of time.

```
{fetch file='x.html'}
{fetch file='x.html' assign=var}
```

This is technically also a `TemplateCompilerPlugin`.

6.4.6. htmloptions

The function `{htmloptions}` creates an output of an HTML-Select-Box with the corresponding option elements.

Programm 6.11 Example of the use of the {htmloptions}-function

```
{* Possible Array containing values *}
$array = array(
 "key1" => "value1",
 "key2" => "value2"
 "option group" => array(
  "keyA" => "valueA",
  "keyB" => "valueB"
 );
);

{* Possible usage of htmloptions *}
{htmloptions options=$array}
{htmloptions options=$array selected=$selected}
{htmloptions options=$array selected=$selectedArray}
{htmloptions options=$array name="x"}
{htmloptions output=$outputArray}
{htmloptions output=$outputArray values=$valueArray}

{* Possible Display *}
<select>
 <option label="output" value="value">output</option>
</select>
```

Parameter	Required	Default	Impact
`name`	no	-	Name of the select-box.
`options`	yes[a]	-	Specification of an associative array with options-values and options-names.
`output`	yes[b]	-	Specification of an array with options-names.
`values`	yes[c]	-	Specification of an array with values of the corresponding options.
`selected`	no	-	The specified option has already been highlighted (multiple highlighting is possible through an array).
`disableEncoding`	no	0	Opens the `htmlspecialchars()`-method internally to protect from incorrect entries.

[a] Alternatively, `output` and `values` or just `output` can be used respectively
[b] Alternatively, `options` can be used.
[c] Can only be used in connection with `output`. For empty values, an options group is created. Subsequent values should be saved in a nested array.

Table 6.5.: parameters of the {htmloptions}-function

6.4.7. htmlcheckboxes

The function `{htmlcheckboxes}` creates a dump of several checkbox elements as HTML-code.

Programm 6.12 Example of the use of the {htmlcheckboxes}-function

```
{* Possible use of htmlcheckboxes *}
{htmlcheckboxes name="x" options=$array}
{htmlcheckboxes name="x" options=$array selected=$selected}
{htmlcheckboxes name="x" options=$array selected=$selectedArray}
{htmlcheckboxes name="x" output=$outputArray}
{htmlcheckboxes name="x" output=$outputArray values=$valueArray}

{* Possible display *}
<label><input type="checkbox" name="x[]" value="value" />output</label>
```

6.4.8. implode

With `{implode}` you can output the contents of arrays, similar to what you are used to from `{foreach}`. The only difference is that the elements are automatically connected

Parameter	Required	Default	Impact
`name`	yes	-	Name of the checkbox-list.
`options`	yes[a]	-	Output of an associative array with checkbox-values and -labels.
`output`	yes[b]	-	Specification of an array with checkbox-labels.
`values`	yes[c]	-	Specification of an array with values of the corresponding checkboxes.
`selected`	no	-	The element specified has already been highlighted (multiple highlighting is possible through an array).
`disableEncoding`	no	0	Opens the `htmlspecialchars()`-method internally to protect from incorrect entries.
`separator`	no	-	ADesignation of text/HTML to be inserted before every element.

[a] Alternatively, `output` and `values` or just `output` can be used respectively.
[b] Alternatively, `options` can be used.
[c] Can only be used in connection with `output`.

Table 6.6.: Parameters of the {htmlcheckboxes}-function

through a string. For this, there is the parameter 'glue' for which you can specify an optional string. By default, "," is assumed.

If for example you want to implement a dump of names and separate these through a comma, `{implode}` is the best choice. With `{foreach}`, you would have to create a special counter to bypass the problem that there is a separator in front of every element (because no separator is supposed to be in front of the first element).

Example 6.13 shows the use of the `{implode}`-function.

Programm 6.13 Example of the use of the {implode}-function

```
{* Possible use of implode *}
{implode from=$names key=key item=name}{$key}: {$name}{/implode}<br />
{implode from=$names item=name glue=";"}{$name}{/implode}<br />

{* Possible display *}
0: Name a, 1: Name b, 2: Name c
Name a;Name b;Name c
```

6.4.9. lang

With the `{lang}`-function you can call up language variables. More on the topic of language variables can be found in section 7.2 on page 48.

Within `{lang}`, specify the name of the designated language variable. If this name is dynamic, i.e. derived from another variable or you assign other parameters to the language variable, this means that this is a dynamic language variable. See example 6.14.

Please note that it with likely only be possible with WCF 1.1 to use template scripting in dynamic language variables.

Programm 6.14 Example of the use of the {lang}-function

```
{* static language variable *}
{lang}wcf.example{/lang}

{* dynamic language variable *}
{lang}wcf.example.$foo{/lang}
{lang foo=$foo bar=$bar}wcf.example{/lang}
```

Technically, this is a `TemplateCompilerPlugin` - i.e. the method is already called during the compiling of the template.

6.4.10. pages

With the help of the `{pages}`-function, you can easily create a page number navigation. If you spread the content of one section over several pages, this lets you put out the individual page numbers.

Parameter	Required	Default	Impact
`pages`	yes	-	number of pages
`page`	no	-	current page
`link`	yes	-	Link to the respective page (see code example).
`assign`	no	-	Output can be assigned to a variable.
`print`	no	1	Output of the page numbers.

Table 6.7.: Parameters of the {pages}-function

Programm 6.15 Examples for the use of the of the {pages}-function

```
{* creates 10 page numbers *}
{pages pages=10 link='page-%d.html'}

{* creates 10 page numbers with 8 being marked as the current page  *}
{pages page=8 pages=10 link='page-%d.html'}

{* assigns the page numbers to the variable output, no output *}
{pages page=8 pages=10 link='page-%d.html' assign='output'}

{* assigns the page numbers to the variable 'output', incl. output *}
{pages page=8 pages=10 link='page-%d.html' assign='output' print=true}
```

6.5. Included modifiers

6.5.1. concat

The `concat`-modifier facilitates the connection of several strings. The following examples returns a string that in PHP-syntax would be created by `"left".$right`.

```
{"left"|concat:$right}
```

6.5.2. date

The `date`-modifier converts a Unix timestamp into a format legible by humans. The standard format for dates contains the year, the month and the day in the following display: "5. November 2007".

```
{$timestamp|date}
{"132845333"|date:"%Y-%m-%d"}
```

6.5.3. encodejs

With the help of the `encodejs`-modifiers, it is possible to format strings of the use Single-Quote Javascript-String. Single quotes and word-wraps are escaped in the process.

```
{$string|encodejs}
{"bl''ah"|encodejs}
```

6.5.4. filesize

The `filesize`-modifiers formats file sizes that are specified in bytes.

```
{$string|filesize}
{123456789|filesize}
```

6.5.5. fulldate

The `fulldate`-midifier formats a Unix timestamp into a date in the following display: “Monday, 5. November 2007, 11:32”.

```
{$timestamp|fulldate}
{"132845333"|fulldate}
```

6.5.6. shorttime

With `shorttime`, you can format a Unix timestamp with a date in the following display: “5. November 2007, 11:32”.

```
{$timestamp|shorttime}
{"132845333"|shorttime:"Y-m-d h:ia"}
```

6.5.7. time

The `time`-modifier formats a Unix timestamp with a date in the following display: “Monday, 5. November 2007, 11:32”. This is difference from `fulldate` because the date is replaced by “Today” or “Yesterday” respectively.

```
{$timestamp|time}
{"132845333"|time:"%Y-%m-%d %I:%M%p"}
```

6.5.8. truncate

With the `truncate`-modifier, you can cut off a string after a certain number of characters. You can also define anything that should be attached to the string.

```
{$foo|truncate:35:'...'}
```

6.6. Expanding the template system

The template system is very flexible and can be extended by your own modifiers, functions or prefilters. For this, all you need to do is program a PHP-class that implements a certain interface and then save this PHP-class in the folder `wcf/lib/system/template/plugin` (or alternatively install it through a package in this folder).

6.6.1. Custom modifiers

Custom modifiers need to implement the `TemplatePluginModifier`[7] interface. This interface contains the method `execute()`.

string **execute** (*array* $tagArgs, *Template* $tplObj)

`$tagArgs` contains all parameters of the modifier. The template variable content that the modifier is applied to is available in the variable `$tagArgs[0]`. A second parameter of the modifier would be saved in the variable `$tagArgs[1]` , etc.

Additionally, the variable `$tplObj` provides an instance of the template class. As a return value, the changed variable content is expected.

You can see an example in program 6.17 on page 46.

6.6.2. Custom functions

For custom functions, there is the `TemplatePluginFunction`[8] interface with the method `execute()`.

string **execute** (*array* $tagArgs, *Template* $tplObj)

`$tagArgs` is an **assoziatives** array containing the parameters and `$tplObj` is the instance of the the template class. A call in the template `{function parameter="test"}` results in `$tagArgs['parameter']` containg the value “test”.

[7] in the file `wcf/lib/system/template/TemplatePluginModifier.class.php`
[8] in the file `wcf/lib/system/template/TemplatePluginFunction.class.php`

6.6.3. Custom block functions

Customs block functions need to implement the `TemplatePluginBlock`[9].

This interface contains the methods `execute()`, `init()` and `next()`.

string **execute** (*array* $tagArgs, *string* $blockContent, *Template* $tplObj)

void **init** (*array* $tagArgs, *Template* $tplObj)

boolean **next** (*Template* $tplObj)

The method `init()` is the first to be called. As long as `next()` returns *true*, the method `execute()` is called up in a while-loop. In this way, loop functions are realisable. If your block functions are only supposed to be called once, `next()` must only return *true* for the first call. For this, you can for instance use a custom class variable.

The array `$tagArgs` contains one associative array respectively with the parameters of the block function.

Warning: `next()` definitely needs to return *false* at one point, otherwise your block function creates an infinite loop.

6.6.4. Custom prefilters

Custom prefilters need to implement the `TemplatePluginPrefilter`[10]. This interface solely contains the method `execute`.

string **execute** (*string* $sourceContent, *TemplateCompiler* $compiler)

`execute()` can in this case perform any desired operations on the content of a template (`$sourceContent`). Additionally, the current instance of the TemplateCompiler is available through `$compiler`.

As a return value, the (changed) template content is expected.

An example for implementation you can find in program 6.17 on page 46.

[9] in the file `wcf/lib/system/template/TemplatePluginBlock.class.php`
[10] in the file `wcf/lib/system/template/TemplatePluginPrefilter.class.php`

6.6.5. Custom Compiler functions

Compiler functions need to implement the interface `TemplatePluginCompiler`[11] with the methods `executeStart()` and `executeEnd()`.

string **executeStart** (*array* $tagArgs, *TemplateCompiler* $compiler)

string **executeEnd** (*TemplateCompiler* $compiler)

`executeStart()` is called for the opening template tag (`{function param='value'}`) and `executeEnd()` is called for the closing template tag (`{/function}`).

The return value is the respective PHP-Code written in the compiled template file.

The parameters of the opening tag are only available as an associative array in the method `executeStart()`.

An example for implementation you can find in 6.18 on page 47.

