

# PCPIN Template API (Application programming interface)



# Contents

<a href="#">Package PcpinTpl Procedural Elements</a>	2
<a href="#">pcpintpl.class.php</a>	2
<a href="#">Define PCPIN_TPL_DEFAULT_PARSE_MODE</a>	2
<a href="#">Define PCPIN_TPL_FULL_NS</a>	2
<a href="#">Define PCPIN_TPL_NAME_MAIN</a>	2
<a href="#">Define PCPIN_TPL_NAME_SUB</a>	3
<a href="#">Define PCPIN_TPL_NS</a>	3
<a href="#">Define PCPIN_TPL_PARSE_MODE_APPEND</a>	3
<a href="#">Define PCPIN_TPL_PARSE_MODE_OVERWRITE</a>	3
<a href="#">Define PCPIN_TPL_TRIM_ROOT</a>	3
<a href="#">Package PcpinTpl Classes</a>	4
<a href="#">Class PcpinTpl</a>	4
<a href="#">Var \$basedir</a>	4
<a href="#">Var \$files</a>	4
<a href="#">Var \$global_vars</a>	4
<a href="#">Var \$last_error</a>	5
<a href="#">Var \$parsed_name_flags</a>	5
<a href="#">Var \$parsed_name_ref</a>	5
<a href="#">Var \$sub_name_ref</a>	5
<a href="#">Var \$tpl_depth</a>	5
<a href="#">Var \$tpl_name_ref</a>	6
<a href="#">Var \$tpl_struct</a>	6
<a href="#">Var \$tpl_vars</a>	6
<a href="#">Var \$tpl_vars_plain</a>	6
<a href="#">Constructor PcpinTpl</a>	6
<a href="#">Method addGlobalVar</a>	6
<a href="#">Method addGlobalVars</a>	7
<a href="#">Method addVar</a>	7
<a href="#">Method addVars</a>	7
<a href="#">Method characterData</a>	8
<a href="#">Method clearTemplate</a>	8
<a href="#">Method displayParsedTemplate</a>	8
<a href="#">Method endElement</a>	9
<a href="#">Method getFileContents</a>	9
<a href="#">Method getLastError</a>	9
<a href="#">Method getLastFileData</a>	9
<a href="#">Method getLineNumber</a>	10
<a href="#">Method getParsedTemplate</a>	10
<a href="#">Method makeRefs</a>	10
<a href="#">Method parseIntoString</a>	10
<a href="#">Method parseIntoStruct</a>	11
<a href="#">Method parseTemplate</a>	11
<a href="#">Method passVars</a>	12

<a href="#">Method readTemplatesFromFile</a>	12
<a href="#">Method resetTpl</a>	12
<a href="#">Method setBasedir</a>	12
<a href="#">Method setError</a>	13
<a href="#">Method startElement</a>	13
<a href="#">Appendices</a>	14
<a href="#">Appendix A - Class Trees</a>	15
<a href="#">PcpinTpl</a>	15
<a href="#">Appendix C - Source Code</a>	16
<a href="#">Package PcpinTpl</a>	17
<a href="#">source code: pcpintpl.class.php</a>	18



# Package PcpinTpl Procedural Elements

## pcpintpl.class.php

PCPIN Template engine Copyright (C) 2007 Kanstantin Reznichak

**This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.**

PCPIN Template engine Copyright (C) 2007 Kanstantin Reznichak

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

- **Package** PcpinTpl
- **Filesource** [Source Code for this file](#)

PCPIN\_TPL\_DEFAULT\_PARSE\_MODE = PCPIN\_TPL\_PARSE\_MODE\_OVERWRITE [[line 65](#)]

### **Default parse mode**

Default parse mode

PCPIN\_TPL\_FULL\_NS = (PCPIN\_TPL\_NS!="")?(PCPIN\_TPL\_NS.':'):" [[line 84](#)]

### **PCPIN template elements namespace full description**

PCPIN template elements namespace full description

PCPIN\_TPL\_NAME\_MAIN = 'TPL' [[line 45](#)]

### **Main template element name**

Main template element name

PCPIN\_TPL\_NAME\_SUB = 'SUB' [[line 50](#)]

**Subtemplate element name**

Subtemplate element name

PCPIN\_TPL\_NS = 'PCPIN' [[line 40](#)]

**PCPIN template elements namespace**

PCPIN template elements namespace

PCPIN\_TPL\_PARSE\_MODE\_APPEND = 'a' [[line 60](#)]

**Parse mode "append" identifier**

Parse mode "append" identifier

PCPIN\_TPL\_PARSE\_MODE\_OVERWRITE = 'w' [[line 55](#)]

**Parse mode "overwrite" identifier**

Parse mode "overwrite" identifier

PCPIN\_TPL\_TRIM\_ROOT = true [[line 70](#)]

**Wether to delete line break between root element and the next line and between the last line and root element's closing tag**

Wether to delete line break between root element and the next line and between the last line and root element's closing tag

# Package PcpinTpl Classes

## Class PcpinTpl

[line [91](#)]

**Class PcpinTpl**  
Class PcpinTpl

- **Package** PcpinTpl

### PcpinTpl::\$basedir

*string* = " [line [103](#)]

**The directory where the template files are stored**  
The directory where the template files are stored

### PcpinTpl::\$files

*array* = null [line [109](#)]

**Array with opened files' description (name=>byte\_offset)**  
Array with opened files' description (name=>byte\_offset)

### PcpinTpl::\$global\_vars

*array* = null [line [171](#)]

### Global variables.

Global variables. These variables are overridden by template variables with the same name.

#### PcpinTpl::\$last\_error

*string* = " [line [97](#)]

#### Last raised error description

Last raised error description

#### PcpinTpl::\$parsed\_name\_flags

*array* = null [line [152](#)]

**Parsed template flags (true, if the template is parsed) The templates are referenced by the value of an attribute "name".**

Parsed template flags (true, if the template is parsed) The templates are referenced by the value of an attribute "name". NOTE: if there are multiple templates with the same, then only the last template will be referenced

#### PcpinTpl::\$parsed\_name\_ref

*array* = null [line [144](#)]

**Parsed template contents The templates are referenced by the value of an attribute "name".**

Parsed template contents The templates are referenced by the value of an attribute "name". NOTE: if there are multiple templates with the same, then only the last template will be referenced

#### PcpinTpl::\$sub\_name\_ref

*array* = null [line [136](#)]

#### Array with references to the subtemplates.

Array with references to the subtemplates. The subtemplates are referenced by the value of an attribute "name" of their parent template.

#### PcpinTpl::\$tpl\_depth

*int* = 0 [line [121](#)]



## Template structure current depth

Template structure current depth

### PcpinTpl::\$tpl\_name\_ref

*array* = null [[line 129](#)]

### Array with references to the templates.

Array with references to the templates. The templates are referenced by the value of an attribute "name". NOTE: if there are multiple templates with the same, then only the last template will be referenced

### PcpinTpl::\$tpl\_struct

*array* = null [[line 115](#)]

## Template structure

Template structure

### PcpinTpl::\$tpl\_vars

*array* = null [[line 158](#)]

### Template variables referenced by the template name.

Template variables referenced by the template name.

### PcpinTpl::\$tpl\_vars\_plain

*array* = null [[line 164](#)]

### All variable names used in all loaded templates

All variable names used in all loaded templates

Constructor *void* function PcpinTpl::PcpinTpl() [[line 181](#)]

### Constructor

Constructor

*void* function PcpinTpl::addGlobalVar([\$var\_name = "], [\$var\_val = null]) [[line 874](#)]

### Function Parameters:

- *string* **\$var\_name** Variable name
- *mixed* **\$var\_val** Variable value

### Add a global variable

Add a global variable

*void* function PcpinTpl::addGlobalVars([\$vars = null]) [*line* [887](#)]

#### **Function Parameters:**

- *array* **\$vars** Variables to add

### Add multiple global variables

Add multiple global variables The \$vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)

*void* function PcpinTpl::addVar([\$template = "], [\$var\_name = "], [\$var\_val = null]) [*line* [846](#)]

#### **Function Parameters:**

- *string* **\$template** Template name
- *string* **\$var\_name** Variable name
- *mixed* **\$var\_val** Variable value