[11] in the file `wcf/lib/system/template/TemplatePluginCompiler.class.php`

Programm 6.16 concat Modifier

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
 require_once(WCF_DIR.'lib/system/exception/SystemException.class.php');
 require_once(WCF_DIR.'lib/system/template/TemplatePluginModifier.class.php');
 require_once(WCF_DIR.'lib/system/template/Template.class.php');
}

/**
 * The 'concat' modifier returns the string that
 * results from concatenating the arguments.
 * May have two or more arguments.
 *
 * Usage:
 * "left"|concat:$right
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *   <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginModifierConcat implements TemplatePluginModifier {
 /**
  * @see TemplatePluginModifier::execute()
  */
 public function execute($tagArgs, Template $tplObj) {
  if (count($tagArgs) < 2) {
   throw new SystemException("concat modifier needs two
       or more arguments", 12001);
  }

  $result = '';
  foreach ($tagArgs as $arg) {
   $result .= $arg;
  }

  return $result;
 }
}
?>
```

Programm 6.17 lang Prefilter

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
 require_once(WCF_DIR . 'lib/system/template/TemplatePluginPrefilter.class.php');
 require_once(WCF_DIR . 'lib/system/template/TemplateCompiler.class.php');
}

/**
 * The 'lang' prefilter compiles static language variables.
 * Dynamic language variables will catched by the 'lang' compiler function.
 * It is recommended to use static language variables.
 *
 * Usage:
 * {lang}foo{/lang}
 * {lang}lang.foo.bar{/lang}
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *      <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginPrefilterLang implements TemplatePluginPrefilter {
 /**
  * @see TemplatePluginPrefilter::execute()
  */
 public function execute($sourceContent, TemplateCompiler $compiler) {
  $ldq = preg_quote($compiler->getLeftDelimiter(), '~');
  $rdq = preg_quote($compiler->getRightDelimiter(), '~');
  $sourceContent = preg_replace("~{$ldq}lang{$rdq}([\w\.]+){$ldq}/lang{$rdq}~e",
                'WCF::getLanguage()->get(\'$1\')', $sourceContent);

  return $sourceContent;
 }
}
?>
```

Programm 6.18 lang Compiler Funktion

```
<?php
// imports
if (!defined('NO_IMPORTS')) {
 require_once(WCF_DIR.'lib/system/template/TemplatePluginCompiler.class.php');
 require_once(WCF_DIR.'lib/system/template/TemplateCompiler.class.php');
}

/**
 * The 'lang' compiler function compiles dynamic language variables.
 * Warning: a dynamic language variable does not support template scripting.
 *
 * Usage:
 * {lang}$blah{/lang}
 * {lang var=$x}foo{/lang}
 *
 * @package com.woltlab.wcf.system.template.plugin
 * @author  Marcel Werk
 * @copyright 2001-2007 WoltLab GmbH
 * @license GNU Lesser General Public License
 *  <http://opensource.org/licenses/lgpl-license.php>
 */
class TemplatePluginCompilerLang implements TemplatePluginCompiler {
 /**
  * @see TemplatePluginCompiler::executeStart()
  */
 public function executeStart($tagArgs, TemplateCompiler $compiler) {
  $compiler->pushTag('lang');

  $newTagArgs = array();
  foreach ($tagArgs as $key => $arg) {
   $newTagArgs['$'.$key] = 'StringUtil::encodeHTML('.$arg.')';
  }

  $tagArgs = $compiler->makeArgString($newTagArgs);
  return "<?php \$this->tagStack[] = array('lang', array($tagArgs));
  ob_start(); ?>";
 }

 /**
  * @see TemplatePluginCompiler::executeEnd()
  */
 public function executeEnd(TemplateCompiler $compiler) {
  $compiler->popTag('lang');
  $hash = StringUtil::getRandomID();
  return "<?php \$_lang".$hash." = ob_get_contents(); ob_end_clean();
  echo WCF::getLanguage()->get(\$_lang".
  $hash.", \$this->tagStack[count(\$this->tagStack) - 1][1]);
  array_pop(\$this->tagStack); ?>";
 }
}
?>
```

7. Language administration

The WCF offers a comprehensive language administration that makes it possible to develop user interfaces for applications indepently of a certain language. In the templates, placeholders (so-called language variables) are used that can then be replaced by the system with the corresponding sentence or word.

7.1. Fundamentals

For a better overview, language variables are divided into categories. Language variables and categories are started in language files (`xml`). These are allocated to a package and written into the database by the language administration of the WCF during the package installation.

To achieve optimal performance during the use of language variables, the variables are automatically converted to a quickly readable format by the system and started as a language file (`php`) in the `language`-folder within the WCF directory. For every language, every package and every language category, a separate language file is created. The file name for the language file composes as follows: `packageID_languageID_languageCategory.php`. A file is only created for language variables, that also contain language variables in the respective language and the respective package. Language files are only generated for packages that are standalone applications. All other packages do not have an own interface and cannot directly access language files.

7.2. Use of language variables

The following characters can be used for the description of language variables and categories: "a-zA-Z0-9 _ -". Like with package names, the use of certain namespaces is recommended, e.g. "wcf.acp.global". Categories can only consist of a maximum of three blocks, separated by dots. By means of the namespace "acp" it is now easy to identify all the variables that only occur in the admin-area.

Note: Please pay attention to the fact that the name of a language variable needs to begin with the name of the related category, e.g. "wcf.acp.global.variable".

Within a template, the language variables are then embedded in the following way: `{lang}wcf.global.acp.variable{/lang}`. To use a language variable directly in a PHP-file, please use the method `get("name.of.variable")` of the class `Language`. As an optional paramter, you can pass the method an associative array with values that can be used within the language variable - as you can see in example 7.1 Please be aware that the keys of the array contain the $-character.

Programm 7.1 Example of the use of the method Language::get()

```
// Program code
$value = 3;
WCF::getLanguage()->get('name.of.the.language.variable', array('$value' => $value);
...
// language.xml
<item name="name.of.the.language.variable">
  <![CDATA[A triangle has { $value } corners]]>
</item>
```

If the system does not find any content matching the variable, only the name of the language variable is output. To insert the variable with the correct contents, the package needs to provide the language variables. Language variables are saved in language files.

7.3. Construction of language files

For every language, a separate file is created, as exemplified in program 7.2.

Programm 7.2 Examplary construction of an English language file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE language SYSTEM "http://www.woltlab.com/DTDs/language.dtd">
<language languagecode="en">
  <category name="example.category.name">
    <item name="example.category.name.var1"><![CDATA[example content]]></item>
    <item name="example.category.name.var2"><![CDATA[example content]]></item>
    ...
  </category>
  ...
</language>
```

From the first line of the example it can be established that the language files are started in XML-Format and with UTF8-coding. The WCF also supports other character encodings and also transforms UTF-8-characters into another encoding, if required. The root element of this XML-file is `<language>`. The attribute `languagecode` defines a language code for the recognition of the respective language. For the naming convention, the ISO 639-1-Standard[1] is assumed as a prerequisite.

[1] http://en.wikipedia.org/wiki/ISO_639

Within `<language>`, the categories will then be created. The tag `<category>` contains the attribute `name`, which specifies the name of the language category. Child elements of `<category>` are the language variable with the `<item>` tag. In this case, too, `name` contains the name of the language variables.

Within `item`, the content of the language variables are specified in a CDATA[2]-block. Template scripting and HTML can be inserted here to solve grammatical problems, facilitate the output of certain values or perform certain formattings.

Warning: Do not use template scripting in variables that you call up directly in a PHP-file: `$language->get("name.of.the.language.variable")`

In program 7.3 you can see an output that is required in a poll. If nobody has voted yet, "no votes yet" will be output. If there has been one or more votes, the text "Total x votes" is output. Instead of "x", the current counter reading is output. Furthermore, you can also see how an additional `if`-condition is employed to distinguish between the singular and the plural of the word "vote".

Programm 7.3 Example of template scripting in language variables

```
<item name="wcf.poll.votes">
  <![CDATA[
    {if $poll->votes == 0}
      No votes yet
    {else}
      Total {#$poll->votes} Vote{if $poll->votes > 1}s{/ if}
    {/if}
  ]]>
</item>
```

Detailed information on template scripting you can find in chapter 6 on page 25.

7.4. Embedding language files

After creating a language file, this can now be allocated to a package. In the `package.xml`, reference the language file within the `<instructions>`-block, as seen in program 7.4 on the facing page on the next page. Here, an English language file with the name of `en.xml` is integrated during the installation.

Further information on the topic of Languages-PIP you can find in chapter 13.2.16 on page 86.

[2] http://en.wikipedia.org/wiki/CDATA

Programm 7.4 Integration of an English language file in the `package.xml`

```
...
<instructions  type="install">
  ...
  <languages languagecode="en">en.xml</languages>
  ...
</instructions>
...
```

8. Events

The WCF includes its own event system to facilitate the intervention into the program flow at elementary positions. In that way, individual changes can be brought about without having to modify the original source code files. Events in this case are events that happen during the program flow – e. g. the sending of a form. With a custom `EventListener`, custom methods can be called at this point.

8.1. Triggering events

Events are triggered by the static method `fireAction()` of the `EventHandler`-class. Two parameters are passed to the method:

`$eventObj` The object where the event was triggered.

`$eventName` The name of the event.

In order to be able to use the event, one needs to know first when the event will be triggered. A list of all the events in the WCF you can find in the appendix, 15 on page 99.

8.2. Using events

Now, a custom `EventListener` can be written. For this, the interface of the same name and its method `execute()` need to be implemented. Three parameters are passed by the system in the process:

`$eventObj` The object where the event was triggered.

`$className` The name of the class of the event-object.

`$eventName` The name of the event.

Within the method you can now define what is supposed to happen in the case of the events. After having written an `EventListener`, you need to register it in the system. For this, use the EventListener-PIP, which is explained in chapter 13.2.1 on page 70.

9. Sessions

The session[1] system of the WCF provides an easy way of providing and saving session data.

The class `WCF` calls the method `initSession()` with every page view. In the process, a `Session` instance is retrieved from the `CookieSessionFactory`. The terms `SessionFactory` and `Session` are illustrated further below.

9.1. SessionFactory

A `SessionFactory` exists to create a `Session` or retrieve it from the database respectively. Through this class, the sessionlink is merely supported. That means that the `Session` is retrieved on the basis of the URL. Throughout the whole administrative area, a `Session` adjustment only happens due to the sessionlink.

The `CookieSessionFactory` extends the `SessionFactory` by the option of identifying the active `Session` with the help of a cookie. Thus, users can automatically be recognized by the system when returning to the page after several days.

To use a custom `SessionFactory`, the `initSession()`-method of the `WCF`-class needs to be overwritten.

9.2. Session

The `Session`-object contains all important (user-)data of a session. Within the object, by default the user agent is used to verify whether it really is the same user. It is also possible to check the IP-address, however this version is not activated by default, because many providers give out changing IP-addresses.

The `CookieSession` extends the `Session`-class by the function of saving the session agent in the cookie. In that way, visitors returning to the website can be automatically recognized. In the standalone application Burning Board, the `CookieSession` is additionally extended by an own `WBBSession` that provides particular information only

[1] http://en.wikipedia.org/wiki/Session_(computer_science)

used by the forum system. Thus, there are for example an own texttttWBBUser- and a `WBBGuestSession`, because some data does not exist at all for guests of the forum.

To be able to temporarily use data in a session, the session variables need to be used:

void **register (*string* $key, *string* $data)** Use this method to assign a value (`$data`) to a key (`$key`).

void **unregister (*string* $key)** To remove a value from a Session, call the `unregister()`-method using the corresponding key.

array **getVars ()** This method returns all the variables saved in an associative array.

mixed **getVar (*string* $name)** To receive a single variable from the session, please use this method, passing the key with which the variable was saved.

10. Caching

A special feature of the WCF is the included caching-system. With this a database can be relieved. Other web technologies such as JEE[1] work with a so-called application scope, in which data can be deposited that needs to be available throughout the entire application. Those are often components of an application that are rarely changed, but often used. For PHP-applications, there is no application scope. For this reason, the WCF offers its own caching-system which helps to buffer the files. To use the system, the interface `CacheBuilder` and its method `getData()` need to be implemented. This method returns an arbitrary value that is serialized and saved in a file in the data system. The lacking type reliability of PHP is utilized here. A user-defined type can be returned – an object, an array or a primitive data type. Within the method, a more complex database query is normally executed. An exemplary `CacheBuilder` is illustrated in program 10.1.

Programm 10.1 Example of a custom CacheBuilder

```
class CacheBuilderTest implements CacheBuilder {
  /**
   * @see CacheBuilder::getData()
   */
  public function getData($cacheResource) {
    $data = array();