### Add a variable to the template

Add a variable to the template

*void* function PcpinTpl::addVars([\$template = "], [\$vars = null]) [*line* [860](#)]

#### **Function Parameters:**

- *string* **\$template** Template name
- *array* **\$vars** Variables to add

### Add multiple variables to the template.

Add multiple variables to the template. The \$vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)

*boolean* function PcpinTpl::characterData(&\$tpl\_struct, &\$tpl\_depth, [\$cdata = "], \$tpl\_struct, \$tpl\_depth) [*line* [668](#)]

#### **Function Parameters:**

- *array* **\$tpl\_struct** A reference to the template structure array
- *array* **\$tpl\_depth** A reference to the current template structure depth
- *string* **\$cdata** Character data
- **&\$tpl\_struct**
- **&\$tpl\_depth**

### Characted data handler

Characted data handler

*void* function PcpinTpl::clearTemplate([\$name = "]) [*line* [1088](#)]

#### **Function Parameters:**

- *string* **\$name** Template name

### Clear template variables and parsed contents of a template

Clear template variables and parsed contents of a template

*void* function PcpinTpl::displayParsedTemplate([\$name = "]) [*line* [1079](#)]

#### **Function Parameters:**

- *string* **\$name** Template name

### Display parsed template.

Display parsed template. If the template is not parsed yet, then it will be parsed.

*boolean* function PcpinTpl::endElement(&\$tpl\_struct, &\$tpl\_depth, \$name, \$tpl\_struct, \$tpl\_depth) [*line* [694](#)]  
**Function Parameters:**

- *array* **\$tpl\_struct** A reference to the template structure array
- *array* **\$tpl\_depth** A reference to the current template structure depth
- *string* **\$name** Element name
- **&\$tpl\_struct**
- **&\$tpl\_depth**

### **End element handler**

End element handler

*mixed* function PcpinTpl::getFileContents([\$file = ""]) [*line* [284](#)]  
**Function Parameters:**

- *string* **\$file** File name (relative to the base directory)

### **Read file into a string**

Read file into a string

*string* function PcpinTpl::getLastError() [*line* [217](#)]

### **Get last raised error description**

Get last raised error description

*void* function PcpinTpl::getLastFileData(&\$filename, &\$offset, \$filename, \$offset) [*line* [780](#)]  
**Function Parameters:**

- *string* **\$filename** A reference to the variable where file name will be stored
- *array* **\$offset** A reference to the variable where byte offset will be stored
- **&\$filename**
- **&\$offset**

**returns name and byte offset of the current opened file**  
returns name and byte offset of the current opened file

*int* function PcpinTpl::getLineNumber([\$filename = "], [\$char = 0]) [[line 824](#)]  
**Function Parameters:**

- *string* **\$filename** File name to search in
- *int* **\$char** Character offset

**Get number of line at which the character with specified offset is located**  
Get number of line at which the character with specified offset is located

*string* function PcpinTpl::getParsedTemplate([\$name = "]) [[line 918](#)]  
**Function Parameters:**

- *string* **\$name** Template name

**Parse template (if not parsed yet) and return it's parsed contents as string**  
Parse template (if not parsed yet) and return it's parsed contents as string

*void* function PcpinTpl::makeRefs(&\$root, [\$parent\_tpl\_name = "]) [[line 795](#)]  
**Function Parameters:**

- *array* **&\$root** Element to start with
- **\$parent\_tpl\_name**

**Create new name reference arrays**  
Create new name reference arrays

*string* function PcpinTpl::parseIntoString(\$tpl, \$vars) [[line 944](#)]  
**Function Parameters:**

- *array* **\$tpl** Template record
- *array* **\$vars** Template variables

### Parse template structure and return it's parsed contents as a string

Parse template structure and return it's parsed contents as a string

*boolean* function PcpinTpl::parseIntoStruct(&\$tpl\_struct, &\$tpl\_depth, &\$cdata\_prefix, &\$cdata\_postfix, &\$tpl, \$tpl\_struct, \$tpl\_depth, \$cdata\_prefix, \$cdata\_postfix, \$tpl) [*line* [347](#)]

#### **Function Parameters:**

- *array* **\$tpl\_struct** A reference to the template structure array
- *array* **\$tpl\_depth** A reference to the current template structure depth
- *array* **\$cdata\_prefix** A reference to the variable where CDATA between offset 0 and first element will be stored
- *array* **\$cdata\_postfix** A reference to the variable where CDATA between last element and end of the file will be stored
- *string* **\$tpl** A reference to the template string
- **&\$tpl\_struct**
- **&\$tpl\_depth**
- **&\$cdata\_prefix**
- **&\$cdata\_postfix**
- **&\$tpl**

### Parse template string into the internal structure

Parse template string into the internal structure

*void* function PcpinTpl::parseTemplate([\$name = ""], [\$mode = PCPIN\_TPL\_DEFAULT\_PARSE\_MODE]) [*line* [901](#)]

#### **Function Parameters:**

- *string* **\$name** Template name

- *string* **\$mode** Parse mode

## Parse template

Parse template

*string* function PcpinTpl::passVars([\$parsed = "], [\$vars = null]) [[line 1038](#)]

**Function Parameters:**

- *string* **\$parsed** Parsed template string
- *array* **\$vars** Variables to pass

## Pass variables to the parsed template string.

Pass variables to the parsed template string. Global variables will be passed too.

*boolean* function PcpinTpl::readTemplatesFromFile([\$file = "]) [[line 257](#)]

**Function Parameters:**

- *string* **\$file** File name (relative to the base directory)

## Read template file and parses contained templates

Read template file and parses contained templates

*void* function PcpinTpl::resetTpl() [[line 190](#)]

## Reset template structure

Reset template structure

*boolean* function PcpinTpl::setBasedir([\$dir = "]) [[line 227](#)]

**Function Parameters:**

- *string* **\$dir** Base directory

## Set new base directory

## Set new base directory

*void* function PcpinTpl::setError([\$errortext = ""]) [[line 208](#)]

### **Function Parameters:**

- *string* **\$errortext** Error text

## Set error status

Set error status

*boolean* function PcpinTpl::startElement(&\$tpl\_struct, &\$tpl\_depth, \$name, \$attrs, \$tpl\_struct, \$tpl\_depth) [[line 497](#)]

### **Function Parameters:**

- *array* **\$tpl\_struct** A reference to the template structure array
- *array* **\$tpl\_depth** A reference to the current template structure depth
- *string* **\$name** Element name
- *array* **\$attrs** Element attributes
- **&\$tpl\_struct**
- **&\$tpl\_depth**

## Start element handler

Start element handler



# Appendices

# Appendix A - Class Trees

## Package PcpinTpl

### PcpinTpl

- [PcpinTpl](#)