    // get data from DB
    $sql = "SELECT  *
            FROM    test";
    $result = WCF::getDB()->sendQuery($sql);
    while ($row = WCF::getDB()->fetchArray($result)) {
      $data[]= $row;
    }
    return $data;
  }
}
```

In order to use a prepared `CacheBuilder`, it needs to be registered in the system. Subsequently, files can be enquired from the cache, as can be seen in progam 10.2 on the next page on the next page:

Through the static method `getCache()`, the global cache-object of the system can be accessed. First of all, a cache-Resource is created. The method `addResource()` expects three parameters:

[1] http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

Programm 10.2 Example of the use of the CacheBuilder

```
WCF::getCache()->addResource(
  'test-'.PACKAGE_ID,
  WCF_DIR.'cache/cache.test-'.PACKAGE_ID.'.php',
  WCF_DIR.'lib/system/cache/CacheBuilderTest.class.php');

$this->testData = WCF::getCache()->get('test-'.PACKAGE_ID);
```

1. Name of the Cache-Resource – the Package-ID of the standalone application should be used here .
2. Path to the cache-file that is stored on the data system – in this example a file is stored in the cache-folder of the WCF-directory.
3. Path to the `CacheBuilder`-class.

In the second command of the code-example, the saved files are retrieved from the cache-file through the method `get("name of the Cache-Resource")`.

To create a new cache or to reset, the cache-file needs to be deleted. At the next call, a new cache is automatically created. A cache-resource is deleted with the following code:

```
WCF::getCache()->clearResource('Name of the Cache');
WCF::getCache()->clear(WCF_DIR . 'cache/', 'cache.Name.php');
```

11. RequestHandler & the page-, form- and action-classes

Internet applications are controlled through requests to the server. Here, the `RequestHandler` offers a uniform gateway for the processing and control of HTTP-requests.

11.1. RequestHandler

The `RequestHandler` can be used via its static method `handle()`. This excepts as the only parameter an array with path designations to the directories of the respective libraries. In this case, this means the `lib`-directory of the WCF and of the respective standalone application. When executing the method, the system checks the HTTP-request for POST- or GET-variables and opens or processes the corresponding site respectively.

There are three different types of sites:

Page Simple sites in the WCF that are needed for the dump of information.

Form Special blank sites, into which data can be entried and processed.

Action For the execution of actions, without the display of a resulting information- or blank site.

A possible URL would for example be: "`.../index.php?page=Test`". The `RequestHandler` will now try to create an object of the class `TestPage`. For the link "`.../index.php?form=Test`", the class `TestForm` would be searched for. Within this class, further parameters can then be processed.

For these types of sites, the WCF offers an interface and a standard implementation within an abstract class each.

Note: It is important that you keep to the naming scheme used in the WCF when naming your classes: Class names have to being with a capital letter, followed by the type of class (page, form or action).

11.2. Page and AbstractPage

The Page-Interface defines the following methods:

readParameters() The POST- and GET-Parameters are read out.

readData() Files needed for the display of the site are readout or assembled respectively.

assignVariables() Variables that are meant to be used in the template, are to be passed on to the Template-Engine.

show() The site is displayed.

`AbstractPage` is the standard implentation of the `Page` interface. With all four methods, the corresponding event is triggered. When deriving the class, bear in mind that when overwriting a method, either the event itself is triggered or the parent method is called up, as can bee seen in program example 11.1. In addition to this, two variables are passed to the template-engine. These can then be used in the template `test.tpl`.

Programm 11.1 Derivation of AbstractPage

```
class TestPage extends AbstractPage {

  public $templateName = 'test';

  /**
   * @see Page::assignVariables()
   */
  public function assignVariables() {
    parent::assignVariables();

    WCF::getTPL()->assign(array(
      'test' => "first testVar",
      'test2' => "second testVar"
    ));
  }
}
```

The class `AbstractPage` is a good start for the realization of simple sites. It is recommended to take another close look at this class prior using it. Other abstract classes such as `MultipleLinkPage` and `SortablePage` can be used for more complex sites.

11.3. Form and AbstractForm

The `Form` interface expands the page interface by the follow methods:

submit() On the sending of the form, this method is called.

validate() Entries into the blank can be validated.

save() This method is used for the saving of form data.

readFormParameters() To read the form entries.

Here, too, an abstract class `AbstractForm` exists that should be used for the form pages.

11.4. Action and AbstractAction

To process user entries without displaying own result- or form-pages, the interface `Action` should be used. The following methods are defined:

readParameters() On the sending of the form, this method is called.

execute() The action is executed.

Furthermore, the abstract class `AbstractAction` defines the method `executed()`, for the methods that are to be carried out after an action. However, this needs to be called manually, whereas the methods `readParameters()` and `execute()` are automatically called when an instance is written.

Part II.

Create packages

12. WCF-packages

In this chapter you will find information on how to create your custom package in the WCF. This covers the format, the special characteristics of the `package.xml`-file and the different package types.

12.1. The format

Packages are `.tar` (or `.tar.gz`) archives, that always contain an XML file called `package.xml` and, depending on the package, other additional files.

```
+ paket.tar
  | ...
  | package.xml
  | ...
  | ...
```

Recommended software:

- tar & gzip (or e.g. 7Zip)
- an XML-Editor

12.2. The `package.xml` file

The `package.xml`-file needs to be located in the main folder in the `.tar`-archiv – moving the file into a subfolder is not possible.

The `package.xml` contains meta-information on the package as well as instructions for installation and the update of earlier version. Table 12.1 on page 63 shows all required and possible XML-tags in the `package.xml`-file.

12.2.1. The package identifier

The package identifier is a unique name used to identify the package. The WCF is geared to Java package names in that respect. To verify the package identifier, the method `Package::isValidPackageName()` is used.

Important:

A valid package identifier consists of at least three parts, which need to be separated by a dot (.) respectively. Every part has to consist at least of an alphanumerical[1] character or the underscore (_). Names can also be separated into more than three parts, e.g. "'com.woltlab.wcf"'.

12.2.2. Multilingual package names and package descriptions

To deliver a package in multiple languages in a single file, the package name as well as the description of the package in the `package.xml`-file can be defined for multiple languages. For the tags `<packagename>` and `<packagedescription>` there exists therefore the parameter `language`, which contains the language code according to the ISO 639-1-Standard[2].

Programm 12.1 Example of multiple package names

```
<package name="com.woltlab.wcf.data.page.legalNotice">
  <packageinformation>
    <packagename>Legal Notice Page</packagename>
    <packagedescription></packagedescription>
    <packagename language="de">imprint</packagename>
    <packagedescription language="de">Commercial German internet pages need to include a disclaime
    ...
  </packageinformation>
  ...
</package>
```

12.2.3. `<requiredpackage>` Tag

To highlight the dependencies of one package on other packages, there is the tag `<requiredpackage`
This refers to the package identifier of the package needed. That package must be installed before the current package can be installed.

Through the optional parameter `minversion` you can define a minimal version of the package, that needs to be installed at the least.

[1] a to z, A to Z or 0 to 9
[2] http://en.wikipedia.org/wiki/ISO_639

Tag	Parameter	Impact
`<package>`	yes, see 12.2.1	
· `<packageinformation>`	–	contains meta-information on the package
· · `<packagename>`	yes, see 12.2.2	name of the package
· · `<packagedescription>`*[a]	yes, see 12.2.2	short description of the package
· · `<version>`	–	package version number[b]
· · `<date>`*	–	release date of this version[c]
· · `<isunique>`*	–	Can this package only be installed once (0 or 1)?
· · `<standalone>`*	–	Is this package a standalone application[d] (0 or 1)?
· · `<plugin>`*	–	name of the superordinate package
· · `<packageurl>`*	–	Website of the package with further information
· `<authorinformation>`*	–	contains meta-information on the package producer
· · `<author>`*	–	author's name
· · `<authorurl>`*	–	author's website
· `<requiredpackages>`*	–	contains a list of packages required by this package
· · `<requiredpackage>`*	yes, see 12.2.3	name of the required package [e]
· `<optionalpackages>`*	–	contains a list of packages that can be optionally installed with this package
· · `<optionalpackage>`*	yes, see 12.2.4	name of the optional package
· `<instructions>`*	yes, see 12.2.5	of instructions for the installation or the update of the package. The distinction between installation and update is made through parameters(see 12.2.5). Within the `<instructions>` tag, so-called PackageInstallationPlugins are called.

[a] tags marked with asterisk (*) are optional

[b] It is recommended to use version numbers split into three parts: X.Y.Z. Appendices such as "Alpha N", "Beta N", "RC N" are also possible. See also PHP function monoversion_compare, http://www.php.net/manual/en/function.version-compare.php

[c] English format, compatiable with the PHP function `strtotime` (http://www.php.net/manual/en/function.strtotime.php), e. g. YYYY-MM-DD

[d] see 12.3 on package types

[e] the optional declaration of a minimal version and the reference to the included package file are possible through parameters, see also 12.2.3.

Table 12.1.: Tags in the `package.xml` file

To stop the installation from canceling when a required package is not yet installed, the location of the package file can also be directly defined by the `file` parameter. This can be an URL (http or ftp) or a relative path referring to a package file within the current package file. The required package will then be automatically downloaded or unpacked and installed from the current package file respectively.

See some tangible examples in table 12.2.

Programm 12.2 List of required packages

```
<requiredpackages>
  <requiredpackage minversion="1.0.0" file="requirements/com.woltlab.wcf.tar">
    com.woltlab.wcf
  </requiredpackage>
  <requiredpackage minversion="1.0.0" file="requirements/com.woltlab.wcf.data.page.tar">
    com.woltlab.wcf.data.page
  </requiredpackage>
</requiredpackages>
...
<requiredpackages>
  <requiredpackage minversion="1.0.0" file="http://server.com/paket.tar.gz">
    package
  </requiredpackage>
</requiredpackages>
```

12.2.4. `<optionalpackage>` Tag

Modelled on the <requiredpackage>-Tag, see 12.2.3

12.2.5. Instructions for installation and updates

The tag `<instructions>` contains a list of instructions for the installation or the update of a package. Via the parameter type, the distinction is made between installation and update: `type='install'` or `type='update'` respectively.

In the case of an update, the required parameter `fromversion` needs to be used to determine the versions that can be updated. In the designation of `fromversion`, space-holders (*) are therefore allowed too to allow update of a whole series of programs: e.g. `fromversion='3.0.0 RC 1'`, `fromversion='1.1.*'`, `fromversion='*'`.

Within the `<instructions>`-tags, the single PIPs[3] are then referred to. There are PIPs for the installation of files (`<files>files.tar</files>`), the installation of templates (`<templates>templates.tar</templates>`) or for the execution of SQL-commands (`<sql>install.sql</sql>`) as well as many more. You will learn more about PIPs in the following chapter.

[3] PackageInstallationsPlugins

12.3. Different package types

The package system differentiates between three types of packages.

Package The standard type. Files of these packages are installed into the WCF-directory. A package normally provides program libraries that can then be uses by standalone applications.

Standalone Application A standalone application is an independent appliaction based on the WCF. During the installation, the user needs to define a separate installation folder.

Plug-in A plugin is a package that is tied to an existing package and delivers additional functions or extensions for this packages. For the creation of the plugin it is sufficient to specify the name of the related package through the `<plugin>` tag in the `package.xml`. Compare table 12.1 on page 63.

13. Package Installation Plugin

A Package Installation Plugin, short PIP, is a plugin that can extend the installation process of packages by new functions. A PIP can then for example access a package archive, extract files, parse XML-files and write data into the database. The possibilities are manifold. In the process, the PIP is responsible for the installation as well as the de-installation of its files (for this, the PIP needs to "log" the changes made, if applicable).

A PIP is mapped onto a corresponding PHP-class, that implements the interface `Package-Install`. Within the `<instruction>`-block of the `package.xml`, a PIP is called through an XML-code in the form of `<pip>file</pip>`. I.e. there is always a reference to the file that the PIP is then supposed to engage with. If the file is not located in the main directory of the package, the path should be specified relatively.

In the following, all PIP are introduced that are included in the WCF or the free packages. For a better overview, they have been divided into three categories: file-based PIPs, Import-PIPs (XML) and Other PIPs. In the following, there is also an explanation of how to realize custom PIPs.

13.1. File-based PIPs

For File-based PIPs, an installation of files always takes place, which are then saved in a Tar-archive.

13.1.1. The Files-PIP

Purpose

The Files-PIP is used for the installation of files.

XML-Code in the `package.xml`

```
<files>files.tar</files>
```

Mode of operation

The folder structure within the archive is copied completely into the installation directory. End application files or their plugins are copied into the directory of the end application. For other packages, the WCF-directory is the target destination. For every file, the installation is recorded in the database. In the table `wcf_package_installation_file_log`, the file name and the package ID are saved.

Note: If in the target directory a file of the same name already exists, there are two possibilities:

1. The file was installed by a different package:
   In this case, the installation is terminated. A package cannot overwrite files of a different package.

2. The file is unknown to the system:
   The user receives a security warning asking if the file should be overwritten. The files has probably been previously copied into the folder manually.

13.1.2. The Templates-PIP

Purpose

The Templates-PIP is used for the installation of templates.

XML-Code in the `package.xml`

```
<templates>templates.tar</templates>
```

Mode of operation

During the package installation, the archive-file is unpacked and the contents are copied into the designated template folder. Every standalone application has got its own template folder, other package types use the template folder of the WCF. If template names are duplicate, the template names are extended by the package-ID. In addition, all template names are saved in the database together with the package-ID.

Note: Like with the Files-PIP, template names are only allowed occur once.

13.1.3. ACPTemplates-PIP

Purpose

The ACPTemplates-PIP is used to install the templates of the administration interface.

XML-Code in the `package.xml`

```
<acptemplates>acptemplates.tar</acptemplates>
```

Mode of operation

See Templates-PIP 13.1.2 on the preceding page on the previous page.

13.1.4. The Style-PIP

Purpose

With the help of the Style-PIPs, it is possible to install a style with a package.

XML-Code in the `package.xml`

`<style>style.tar</style>` Through the optional parameter `default="true"`, the style can also be set as default during the installation.

Mode of operation

The archive-file mandatorily needs to contain a file called `style.xml`. Comparable to `package.xml` for packages, it contains all important information on a style. An exemplary structure of a file like this can be seen in code example 13.1 on the next page on the following page.

While in the `<general>`- and `<author>`-block only meta-informations are given, the `<files>`-block contains important information that is used during the installation of the style.The archive `images.tar` contains all image files required by the style. In the `variables.xml`, all values of the style variable are determined.

Note: All style variables and their possible values can be found in the appendix, in chapter 16 on page 101.

Programm 13.1 Exemplary structure of a `style.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE style SYSTEM "http://www.woltlab.com/DTDs/SXF/style.dtd">
<style>
  <general>
    <stylename><![CDATA[WoltLab Blue Sunrise]]></stylename>
    <description><![CDATA[The WoltLab Burning Board 3 default style.]]></description>
    <version><![CDATA[1.0.0]]></version>
    <date><![CDATA[2007-07-17]]></date>
    <image><![CDATA[WoltLab Blue Sunrise.png]]></image>
    <copyright><![CDATA[WoltLab GmbH]]></copyright>
    <license><![CDATA[Commercial]]></license>
  </general>
  <author>
    <authorname><![CDATA[Arian Glander, Harald Szekely]]></authorname>
    <authorurl><![CDATA[http://www.woltlab.com/]]></authorurl>
  </author>
  <files>
    <variables>variables.xml</variables>
    <images>images.tar</images>
  </files>
</style>
```

13.1.5. The PIPs-PIP

Purpose

The PackageInstallationPlugins-PIP is used for the installation of PIPs. Like this, the system can be expanded by new PIPs.

XML-Code in the `package.xml`

```
<packageinstallationplugins>pip.tar</packageinstallationplugins>
```

Mode of operation

During the installation, the archive file is unpacked and the individual PIPs are copied into the folder `wcf/lib/acp/package/plugin`. In addition, the PIPs are registered in the database, together with the package-ID.

13.2. Import PIPs (XML)

The PIPs listed here all have got one thing in common: All are based on XML-files, whose informations are to be entered in the database – i.e. imported. The structure of such an XML-file is often very similar and is exemplified in program 13.2 on the next page.

Programm 13.2 Exemplary structure of an Import PIP XML-file

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/custom.dtd">
<data>
  <import>
    <custom>
    ...
    </custom>
    ...
  </import>
</data>
```

The starting XML-tag is followed by the specification of the doctype with reference to the corresponding DTD. The root tag `<data>` includes all other tags. In this `<import>`-tag follows, which contains the single elements that are to be imported. These differ from PIP to PIP. For some PIPs, after the `<import>`-tag, a `<delete>`-tag can also be specified to delete certain files again. This is also separately so pointed out for the respective PIPs.

13.2.1. The EventListener-PIP

Purpose

The EventListener-PIP is used to register the `EventListener` with the system.

XML-Code in the `package.xml`

```
<eventlistener>eventlistener.xml</eventlistener>
```

Tags and their meaning

<eventlistener> Defines the `EventListener` that is to be installed.

- **<eventclassname>** Name of the class that triggers the event.
- **<eventname>** Name of the event.
- **<listenerclassfile>** relative path designation for the `EventListener`-class. After convention, these are stored in the folder `lib/system/event/listener`. For the installation of this file, use the Files-PIP.
- **<environment>** If the event happens inside the administration range, “admin” is entered here. Otherwise, please enter “user”. This is the default value, so this tag can also be omitted in this case.

· **<inherit>** If the `EventListener` is also supposed to be called at classes that are inherited from the class specified in the `<eventclassname>`-tag, enter the value 1 here. Otherwise, enter 0 or leave the tag blank. *Note:* The inheritance uses up additional calculating time. Please only use this function if really necessary.

Code example

Programm 13.3 Exemplary structure of an `eventlistener.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/eventListeners.dtd">
<data>
  <import>
    <eventlistener>
      <eventClassName>URLParser</eventClassName>
      <eventName>didParse</eventName>
      <listenerClassFile>
        lib/system/event/listener/URLParserThreadURLListener.class.php
      </listenerClassFile>
    </eventlistener>

    <!-- admin -->
    <eventlistener>
      <eventClassName>UserListPage</eventClassName>
      <eventName>assignVariables</eventName>
      <environment>admin</environment>
      <listenerClassFile>
        lib/system/event/listener/UserListPagePermissionsButtonListener.class.php
      </listenerClassFile>
    </eventlistener>

    ...
  </import>
</data>
```

13.2.2. The Cronjobs-PIP

Purpose

The Cronjobs-PIP is used to register Cronjobs with the system. Cronjobs are time-controlled tasks known from the world of Unix. Real cronjobs are not available most of the time as they require their own server. That is why the functionality of Cronjobs is imitated by AJAX and PHP. Through this simulation, regular tasks can still be executed. These tasks could include the refreshment of readouts. Often, certain readouts are buffered (caching), because they are frequently required by users and to relieve the database. With the help of a Cronjob, a buffer like that can for instance be update once a day.

XML-Code in the `package.xml`

```
<cronjobs>cronjobs.xml</cronjobs>
```

Mode of operation

Your custom Cronjob can be written by implementing the `Cronjob`-Interface and its `execute()`-method. Within the method, any desired action can be activated. To use the compiled Cronjob-class, it needs to be entered through the administration area of the WCF or installed via the PIP described here. The system manages all entered Conjobs and automatically runs the corresponding cronjob when its turn comes around. Here, more complex operations can also be conducted. The user who initiates the Cronjob by calling the page, does not notice any of this, because through the use of AJAX, the whole operation runs asynchronously in the background.

Tags and their meaning

<cronjob> Contains the tags described below.

- **<classpath>** Path specification of the PHP-file that contains the corresponding class to be executed. The specification needs to be relative to the installation directory of the selected end application.
- **<description>** A short description of the Cronjob's task.
- **<startminute>** At these minutes (0 - 59) the task is to be carried out, otherwise please enter the *-symbol.
- **<starthour>** At these hours (0 - 23) the task is to be carried out, otherwise please enter the *-symbol.
- **<startdom>** On these days of the month (1 - 31) the task is to be carried out, otherwise please enter the *-symbol.
- **<startmonth>** In these months (1 - 12 or jan - dec) the task is to be carried out, otherwise please enter the *-symbol.
- **<startdow>** On these days of the week (0 - 6 with Sunday = 0 or mon - sun) the task is to be carried out, otherwise please enter the *-symbol.
- **<execmultiple>** Multiple execution (0 - no, 1 - yes): through the activation of this option, the task is executed multiple times, if there are additional execution dates between the last execution date and the current date.
- **<canbeedited>** Can this Cronjob be edited later through the ACP (0 - no, 1 - yes).

· **<canbedisabled>** Can this Cronjob be deactivated later through the ACP (0 - no, 1 - yes).

Code example

Programm 13.4 Exemplary structure of a `cronjobs.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/cronjobs.dtd">
<data>
  <import>
    <cronjob>
      <classpath>lib/system/cronjob/CleanupCronjob.class.php</classpath>
      <description>Hourly Cleanup</description>
      <startminute>0</startminute>
      <starthour>*</starthour>
      <startdom>*</startdom>
      <startmonth>*</startmonth>
      <startdow>*</startdow>
      <execmultiple>0</execmultiple>
      <canbeedited>0</canbeedited>
      <canbedisabled>0</canbedisabled>
    </cronjob>
    ...
  </import>
</data>
```

13.2.3. The Options-PIP

Purpose

The Options-PIP is used to define the settings options within the adminstrative interface.

XML-Code in the `package.xml`

```
<options>options.xml</options>
```

Tags and their meaning

<categories> Contains a list of categories.

· **<category>** This category is used to sort the options. The tag contains the attribute `name`. This specifies the name of the category.

· · **<parent>** Specification of the parent category's name.

· · **<showorder>** Designation of a number defining the position of the category in relation to other categories on the same level.

<options> Contains a list of options.

· **<option>** The tag contains the attribute `name`. This specifies the name of the option.

· · **<categoryname>** Specifies the name of the category to which the option is allocated.

· · **<optiontype>** Describes the option type. For a list of available option types, please see the appendix. It is also possible to write custom option types.

· · **<showorder>** Position of the option within the specified category.

· · **<defaultvalue>** Default value of the option.

· · **<hidden>** With `<hidden>1</hidden>`, options can be hidden. These are for example created during the installation (e.g. date of the installation), but are on the one hand not displayed and on the other not changeable.

· · **<selectoptions>** If for the options type `radiobuttons` or a `select`-type have been selected, possible options can be speficied through this tag. Every line contains an option structured in the way described below:
`Value:Name of the language variable`
Please do not format the lines with tabs or blanks, just use the line break.

· · **<enableoptions>** If `boolean` is used as an option type and you wish to create a link to other options, use `<enableoptions>`. This means when this option is activated, the options listed under `<enableoptions>` are also visible. Specify the name of the respective options and separate them using commas.

· · **<validationpattern>** Specification of a regular expression used for the validation of this option.

For the Options-PIP, in addition to the `<import>`-block, the `<delete>`-block can be used to remove certain options from the system.

Code example

13.2.4. The UserOptions-PIP

Purpose

The UserOptions-PIP facilitates the creation of settings for registered users. While with the Options-PIP the settings are later visible for the administrator in the corresponding interface, the UserOptions-PIP defines the settings that a member can choose for his profile.

Programm 13.5 Exemplary structure of an `options.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/options.dtd">
<data>
  <import>
    <categories>
      <category name="offline"></category>
      <category name="offline.general">
        <parent>offline</parent>
      </category>
      ...
    </categories>
    ...
    <options>

      <option name="page_title">
        <categoryname>general.page</categoryname>
        <optiontype>text</optiontype>
        <showorder>1</showorder>
        <defaultvalue>WoltLab Burning Board</defaultvalue>
      </option>

      <option name="board_default_days_prune">
        <categoryname>board.threads</categoryname>
        <optiontype>select</optiontype>
        <defaultvalue>1000</defaultvalue>
        <selectoptions><![CDATA[
          1:wbb.board.filterByDate.1
          3:wbb.board.filterByDate.3
          7:wbb.board.filterByDate.7
        ]]></selectoptions>
      </option>
      ...
    </options>
  </import>
  <delete>
    <option name="page_title"/>
  </delete>
</data>
```

XML-Code in the `package.xml`

```
<useroptions>useroptions.xml</useroptions>
```

Tags and their meaning

Within the XML-file, all tags can be used that have also been explained for the Options-PIP. Additionally, the following tags are also available:

<categories> see Options-PIP.