# Appendix C - Source Code

## Package PcpinTpl

# File Source for pcpintpl.class.php

Documentation for this file is available at [pcpinTpl.class.php](http://pcpinTpl.class.php)

```
1  <?php
2  /**
3   *   PCPIN Template engine
4   *   Copyright (C) 2007  Kanstantin Reznichak
5   *
6   *   This program is free software: you can redistribute it and/or modify
7   *   it under the terms of the GNU General Public License as published by
8   *   the Free Software Foundation, either version 3 of the License, or
9   *   (at your option) any later version.
10  *
11  *   This program is distributed in the hope that it will be useful,
12  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
13  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  *   GNU General Public License for more details.
15  *
16  *   You should have received a copy of the GNU General Public License
17  *   along with this program.  If not, see <http://www.gnu.org/licenses/>.
18  */
19
20
21 /**
22  * PCPIN Template engine
23  * @package PcpinTpl
24  * @author Konstantin Reznichak <k.reznichak@pcpin.com>
25  * @copyright Copyright &copy; 2007, Kanstantin Reznichak
26  * @version 1.0
27  */
28
29
30
31 /*****
32  *
33  * CONFIGURATION
34  *
35  *****/
36
37 /**
38  * PCPIN template elements namespace
39  */
40 define('PCPIN_TPL_NS', 'PCPIN');
41
42 /**
43  * Main template element name
44  */
45 define('PCPIN_TPL_NAME_MAIN', 'TPL');
46
47 /**
48  * Subtemplate element name
49  */
50 define('PCPIN_TPL_NAME_SUB', 'SUB');
51
52 /**
53  * Parse mode "overwrite" identifier
54  */
55 define('PCPIN_TPL_PARSE_MODE_OVERWRITE', 'w');
56
57 /**
58  * Parse mode "append" identifier
59  */
60 define('PCPIN_TPL_PARSE_MODE_APPEND', 'a');
61
62 /**
63  * Default parse mode
64  */
65 define('PCPIN_TPL_DEFAULT_PARSE_MODE', PCPIN_TPL_PARSE_MODE_OVERWRITE);
66
67 /**
```

```

68  * Wether to delete line break between root element and the next line and between the last line and
root element's closing tag
69  */
70  define('PCPIN_TPL_TRIM_ROOT', true);
71
72
73
74  /*****
75  *
76  * CONFIGURATION ENDS HERE
77  * DO NOT EDIT ANYTHING BELOW UNTIL YOU ARE EXACTLY KNOW WHAT YOU ARE DOING!!!
78  *
79  *****/
80
81  /**
82  * PCPIN template elements namespace full description
83  */
84  define('PCPIN_TPL_FULL_NS', (PCPIN_TPL_NS!='')? (PCPIN_TPL_NS.':') : '');
85
86
87  /**
88  * Class PcpinTpl
89  * @package PcpinTpl
90  */
91  class PcpinTpl {
92
93      /**
94       * Last raised error description
95       * @var string
96       */
97      var $last_error='';
98
99      /**
100       * The directory where the template files are stored
101       * @var string
102       */
103      var $basedir='';
104
105      /**
106       * Array with opened files' description (name=>byte_offset)
107       * @var array
108       */
109      var $files=null;
110
111      /**
112       * Template structure
113       * @var array
114       */
115      var $tpl_struct=null;
116
117      /**
118       * Template structure current depth
119       * @var int
120       */
121      var $tpl_depth=0;
122
123      /**
124       * Array with references to the templates.
125       * The templates are referenced by the value of an attribute "name".
126       * NOTE: if there are multiple templates with the same, then only the last template will be
referenced
127       * @var array
128       */
129      var $tpl_name_ref=null;
130
131      /**
132       * Array with references to the subtemplates.
133       * The subtemplates are referenced by the value of an attribute "name" of their parent
template.
134       * @var array
135       */
136      var $sub_name_ref=null;
137
138      /**
139       * Parsed template contents
140       * The templates are referenced by the value of an attribute "name".
141       * NOTE: if there are multiple templates with the same, then only the last template will be
referenced
142       * @var array
143       */

```

```

144     var $parsed_name_ref=null;
145
146     /**
147      * Parsed template flags (true, if the template is parsed)
148      * The templates are referenced by the value of an attribute "name".
149      * NOTE: if there are multiple templates with the same, then only the last template will be
referenced
150      * @var array
151      */
152     var $parsed_name_flags=null;
153
154     /**
155      * Template variables referenced by the template name.
156      * @var array
157      */
158     var $tpl_vars=null;
159
160     /**
161      * All variable names used in all loaded templates
162      * @var array
163      */
164     var $tpl_vars_plain=null;
165
166     /**
167      * Global variables.
168      * These variables are overridden by template variables with the same name.
169      * @var array
170      */
171     var $global_vars=null;
172
173
174
175
176
177
178     /**
179      * Constructor
180      */
181     function PcpinTpl() {
182         // Reset template structure
183         $this-> resetTpl();
184     }
185
186
187     /**
188      * Reset template structure
189      */
190     function resetTpl() {
191         $this-> tpl_struct=array();
192         $this-> tpl_depth=0;
193         $this-> files=array();
194         $this-> tpl_name_ref=array();
195         $this-> sub_name_ref=array();
196         $this-> parsed_name_ref=array();
197         $this-> parsed_name_flags=array();
198         $this-> tpl_vars=array();
199         $this-> tpl_vars_plain=array();
200         $this-> global_vars=array();
201     }
202
203
204     /**
205      * Set error status
206      * @param string $errortext Error text
207      */
208     function setError($errortext='') {
209         $this-> last_error=$errortext;
210     }
211
212
213     /**
214      * Get last raised error description
215      * @return string
216      */
217     function getLastError() {
218         return $this-> last_error;
219     }
220
221
222     /**