· **<category>** see Options-PIP.

· · **<menuicon>** Specification of a path to an icon-file (preferably S-size) to be used for the menu entry.

· · **<icon>** Specification of a path to an icon, which is to be used for the visual identification of the category (preferably M-Size).

<options> see Options-PIP.

· **<option>** see Options-PIP.

· · **<outputclass>** Specification of the name of a class that is to be used for the subsequent output of this option.

· · **<searchable>** With `<searchable>1</searchable>` it can be defined if the specification is also supposed to be searched in a member search.

· · **<visible>** Specification of the group-ID of a user group that can view this option by default. For other groups, it is hidden.

· · **<editable>** Specification of the group-ID of a user group that can edit this option by default.

13.2.5. The GroupOptions-PIP

Purpose

The GroupOptions-PIP is employed to create user rights within the system. These can then later be adjusted for single user groups. This PIP is also very similar to the Options-PIP.

XML-Code in the `package.xml`

```
<groupoptions>groupotions.xml</groupoptions>
```

Tags and their meaning

The same specifications can be made as within the XML-file for the Options-PIP.

13.2.6. The FeedReaderSource-PIP

Purpose

The FeedReaderSource-PIP is used to register sources of RSS-Feeds in the system.

XML-Code in the `package.xml`

```
<feedsource>feedsource.xml</feedsource>
```

Tags and their meaning

<feedsource> Contains the attribute `name` which defines the name of the feed.

- **<url>** Specifies the URL to the feed. The example shows that a string-placeholder (`%s`) is used here. This is replaced automatically by the system with the language code of the language employed by the user. You do not need to use the placeholder. If you do use it, do make sure that the corresponding XML-file exists.
- **<cycle>** specifies, how often the information from the feed is to be checked and read. The number of seconds after which the source should be read again needs to be specified here (e.g. after one day in the example given).

Code example

Programm 13.6 Exemplary structure of a `feedsource.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/feedsource.dtd">
<data>
  <import>
    <feedsource name="woltlab-news">
      <url>http://www.woltlab.com/rss_%s.xml</url>
      <cycle>86400</cycle>
    </feedsource>
    ...
  </import>
</data>
```

13.2.7. The Help-PIP

Purpose

The Help-PIP is used to create a custom text within the help function. A help element is comparable to a point in the table of contents of a book.

XML-Code in the `package.xml`

```
<help>help.xml</help>
```

Tags and their meaning

<helpitem> Has the attribute `name` to describe a help element.

- **<showorder>** Expects a number to put the elements in order.
- **<parent>** Specification of the name of the parent element. Like this, a hierarchical structure of help elements can be achieved.
- **<refererpattern>** Within the CDATA-Block, with the help of a regular expression you can specify for which referer-adress the corresponding help element is to be opened. The point of this is that on clicking the help-button, the user is immediately directed to the help-page that corresponds with the current page. That way, one or more pages can refer to a help element.

After setting up the help structure on this file, all you need to do is provide the corresponding content. For this, language files should be used. For every help element, there are two language variables. For the help element `board`, these are the following variables:

```
<item name="wcf.help.item.board">
<item name="wcf.help.item.board.description">
```

The first variable specifies a heading, the second contains the corresponding help text. HTML can also be employed here. The category of the language variables is `<category name="wcf.help.item">`.

Programm 13.7 Exemplary structure of a `help.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/help.dtd">
<data>
  <import>
    <helpitem name="board">
      <showorder>3</showorder>
    </helpitem>
    <helpitem name="board.index">
      <parent>board</parent>
      <showorder>1</showorder>
    </helpitem>
    ...
  </import>
</data>
```

Code example

13.2.8. The BBCodes-PIP

Purpose

The BBCodes-PIP is used to install BB-Codes in the system.

XML-Code in the `package.xml`

```
<bbcodes>bbcodes.xml</bbcodes>
```

Tags and their meaning

<bbcode> With the `name`-attribute of this tag, the name of the bbcode can be specified.

- **<classname>** Specification of the name of a class that deals with the processing of the input.
- **<htmlopen>** The opening HTML-Tag.
- **<htmlclose>** The closing HTML-Tag.
- **<textopen>** The opening text.
- **<textclose>** The closing text.
- **<allowedchilden>** Specification of the names of BBCodes (separated by commas) that this BBCode may contain.
- **<wysiwyg>** This BBCode can be only displayed in the Editor as HTML, if the array "PHP-classname" is empty.

· **<wysiwygicon>** Specification of the file name of the icon to be used in the WYSIWYG-Editor. The icon needs to be located in the folder `icon/wsyiwyg` You should add a language variable for the icon title. Language category: "wcf.bbcode, Name of the language variable: "wcf.bbcode.mycode.title", with "mycode" corresponding exactly to the entry "BBCode-Tag".

· **<attributes>** Contains a list of attributes.

· · **<attribute>** With the parameter `name`, to describe the `<attribute>`.

· · · **<html>** HTML-Code of the attribute.

· · · **<validationpattern>** Speficitation of a regular expression required for the validation of the attribute.

· · · **<required>** This attribute needs to be mandatorily completed.

· · · **<text>** The text / the content of the attribute.

· · · **<usetext>** If this attribute is not completed, optionally the text content of the BBCode can be adopted.

13.2.9. The Smilies-PIP

Purpose

The Smilies-PIP is used to install smilies in the system.

XML-Code in the `package.xml`

```
<smilies>smilies.xml</smilies>
```

Tags and their meaning

<smiley> The smiley-code is saved here in the attribute `name`, which the user will later need to use when writing a message.

· **<title>** Contains the name of the smiley, which is also used for an HTML-Title- and Alt-Tag.

· **<path>** Contains the path to the image file that replaces the smiley-code in the message.

Note: Because in the XML-file, only the path to the smiley image but not the image itself are transmitted, you need to make sure to also have the image copied into the corresponding folder. To do this, please use the Files-PIP.

Code example

Programm 13.8 Exemplary structure of a `smilies.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/smilies.dtd">
<data>
  <import>
    <smiley name=":)">
      <title>smile</title>
      <path>images/smilies/smile.png</path>
    </smiley>
    <smiley name=":(">
      <title>sad</title>
      <path>images/smilies/sad.png</path>
    </smiley>
    ...
  </import>
</data>
```

13.2.10. The SearchableMessageType-PIP

Purpose

The WCF already offers its own search function, including search form and result display. To introduce a message system to the search engine, the SearchableMessageType-PIP needs to be used.

XML-Code in the `package.xml`

```
<searchablemessagetypes>smt.xml</searchablemessagetypes>
```

Tags and their meaning

<smt> This saves the name of the message type in the attribute `name`.

- **<classpath>** A path to a class that impletments the interface `SearchableMessageType` needs to specified.

Additionally, a language variable has to be started. The name is derived from the name of the message type:

```
<item name="wcf.search.type.pm"><![CDATA[private messages]]></item>
```

Programm 13.9 Exemplary structure of a `smt.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/smt.dtd">
<data>
  <import>
    <smt name="pm">
      <classpath>lib/data/message/pm/PMSearch.class.php</classpath>
    </smt>
  </import>
</data>
```

Code example

13.2.11. The PageLocation-PIP

Purpose

There is a page that shows which user is active where on the page. For this, a list of places in the system is required that can be checked against the current URL. The PageLocation-PIP is used to installed possible page locations of the user within the system.

XML-Code in the `package.xml`

```
<pagelocation>pagelocation.xml</pagelocation>
```

Tags and their meaning

<pagelocation> Saves the name of the location in the attribute `name`.

- **<classpath>** Optionally, provide the path to a class that processes this URL, e.g. to find the corresponding topic name to a `threadID`.
- **<pattern>** Contains within the CDATA-Block the regular expression for a URL This is later checked against the current URL of the user.

The name of a page location can also be used as a language variable in this way, for this the corresponding translations need to be manually added in a language file. If for example your page location has the following name, please use this also as the name of the language variable:

```
<pagelocation name="wbb.usersOnline.location.board"></pagelocation>
<item name="wbb.usersOnline.location.board">
     <![CDATA[Forum: {$board}]]>
</item>
```

Code example

Programm 13.10 Exemplary structure of a `pagelocation.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/pageLocations.dtd">
<data>
  <import>
    <pagelocation name="wbb.usersOnline.location.subscriptions">
      <pattern><![CDATA[index\.php\?page=Subscriptions]]></pattern>
    </pagelocation>
    <pagelocation name="wbb.usersOnline.location.thread">
      <pattern><![CDATA[index\.php\?page=Thread&.*threadID=(\d+)]]></pattern>
      <classpath>lib/data/page/location/ThreadLocation.class.php</classpath>
    </pagelocation>
    ...
  </import>
</data>
```

13.2.12. The HeaderMenu-PIP

Purpose

The Header-PIP can be used to create new menu items within the main menu.

XML-Code in the `package.xml`

```
<headermenu>headermenu.xml</headermenu>
```

Tags and their meaning

<headermenuitem> This saves the `name` of the language variable in the attribute name.This then needs to be included in the language file.

- **<icon>** Contains the path to an image file to be used as an icon. The icons in the WCD are used in the main menu in M-size (24px:24px). It is recommended to create icons of the same size for your custom menu items to ensure a consistent appearance.

- **<link>** Specifies which page is to be linked with the menu item.

- **<showorder>** Facilitates the sorting of a menu item. The larger the number, the further back the menu item is located.

Note: Since in the XML-file only the path to the icon-image and the name of the language variable are specified, you need to ensure that the picture and the language variable are installed in the system. For this, use the Files-PIP and the Languages-PIP. If your menu element for example has the following names, please also use this as the name of the language variable:

```
<headermenuitem name="wbb.header.menu.board"></headermenuitem>
<item name="wbb.header.menu.board"><![CDATA[board]]></item>
```

Code example

Programm 13.11 Exemplary structure of a `headermenu.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/headerMenu.dtd">
<data>
  <import>
    <headermenuitem name="wbb.header.menu.board">
      <icon>icon/indexM.png</icon>
      <link>index.php?page=Index</link>
      <showorder>1</showorder>
    </headermenuitem>
    ...
  </import>
</data>
```

13.2.13. The UserCPMenu-PIP

Purpose

The UserCPMenu-PIP is used to insert new menu items in the "My profile" section.

XML-Code in the `package.xml`

```
<usercpmenu>usercpmenu.xml</usercpmenu>
```

Tags and their meaning

This PIP is very similar to the HeaderMenu-PIP. It has the following additional tags:

<usercpmenuitem> See HeaderMenu-PIP.

- **<parent>** Unlike the HeaderMenu, the UserCPMenu can be structured hierarchically. Specify the parent element with `<parent>`.

- **<permissions>** Is used to link the page to a user right. Only if a user has the specified right, he will be able to open the page. To specify multiple rights, just enter them separates by a comma.