```

```

223 * Set new base directory
224 * @param string $dir Base directory
225 * @return boolean TRUE, if directory exists and readable, otherwise FALSE
226 */
227 function setBasedir($dir='') {
228     $result=false;
229     // Reset error status
230     $this-> setError();
231     // Convert backslashes into forward slashes
232     $dir=str_replace('\\', '/', $dir);
233     // Check directory
234     if (!file_exists($dir)) {
235         // Specified directory does not exists
236         $this-> setError('Directory "' . $dir . '" does not exists' );
237     } elseif (!is_dir($dir)) {
238         // Specified resource is not a directory
239         $this-> setError('"' . $dir . '" is not a directory' );
240     } elseif (!is_readable($dir)) {
241         // Specified directory is not readable
242         $this-> setError('Directory "' . $dir . '" is not readable' );
243     } else {
244         // Everything is OK.
245         $result=true;
246         $this-> basedir=$dir;
247     }
248     return $result;
249 }
250
251 /**
252 * Read template file and parses contained templates
253 * @param string $file File name (relative to the base directory)
254 * @return boolean TRUE on success or FALSE on error
255 */
256 function readTemplatesFromFile($file='') {
257     $result=false;
258     // Reset error status
259     $this-> setError();
260     // Read file contents
261     if (false!==$tpl=$this-> getFileContents($file)) {
262         // Parse template string into structure
263         $cdata_prefix='';
264         $cdata_postfix='';
265         if (false=== $result=$this-> parseIntoStruct($this-> tpl_struct, $this-> tpl_depth,
266 $cdata_prefix, $cdata_postfix, $tpl)) {
267             // Template parser error
268             echo htmlentities($this-> getLastError());
269             die();
270         }
271         // Create new reference arrays
272         $this-> makeRefs($this-> tpl_struct, '');
273         array_pop($this-> files);
274     }
275     return $result;
276 }
277
278 /**
279 * Read file into a string
280 * @param string $file File name (relative to the base directory)
281 * @return mixed (string) File contents on success or (boolean) FALSE on error
282 */
283 function getFileContents($file='') {
284     $result=false;
285     // Reset error status
286     $this-> setError();
287     // Convert backslashes into forward slashes
288     // Create file name with path
289     $fn=realpath($this-> basedir.'/'.$file);
290     // Check file
291     if ($file=='') {
292         // Empty filename
293         $this-> setError('Template file name is empty');
294     } elseif (!file_exists($fn)) {
295         // File does not exists
296         $this-> setError('Could not open file "' . $this-> basedir.'/'.$file . '" for
reading: file does not exists');
297     } else if (!is_file($fn)) {
298         // Specified resource is not a file
299         $this-> setError('Could not open file "' . $fn . '" for reading: it is not a file' );
300     }