Code example

Programm 13.12 Exemplary structure of a `usercpmenu.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/userCPMenu.dtd">
<data>
  <import>
    <usercpmenuitem name="wcf.user.usercp.menu.link.profile">
      <icon>icon/profileM.png</icon>
      <link>index.php?form=UserProfileEdit</link>
      <showorder>1</showorder>
    </usercpmenuitem>

    <usercpmenuitem name="wcf.user.usercp.menu.link.profile.email">
      <icon>icon/emailS.png</icon>
      <link>index.php?form=EmailChange</link>
      <parent>wcf.user.usercp.menu.link.profile</parent>
      <showorder>2</showorder>
      <permissions>user.profile.canChangeEmail</permissions>
    </usercpmenuitem>
    ...
  </import>
</data>
```

13.2.14. The ACPMenu-PIP

Purpose

The ACPMenu-PIP is used to add new menu items in the administration section.

XML-Code in the `package.xml`

```
<acpmenu>acpmenu.xml</acpmenu>
```

Tags and their meaning

This PIP is very similar to the HeaderMenu- and the UserCPMenu-PIP. The only difference to the UserCPMenu-PIP is the tag `<acpmenuitem>`, which is used instead of `<usercpmenuitem>`.

13.2.15. The StyleAttributes-PIP

Purpose

The StyleAttributes-PIP is used to link the attributes of selected CSS-Selectors with style variables. Like this, the application becomes independent of certain styles.

Example: In a template, you have defined an information box:

```
<div class="infoContainer">Informations</div>
```

Now, you would like this box to make use of the style system of the WCF and use an existing color as the background color. You choose the variable`container1.background.color`. For this, you create an StyleAttributes-PIP, as shown in Code example 13.13 on the facing page on the following page.

In this way, you have not permanently encoded a color in the system, but are style-independent. Your information box thus conforms to the respective style. For many elements, these links have already been created. You rarely need to create custom links.

XML-Code in the `package.xml`

```
<styleattributes>styleattributes.xml</styleattributes>
```

Tags and their meaning

<attribute> The style attribute.

- **<selector>** Specification of a CSS-Selector[1] – e.g. `.class`, `#id` or `tag`.
- **<name>** Specification of a CSS-attribute – e.g. `color`, `border-style` or `width`.
- **<value>** Name of a style variable whose value needs to be inserted here. A list of all style variables you can find in the appendix.

Code example

13.2.16. The Languages-PIP

Purpose

The Languages-PIP is used for the installation of language files.

[1] http://www.w3.org/TR/REC-CSS2/selector.html

Programm 13.13 Exemplary structure of a `styleattributes.xml`

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://www.woltlab.com/DTDs/styleattributes.dtd">
<data>
  <import>
    <attribute>
      <selector><![CDATA[.infoContainer]]></selector>
      <name><![CDATA[background-color]]></name>
      <value><![CDATA[container1.background.color]]></value>
    </attribute>
    ...
  </import>
</data>
```

XML-Code in the `package.xml`

`<languages languagecode="de">de.xml</languages>` For every language installed, a new tag is used. The parameter languagecode specifies the language code of the respective language.

Tags and their meaning

<language> The tag contains a list of language categories (`<category>`). The attribute `languagecode` expects the correct specification of the language code according to ISO639-1-Standard[2].

· **<category>** The tag contains a list of language variables (`<item>`). The attribute `name` specifies the name of the language category. Categories are needed to sort the many language variables.

Like with package names, it is recommendable to use certain name spaces, e.g. “wcf.acp.global”. Categories can only be made up of a maximum of the blocks, separated by a dot (.). Each part needs to consist of at least one alphanumerical character[3] or an underscore (_).

· · **<item>** Each <item>-tag represents a custom language variable. Like with the categories, the attribute name provides information about the name of the variable. The name begins with the name of the language category. For the description, the same characters as for the categories can be used.

The tag contains the content of the language variable. This should be noted within the CDATA[4]-block. Furthermore, the following rules apply:

[2] http://en.wikipedia.org/wiki/ISO_639
[3] a until z, A until Z or 0 until 9
[4] http://en.wikipedia.org/wiki/CDATA

Mind the coding For the coding of language files, only UTF-8 is permitted. Make sure not to use any HTML-Entities, e.g. to write special characters. For this rule, there are only three exceptions:

```
&quot; used for "
&lt;   used for <
&gt;   used for >
```

Templatescripting allowed Within the language variables, templatescripting may be used.

Mind the context When using language variables, you need to pay attention to the context in which the variable is used. This applies to the templatescripting as well as to HTML-Entities. These are not valid in Javascript-instructions or e-mails. When templatescripting, you cannot fall back on variables that are unknown within the template. When variables are used directly through PHP, templatescripting is not permitted[5].

Code example

Programm 13.14 Exemplary structure of an english language file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE language SYSTEM "http://www.woltlab.com/DTDs/language.dtd">
<language languagecode="en">
  <category name="example.category.name">
    <item name="example.category.name.var1"><![CDATA[example content]]></item>
    <item name="example.category.name.var2"><![CDATA[example content]]></item>
    ...
  </category>
  ...
</language>
```

13.3. Other PIPs

13.3.1. The SQL-PIP

Purpose

The SQL-PIP is used to execute SQL-instructions during the installation of packages. In the typical use case, this will for example concern `CREATE TABLE` or `ALTER TABLE` commands. However, other SQL commands can be executed this way too.

[5] This will be changed in later versions

XML-Code in the `package.xml`

```
<sql>install.sql</sql>
```

Mode of operation

Before execution, the SQL-PIP checks if the respective commands are allowed. It is for example prohibited for a package to delete tables or table elements of another package.

The SQL-PIP logs all commands of the following type, so they can be reversed at deinstallation.

- `CREATE TABLE`
- `DROP TABLE`
- `CREATE INDEX`
- `DROP INDEX`
- `ALTER TABLE`
- `RENAME TABLE`

Warning: As you might have noticed, `INSERT INTO`, `DELETE FROM` and `UPDATE TABLE` commands are **not** logged by the SQL-PIP and are therefore irreversible.

13.3.2. The Script-PIP

Purpose

The Script-PIP is used to call up non-recurring and more complex operations during the installation.

XML-Code in the `package.xml`

```
<script>script.php</script>
```

Mode of operation

The call occurs within the class `ScriptPackageInstallationPlugin`. All variables used there are thus also possible in the Script-file.

Note: The PHP-Script needs to be installed first through the Files-PIP.

13.3.3. Das TemplatePatch-PIP

Purpose

The TemplatePatch-PIP is used to modify already existing templates via plugin. For this, a so-called *Unified Diff*[6] or *patch* is used. This *patch* only contains the changed rows of the template, not the complete template.

A *patch* like this is created with the command line program *diff*[7] and the command

`diff -u template.tpl.old template.tpl`.

XML-Code in the `package.xml`

```
<templatepatch>patch.diff</templatepatch>
```

Mode of operation

The TemplatePatch-PIP tries to apply the supplied patch to the templates by searching for the invidiaul parts of the patch, the so-called "hunks". The rows that are to be replaced and the surrounding context are then searched for (usually three rows of context).

Use of imprecise patches This normally works without any problems. However, in some cases it can happen the the template to be patched has already been changed by another plugin and therefore the correct place for the change cannot be located.

In case "only" the content of the rows to be replaced cannot be found, the TemplatePatch-PIP contains the so-called *Fuzz-Faktor*[8]. The fuzz-factor is a natural number[9] and determines the maximum number of rows that are ignored by the TemplatePatch-PIP when searching for the correct text passage. First, the TemplatePatch-PIP searches for a passage where all the lines exactly match. If this passage is not found and the fuzz-factor is equal to or larger than 1, the TemplatePatch-PIP repeats the search, this time ignoring the first and last row of the context. If this search also fails and the fuzz-factor is equal to or larger than 1, the TemplatePatch-PIP no ignores the first two and the last two rows of the context in another search.

[6] http://en.wikipedia.org/wiki/Diff
[7] http://www.gnu.org/software/diffutils/ alternatively
http://sourceforge.net/projects/unxutils for Windows
[8] see also http://www.gnu.org/software/diffutils/manual/html_node/Inexact.html
[9] the numbers 1, 2, 3, ... etc.

Note: The default value for the fuzz-factor is 2. It does not make sense to have a fuzz-factor larger than the size of the context (usually 3). The larger the fuzz-factor, the bigger the danger that the TemplatePatch-PIP chooses the wrong passage to patch.

To change the default value of the fuzz-factor, change in the parameter fuzzfactor in the `package.xml` in the tag <templatepatch>:
`<templatepatch fuzzfactor="3">patch.diff</templatepatch>`

If (independent of context) the passages to be replaced are not found, the patch cannot be applied at all. In this case, a manual processing of the template is required.

Repatching an update If during the update of package a new version of a template that has already been patches is delivered, the TemplatePatch-PIP automatically attempts to apply the patch to the new version of the template after the update. If his fails, a respective error message is displayed on the screen.

Note: The TemplatePatch-PIP only uses *Unified Diffs*, normal diffs or *Context Diffs* do not work.

Note: The TemplatePatch-PIP only handles templates in the standardtemplatepack. Modifications in custom templatepacks need to be made manually.

13.3.4. The ACPTemplatePatch-PIP

Purpose

The ACPTemplatePatch-PIP is used for the application of patches ACP-Templates.

XML-Code in the `package.xml`

```
<acptemplatepatch>patch.diff</acptemplatepatch>
```

Mode of operation

See TemplatePatch-PIP 13.3.3 on the preceding page.

13.4. Custom PIPs

It is possible to expand the system by your own PIPs if the existing PIPs do not cover the desired functionality. As mentioned in the introduction of the chapter, a PIP is mapped by a class that implements the interface `PackageInstallationPlugin`.

13.4.1. The Interface

The first step thus consists of creating a respective class that implements this interface. Some abstract classes that will be introduced later, already relieve you of a large part of the implementation. Firstly, the methods to be implemented are presented:

hasInstall

boolean **hasInstall** ()

`hasInstall()` Returns `true`, if the installation of the current plugin is supposed to execute this plugin. A verification of the tag used is conceivable here.

install

void **install** ()

`install()` Carries out the installation of the plugin.

hasUpdate

boolean **hasUpdate** ()

`hasUpdate()` Returns `true`, if the update of the current package is supposed to exectute this plugin. A verification of the tag used is conceivable here.

update

void **update** ()

`update()` Carries out the update of the plugin.

hasUninstall

boolean **hasUninstall** ()

`hasUninstall()` Returns `true`, if the deinstallation of the current plugin is supposed to execute this plugin. A verification of the tag used is conceivable here.

uninstall

void **uninstall** ()

`uninstall()` Carries out the de-installation of the plugin.

13.4.2. Abstract classes

Since the requirements of a PackageInstallationPlugin are often similar, the WCF already provides several abstract classes implementing some of the methods.

AbstractPackageInstallationPlugin

This is the default implementation of the interface. All six methods are implemented, while in this case mainly the triggering of the respective events happens. The `install()`-method is the only one that really needs to be overwritten to effectively implement a new functionality.

AbstractXMLPackageInstallationPlugin

For the many XML-based PIPs this abstract class is the starting point, it expands `AbstractPackageInstallationPlugin` by the following methods:

getXML *XML* **getXML** ()

`getXML()` reads the specified XML-Datei and saves the content in a XMLObject[10] which is returned.

getShowOrder *integer* **getShowOrder** (*integer* $showOrder, [*string* $parentName = null], [*string* $columnName = null], [*string* $tableNameExtension = ''])

`getShowOrder()` Returns an `showOrder`-value, which determines the position of an element within a level.

AbstractOptionPackageInstallationPlugin

This abstract class is used for the numerous Options-PIPs and in particular implements the `install()`-method, in which the XML-file is perambulated and single tags are read out. Especially important is the new abstract method `saveOption()`:

[10]internal the *SimpleXML* The class `XML` is defined in `wcf/lib/util/XML.class.php`.

saveOption *void* **saveOption** (`array` $option, `string` $categoryName, [`integer` $existingOptionID = 0])

saves an option with the respective properties in the database. The parameter `option` passes an array with values from the XML-file. `categoryName` indicates the name of the respective category. If the `optionID` of an option is already available, this can be passed through `existingOptionID`.

13.4.3. Installation of the PIP

The class now needs to be packed into a tar-archive and can be installed with the PIPs-PIP. More on this in chapter 13.1.5 on page 69.

14. Standalone applications

As already mentioned in the preceding chapters, standalone applications are special packages that provide their own graphic interfaces and are based on existing packages. Additionally it must be mentioned that standalone applications mandatorily have their own administrative section. Below, the individual steps are described that are necessary to create your own standalone application based on WCF.

14.1. Creating a package

First of all, of course a respective package needs to be created. You can find more on this in chapter 12 on page 61. In the `package.xml`, enter the following command in the `<packageinformation>`-block for an standalone application:

```
<standalone>1</standalone>
```

Any further specifiations are optional. For the required packages, you need to specify all packages that your standalone application depends upon.

Note: The dependency on `com.woltlab.wcf` only needs to be specified if the dependency is tied to a specific version. Otherwise, all packages are automatically dependent on that package.

14.2. Inheritance of the classes `WCF` and `WCFACP`

The structure of the WCF closely follows the *MVC-Pattern*[1]. Here, the class `WCF` (or `WCFACP` respectively) corresponds to the central control unit of the application (*Controller*). This can be used directly. However, you need to write your own inheritance of the class, e.g. to access your own cache-resources or templates. It is recommended to closely study the `WCF`-class and its method (the same goes for `WCFACP`).

[1] http://en.wikipedia.org/wiki/Model_View_Controller

14.3. Creating an IndexPage-class

The IndexPage-class is called by the RequestHandler by default if no parameters on the URL have been provided, i.e. this is the start page of your standalone application, or the entry page to the administrative section respectively.

Hints on the RequestHandler and the Page-interface you can find in chapter 11 on page 57.

14.4. Creating an index.php-file

Within the index.php-file, all important data is contained and the application is initialized. Some of the following commands can also be externalized to a separate file, which then needs to be embedded again. We will dispense with this here. The following code-example assumes that the standalone application is called com.application.test.

Programm 14.1 Example of an index.php-file

```
// initialize package array
$packageDirs = array();
// include config
require_once(dirname(__FILE__).'/config.inc.php');

// include WCF
require_once(RELATIVE_WCF_DIR.'global.php');
//
if (!count($packageDirs)) $packageDirs[] = TEST_DIR;
$packageDirs[] = WCF_DIR;

// starting test application
require_once(TEST_DIR.'lib/system/Test.class.php');
new Test();

RequestHandler::handle(ArrayUtil::appendSuffix($packageDirs, 'lib/'));
```

First, the array $packageDirs is initialized. This is then used in the embedded config.inc.php-file. The file is automatically started by the WCF and contains several constant definitions. Amongst others, the path to the directory of the standalone application is saved in an absolute term. For this, the last block is used. For com.application.test, this would be *test*. The constant name then reads RELATIVE_TEST_DIR and just TEST_DIR for the absolute path.

Note: In our example you would need to name own database tables within the install.sql-file of this package with the prefix *test1_1_*.

Next, the WCF is embedded and the array $packageDirs is filled with values. On the one hand, the path to the current standalone application is entered, on the other hand the path to the WCF-directory. The application can now be started. The class Test is derived from WCF.

Finally, the array `$packageDirs` is passed to the `RequestHandler`. The lib-directory is also attached to both path specifications. This contains the actual classes in the WCF as well as in the standalone application.

Part III.

Appendices

15. Events

15.1. Events of the free WCF-packages

Location of the Event	Name of the Event
com.woltlab.wcf.data.message.bbcode	
· URLParser	
parse()	didParse
	shouldParse
com.woltlab.wcf.page.util.menu	
· HeaderMenu	
loadCache()	loadCache
buildMenu()	buildMenu
com.woltlab.acp.package.plugin	
· StylePackageInstallationPlugin	
uninstall()	uninstall
com.woltlab.wcf.acp.form	
· UserSearchForm	
search()	buildConditions
com.woltlab.wcf.acp.package.plugin	
· AbstractPackageInstallationPlugin	
__construct()	construct
hasInstall()	hasInstall
install()	install
hasUpdate()	hasUpdate
update()	update
hasUninstall()	hasUninstall
uninstall()	uninstall
com.woltlab.wcf.action	
· AbstractAction	
readParameters()	readParameters
execute()	execute
executed()	executed

com.woltlab.wcf.page.util.menu	
· TreeMenu	
loadCache()	loadCache
buildMenu()	buildMenu
com.woltlab.wcf.page	
· AbstractForm	
submit()	submit
readFormParameters()	readFormParameters
validate()	validate
save()	save
saved()	saved
· AbstractPage	
readParameters()	readParameters
readData()	readData
assignVariables()	assignVariables
show()	show
· MultipleLinkPage	
calculateNumberOfPages()	calculateNumberOfPages
countItems()	countItems
· SortablePage	
validateSortField()	validateSortField
validateSortOrder()	validateSortOrder
com.woltlab.wcf.system.auth	
· UserAuth	
getInstance()	loadInstance
com.woltlab.wcf.system.session	
· SessionFactory	
get()	shouldInit
	didInit
com.woltlab.wcf.system.template	
· Template	
display()	shouldDisplay
	didDisplay

Table 15.1.: Events of the free WCF-packages

16. Style variables

This chapter describes all the variables that are permitted within the `variables.xml`-file. The variables are sorted in the same order that they later need to be processed in within the graphic interface of the administrative section. The stated default values apply to the style "'WoltLab Basic"'.

Please note that values that can be specified for individual variables can differ from those that need to be entered into the style editor.

When listing the variables, first of all the **name** is specified, followed by the **value** as used in the "'WoltLab Basic"'-style. It is therefore possible that no value is specified. In square brackets the *value margin* is given, which gives information on which specifications can be made here. All values need to be specified within a CDATA-block. The specification concluded by a short description of the variables.

16.1. Global

16.1.1. General

Display

page.alignment `center` [*align*] Allignment of the page – this command only works in connection with the next variable.

page.alignment.margin `margin-left:auto;margin-right:auto;` [*custom*] Alignment of the page (left: "'margin-left: auto; margin-right: 0"', right: "'margin-left: 0; marginright: auto"' and centered: "'margin-left: auto; margin-right: auto"').

page.width [*lengths*] Fixed page width (css-command `width`) for a static width.

page.width.max `80%` [*lengths*] Max. page width (css-command `max-width`) for a flexible width – should not be used at the same time as the static width.

page.width.min `760px` [*lengths*] Min. page width (css-command `min-width`) for a flexible width – should not be used at the same time as the static width.

Saving location for graphics

global.icons.location `icon/` [*path*] Path to the icon folder (this specification is currently ignored)

global.images.location `images/` [*path*] Path to the images folder (the best way is to place your pictures in a sub-folder of images)

Favorites-Icon

global.favicon `grey` [*favicon*] Name

16.1.2. Page

Page header

page.header.background.color `#777` [*bcolor*] Page color

page.header.height `90px` [*lengths*] Height

page.header.background.image [*burl*] Background image URL

page.header.background.image.alignment [*balign*] Background image alignment

page.header.background.image.repeat [*brepeat*] Repeat background image

Logo

page.logo.image `images/wbb3-header-logo.png` [*path*] Path to the image

page.logo.image.alignment `left` [*align*] Alignment

page.logo.image.padding.top `5px` [*lengths*] Inner padding (top)

page.logo.image.padding.right `0px` [*lengths*] Inner padding (right)

page.logo.image.padding.left `13px` [*lengths*] Inner padding (left)

Global title

global.title.hide `position: absolute; top: -9000px; left: -9000px;` [*hide*] Show global title

global.title.font [*font*] Font type

global.title.font.style [*custom*] Style (font-style) – the values normal, italic and oblique are possible.

global.title.font.weight [*custom*] Style (font-weight) – the values normal and bold are supported by all browsers.

global.title.font.size [*lengths*] Size

global.title.font.color [*color*] color

global.title.font.alignment [*align*] Alignment

global.title.font.padding.top [*lengths*] Inner padding (top)

global.title.font.padding.right [*lengths*] Inner padding (right)

global.title.font.padding.left [*lengths*] Inner padding (left)

Background

page.background.color `#fff` [*bcolor*] Background color

page.background.image [*burl*] Background image URL

page.background.image.attachment [*bfix*] Fix background image

page.background.image.alignment [*balign*] Background image alignment

page.background.image.repeat [*brepeat*] Repeat background image

16.1.3. Boxes

Box 1

container1.background.color `#f7f7f7` [*bcolor*] Background color

container1.font.color `#666` [*color*] Text color

container1.font.2nd.color `#888` [*color*] Second text color

container1.link.color `#666` [*color*] Link color

container1.link.color.hover `#333` [*color*] Link color (hover)

Box 2

container2.background.color `#efefef` [*bcolor*] Background color

container2.font.color `#666` [*color*] Text color

container2.font.2nd.color `#888` [*color*] Second text color

container2.link.color `#666` [*color*] Link color

container2.link.color.hover `#333` [*color*] Link color (hover)

Box 3

container3.background.color `#e0e0e0` [*bcolor*] Background color

container3.font.color `#333` [*color*] Text color

container3.font.2nd.color `#777` [*color*] Second text color

container3.link.color `#666` [*color*] Link color

container3.link.color.hover `#333` [*color*] Link color (hover)

16.1.4. Borders

Border heads

container.head.font.color `#fff` [*color*] Text color

container.head.font.2nd.color `#fff` [*color*] Second text color

container.head.link.color `#fff` [*color*] Link color

container.head.link.color.hover `#fff` [*color*] Link color (hover)

container.head.background.color `#777` [*bcolor*] Background color

container.head.background.image [*burl*] Background image URL

Borders

container.border.outer.width `1px` [*lengths*] Outer border width

container.border.outer.style `solid` [*style*] Outer border style

container.border.outer.color `#999` [*color*] Outer border color

container.border.inner.color `#fff` [*color*] Inner border color

divider.width `1px` [*lengths*] Border width (dividing line)

divider.style `solid` [*style*] Border style (dividing line)

divider.color `#bbb` [*color*] Border color (dividing line)

16.1.5. Forms

Text

input.font `'Trebuchet MS', Arial, sans-serif` [*font*] Font type

input.font.size `.85em` [*lengths*] Font size

input.font.color `#333` [*color*] Font color

input.font.color.focus `#000` [*color*] Font color (focus)

Background

input.background.color `#fff` [*bcolor*] Background color

input.background.color.focus `#ffd` [*bcolor*] Background color (focus)

Border

input.border.width `1px` [*lengths*] Border width

input.border.style `solid` [*style*] Border style

input.border.color `#999` [*color*] Border color

input.border.color.focus `#08f` [*color*] Border color (focus)

16.2. Text

16.2.1. Text types

Texts

page.font `'Trebuchet MS', Arial, sans-serif` [*font*] Font type

page.font.size `.8em` [*lengths*] Text size

page.font.2nd.size `.85em` [*lengths*] Second text size

page.font.line.height `1.5` [*lengths*] Line height

page.font.color `#333` [*color*] Text color

page.font.2nd.color `#888` [*color*] Second text color

Heading

page.title.font `'Trebuchet MS', Arial, sans-serif` [*font*] Font type

page.title.font.style `normal` [*custom*] Style (font-style) – the values normal, italic and oblique are possible.

page.title.font.weight `normal` [*custom*] Style (font-weight) – the values normal and bold are supported by all browsers.

page.title.font.size `1.3em` [*lengths*] Text size

page.title.font.color `#333` [*color*] Text color

16.2.2. Links

Links

page.link.color `#666` [*color*] Link color

page.link.color.hover `#333` [*color*] Link color (hover)

External links

page.link.external.color `#333` [*color*] External link color

page.link.external.color.hover `#08f` [*color*] External link color (hover)

Active links

page.link.color.active `#08f` [*color*] Link color (active)

16.3. Buttons

16.3.1. Small Buttons

Labeling

buttons.small.caption.hide [*hide*] Show labeling of the buttons

buttons.small.caption.color `#666` [*color*] Text color

buttons.small.caption.color.hover `#333` [*color*] Text color (hover)

Outer border

buttons.small.border.outer.width `1px` [*lengths*] Outer border width

buttons.small.border.outer.style `solid` [*style*] Outer border style

buttons.small.border.outer.color `#999` [*color*] Outer border color

buttons.small.border.outer.color.hover `#999` [*color*] Outer border color (hover)

Inner border

buttons.small.border.inner.width `1px` [*lengths*] Inner border width

buttons.small.border.inner.style `solid` [*style*] Inner border style

buttons.small.border.inner.color `#fff` [*color*] Inner border color

buttons.small.border.inner.color.hover `#fff` [*color*] Inner border color (hover)

Background color

buttons.small.background.color `#e8e8e8` [*bcolor*] Background color

buttons.small.background.color.hover `#fff` [*bcolor*] Background color (hover)

Background image

buttons.small.background.image [*burl*] Background image URL

buttons.small.background.image.hover [*burl*] Background image URL (hover)

16.3.2. Large Buttons

Labeling

buttons.large.caption.hide [*hide*] Show labeling of the buttons

buttons.large.caption.color `#fff` [*color*] Text color

buttons.large.caption.color.hover `#333` [*color*] Text color (hover)

Outer border

buttons.large.border.outer.width `1px` [*lengths*] Outer border width

buttons.large.border.outer.style `solid` [*style*] Outer border style

buttons.large.border.outer.color `#999` [*color*] Outer border color

buttons.large.border.outer.color.hover `#999` [*color*] Outer border color (hover)

Inner border

buttons.large.border.inner.width `1px` [*lengths*] Inner border width

buttons.large.border.inner.style `solid` [*style*] Inner border style

buttons.large.border.inner.color `#fff` [*color*] Inner border color

buttons.large.border.inner.color.hover `#fff` [*color*] Inner border color (hover)

Background color

buttons.large.background.color `#777` [*bcolor*] Background color

buttons.large.background.color.hover `#cecece` [*bcolor*] Background color (hover)

Background image

buttons.large.background.image [*burl*] Background image URL

buttons.large.background.image.hover [*burl*] Background image URL (hover)

16.4. Menus

16.4.1. Main Menu

Buttons

menu.main.position `text-align:left;margin:0 auto 0 0` [*custom*] Alignment of the buttons (left: "'text-align: left; margin: 0 auto 0 0"', right: "'text-align: right; margin: 0 auto 0 0"' and centered: "'textalign: center; margin: 0 auto 0 auto;"')

menu.main.bar.hide `#f7f7f7` [*bcolor*] Background color

menu.main.bar.show `#f7f7f7` [*bcolor*] Background color

menu.main.bar.divider.show `1px` [*custom*] Should a dividing line be displayed (yes: "'1px"' and no: "'0"')

Caption

menu.main.caption.hide [*hide*] If the caption should not be displayed, then "'position: absolute; top: -9000px; left: -9000px;"', otherwise leave clear.

menu.main.caption.color `#666` [*color*] Text color

menu.main.caption.color.hover `#333` [*color*] Text color (hover)

menu.main.active.caption.color `#fff` [*color*] Text color (active)

menu.main.active.caption.color.hover `#000` [*color*] Text color (active hover)

Background color

menu.main.background.color `#efefef` [*bcolor*] Background color

menu.main.background.color.hover `#fff` [*bcolor*] Background color (hover)

menu.main.active.background.color `#777` [*burl*] Background color (active)

menu.main.active.background.color.hover `#cecece` [*burl*] Background color (active hover)

Background image

menu.main.background.image [*burl*] Background image URL

menu.main.background.image.hover [*burl*] Background image URL (hover)

16.4.2. Tabs

Caption

menu.tab.caption.color `#666` [*color*] Text color

menu.tab.caption.color.hover `#333` [*color*] Text color (hover)

menu.tab.active.caption.color [*color*] Text color (active)

menu.tab.active.caption.color.hover [*color*] Text color (active hover)

Background color

menu.tab.background.color `#e8e8e8` [*bcolor*] Background color

menu.tab.background.color.hover `#fff` [*bcolor*] Background color (hover)

menu.tab.active.background.color [*bcolor*] Background color (active)

menu.tab.active.background.color.hover [*bcolor*] Background color (active hover)

Background image

menu.tab.background.image [*burl*] Background image URL

menu.tab.background.image.hover [*burl*] Background image URL (hover)

16.4.3. Tab-Buttons

Caption

menu.tab.button.caption.color `#ddd` [*color*] Text color

menu.tab.button.caption.color.hover `#fff` [*color*] Text color (hover)

menu.tab.button.active.caption.color `#fff` [*color*] Text color (active)

menu.tab.button.active.caption.color.hover `#fff` [*color*] Text color (active hover)

Background color

menu.tab.button.background.color [*bcolor*] Background color

menu.tab.button.background.color.hover `#666` [*bcolor*] Background color (hover)

menu.tab.button.active.background.color `#444` [*bcolor*] Background color (active)

menu.tab.button.active.background.color.hover `#666` [*bcolor*] Background color (active hover)

Outer border

menu.tab.button.border.style `solid` [*style*] Border style

menu.tab.button.border.color `#aaa` [*color*] Border color

menu.tab.button.border.color.hover `#bbb` [*color*] Border color (hover)

16.4.4. Table heads

Caption

table.head.caption.color `#666` [*color*] Text color

table.head.caption.color.hover `#333` [*color*] Text color (hover)

table.head.active.caption.color `#333` [*color*] Text color (active)

table.head.active.caption.color.hover `#333` [*color*] Text color (active hover)

Underline

table.head.border.bottom.style `solid` [*style*] Style

table.head.border.bottom.color `#999` [*color*] Border color

table.head.border.bottom.color.hover `#999` [*color*] Border color (hover)

table.head.active.border.bottom.color `#08f` [*color*] Border color (active)

table.head.active.border.bottom.color.hover `#08f` [*color*] Border color (active hover)

Background color

table.head.background.color `#cecece` [*bcolor*] Background color

table.head.background.color.hover `#e8e8e8` [*bcolor*] Background color (hover)

table.head.active.background.color `#e8e8e8` [*bcolor*] Background color (active)

table.head.active.background.color.hover `#efefef` [*bcolor*] Background color (active hover)

Background image

table.head.background.image [*burl*] Background image URL

table.head.background.image.hover [*burl*] Background image URL (hover)

16.4.5. Extras

Dropdown- & List-menus

menu.dropdown.link.color `#555` [*color*] Text color

menu.dropdown.link.color.hover `#000` [*color*] Text color (hover)

menu.dropdown.background.color `#f7f7f7` [*bcolor*] Background color

menu.dropdown.background.color.hover `#e0e0e0` [*bcolor*] Background color (hover)

menu.dropdown.background.image [*burl*] Background image URL

menu.dropdown.background.image.hover [*burl*] Background image URL (hover)

Selected Elements

selection.font.color `#333` [*color*] Text color

selection.font.2nd.color `#333` [*color*] Second Text color

selection.link.color `#666` [*color*] Link color

selection.background.color `#def` [*bcolor*] Background color

selection.border.width `1px` [*lengths*] Border width

selection.border.style `solid` [*style*] Border style

selection.border.color `#08f` [*bcolor*] Border color

selection.background.image [*burl*] Background image URL

16.5. Advanced

16.5.1. Message display

Sidebars

messages.sidebar.alignment `left` [*custom*] Alignment(left: "'left"', rechts: "'right"' or top: "'top"')

messages.sidebar.text.alignment `center` [*align*] Text alignment

messages.sidebar.avatar.framed `0` [*boolean*] Framed view

messages.sidebar.color.cycle `0` [*boolean*] Alternate background color

messages.sidebar.divider.use `1` [*boolean*] Divider line

Message field

messages.framed `0` [*boolean*] Framed view

messages.color.cycle `1` [*boolean*] Alternate background color

messages.boxes.background.color [*bcolor*] Background color (boxes)

messages.boxes.font.color [*color*] Text color (boxes)

messages.footer.alignment `right` [*custom*] Alignment of the buttons (left: "'left"' or right: "'right"')

16.5.2. Additional CSS-declarations

Additional CSS-declarations 1

user.additional.style.input1 [*custom*] Custom CSS-Code 1

user.additional.style.input1.use `0` [*boolean*] Should custom CSS-Code 1 be used

Additional CSS-declarations 2

user.additional.style.input2 [*custom*] Custom CSS-Code 2

user.additional.style.input2.use `0` [*boolean*] Should custom CSS-Code 2 be used

16.5.3. Comments

Comments

user.comment `"WoltLab Basic" [...]` [*custom*] Comment on the style

16.6. Value margins

Within the listing of the style variables, the value margin for these specifications has been stated. These will now be explained in alphabetical order. Please note that it is often possible to not make any specification and still achieve the desired results – because the principle of heredity plays an important part in CSS. In some cases, it is also possible that a specification is ineffective because it is not supported by the browser or overwritten in another location.

align

This means the alignment of the text. Browsers such as the Internet Explorer also align texts with `text-align`:

`left` Align left

`center` Align center

`right` Align right

`justify` Justification - should be used with care and does not work for the alignment of elements.

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap16.html#heading-16.2%A0

balign

For the alignment of background images, combine the specifications for the vertical alignment (top-bottom) with those of the horizontal alignment:

`top` Align top

`bottom` Align bottom

`center` Align center – can be used for horizontal as well as for vertical alignment.

`left` Align left

`right` Align right

Always specify the value for the vertical alignment first and the horizontal one afterwards, e.g. `top left` to align an image at the top left.

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap14.html#heading-14.2.1%A0

bcolor

The background color can be controlled with the color specifications described in `color`. Additionally, the keyword `transparent` is available to make the background translucent.

bfix

To fix a background image, enter the value `fixed`. Please note that a fixed background can slow down the operation of the page (.e.g when scrolling).

boolean

Enter a `1` for `true` and a `0` for `false`.

brepeat

Background images can also be repeated (tiled). This can be controlled through the values below:

`repeat` Image is repeated in all directions.

`repeat-x` Image is repeated in all x-direction (left-right).

`repeat-y` Image is repeated in all y-direction (up-down).

`no-repeat` Image is displayed once, but not repeated.

burl

The URL of background images is specified through the key word `url()` . The brackets contain the path to the image file.

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.4%A0

color

Colors can be specified as hexadecimal numbers or in RGB-notation. Addtionally, a list with key words such as red, green or blue is available. The color red can thus be specified like this:

```
#f00
#ff0000
rgb(255,0,0)
rgb(100%, 0%, 0%)
red
```

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.6%A0

custom

All variables that need their own, very particular specification, have been declared as `custom`. Here, you find a fitting description with the respective style variable.

favicon

When specifying your Favorites Icon, you can specify a name from the list below. These are ready-made Favicons:

- black
- blue
- blueExtra
- brown
- brownExtra
- darkBlue
- darkGreen
- darkRed
- darkViolet
- green
- greenExtra
- grey
- greyExtra
- lightBlue
- lightGreen
- lightGrey
- ochre
- orange
- pink
- red
- redExtra
- violet
- violetExtra
- yellow

If you would like to use your own Favicon, leave this variable empty and overwrite the file `favicon.ico` in the folder of the end application.

font

There are two ways for the specification of the font type:

1. Name of the font - this name should be put in quotation marks. Make sure only to use fonts that a reinstalled on a majority of computers. Exotic font types might work on your own computer, but not on the ones of your users.
2. Specification of a generic family: `serif`, `sans-serif`, `cursive`, `fantasy` and `monospace`

Several font types need to be separated by commas. It is recommended to speficy a generic font type last.

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap15.html#heading-15.2.2%A0

hide

Certain variables exist to hide elements. If the variable is empty, the element is displayed. To hide it, please use the following code:

```
position:  absolute; top:  -9000px; left:  -9000px;
```

lengths

For the specification of heights, lengths and widths, the units `em`, `ex`, `px` and `%` are available to you. The use of absolute sizes such as `pt` or `cm` is not recommended.

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap04.html#heading-4.3.2%A0

path

Path specifications to files and folders should be made relative to the WCF-folder.

style

Borders are changeable in their style. Please be aware that not all browsers supported the styles below:

- none
- hidden
- dotted
- dashed
- solid
- double
- groove
- ridge
- inset
- outset

Further information: http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap08.html#heading-8.5.3%A0

Index