```



```

301     }else if (!is_readable($fn)) {
302         // File not readable
303         $this-> setError('Could not open file "' . $fn . '" for reading: file is not
readable');
304     } else {
305         // File exists and readable.
306         // Check whether the file were already read (avoiding unterminated recursion)
307         $result=true;
308         if (!empty($this-> files)) {
309             $tmp=$this-> files;
310             foreach ($tmp as $frec) {
311                 list($name,)=each($frec);
312                 if ($name==$fn) {
313                     // File has already been opened
314                     $this-> getLastFileData($fn_old, $offset);
315                     $result=false;
316                     $this-> setError('Error in file "' . $fn_old . '" at line ' . $this->
getLineNumber($fn_old, $offset) . ': the file "' . $fn . '" has already been opened: unterminated
recursion detected!');
317                     break;
318                 }
319             }
320         }
321         if ($result) {
322             // Read file contents
323             if (false=== $result=file_get_contents($fn)) {
324                 // Failed to read file contents into string
325                 $this-> setError('Could not open file "' . $fn . '" for reading: file read
failure');
326             } else {
327                 // File contents were successfully read
328                 array_push($this-> files, array($fn=> 0));
329             }
330         }
331     }
332     return $result;
333 }
334
335
336 /**
337  * Parse template string into the internal structure
338  * @param array $tpl_struct A reference to the template structure array
339  * @param array $tpl_depth A reference to the current template structure depth
340  * @param array $cdata_prefix A reference to the variable where CDATA between offset 0
341  * and first element will be stored
342  * @param array $cdata_postfix A reference to the variable where CDATA between last element
343  * and end of the file will be stored
344  * @param string $tpl A reference to the template string
345  * @return boolean TRUE on success or FALSE on error
346  */
347 function parseIntoStruct(& $tpl_struct, & $tpl_depth, & $cdata_prefix,
& $cdata_postfix, & $tpl) {
348     // Reset error status
349     $this-> setError();
350     $result=false;
351     if ($tpl!='') {
352         $ns=((PCPIN_TPL_NS!='')? PCPIN_TPL_NS.': ' : '');
353         // REGEX pattern for matching PCPIN template tags
354         $tag_pattern='/(<([ ])*' . $ns . '([A-Za-z0-9_])+([ ])*([ ])+([A-Za-z0-9_])+([ ])*=([
 ])*"^[^"]*"([ ])*([\/])?([ ])*>)|((<([ ])*[\/]([ ])*' . $ns . '([A-Za-z0-9_])+([ ])*
 ])*>)/' ;
355         // Parse tags
356         if (false===preg_match_all($tag_pattern, $tpl, $matches,
PREG_PATTERN_ORDER|PREG_OFFSET_CAPTURE)) {
357             // An unknown error occurred
358             $this-> setError('Template parser internal error');
359         } else {
360             if (empty($matches[0])) {
361                 // There are no template tags found.
362                 $result=true;
363                 $cdata_prefix=$tpl;
364                 $tpl_struct=array();
365                 $cdata_postfix='';
366             } else {
367                 // There are some template tags. Parse.
368                 $matches=$matches[0];
369                 $total_elements=count($matches);
370                 $tag_opened=false;
371                 $i=0;
372                 foreach ($matches as $match) {

```

```

373 $match_0=$match[0];
374 // This match' offset
375 $offset=$match[1];
376 // Is there CDATA before the first element?
377 if ($i==0 && $offset> 0) {
378     // There is some CDATA before first element
379     $cdata_prefix=substr($tpl, 0, $offset);
380 }
381 $thisfiles=array_pop($this-> files);
382 list($fn,)=each($thisfiles);
383 array_push($this-> files, array($fn=> $offset));
384 // Get first CDATA borders
385 $cdata_start=$offset+strlen($match_0);
386 $cdata_end=isset($matches[$i+1])? $matches[$i+1][1] : $cdata_start;
387 // Prepare string
388 $match_0=trim(ltrim(rtrim(str_replace($ns, '', $match_0), '>' ), '<' ));
389 // Which tag? (opening/closing/closed)
390 $slashpos=strpos($match_0, '/');
391 if (false=== $slashpos) {
392     // Opening tag
393     $tag_type=0;
394 } elseif ($slashpos> 0) {
395     // Closed tag (both start and end elements)
396     $tag_type=1;
397     $match_0=trim(rtrim($match_0, '/'));
398 } else {
399     // Closing tag
400     $tag_type=2;
401     $match_0=trim(ltrim($match_0, '/'));
402 }
403 // Tag name
404 $name=substr($match_0, 0, strpos($match_0.' ', ' '));
405 // Parse attributes
406 $attrs=array();
407 if ($tag_type==0 || $tag_type==1) {
408     $pattern='/( [ ])*[A-Za-z0-9_]*=([ ])*"[^"]*"([ ])*\//';
409     if (false!==preg_match_all($pattern, $match_0, $attr_matches)) {
410         if (!empty($attr_matches[0])) {
411             $attr_pairs=$attr_matches[0];
412         }
413     }
414     foreach ($attr_pairs as $attrstr) {
415         if ($attrstr!='' && $attrstr!='/') {
416             list($key, $val)=explode('=', $attrstr);
417             $attrs[trim($key)]=substr(trim($val), 1, -1);
418         }
419     }
420 }
421 // Call handlers
422 if ($tag_type==0) {
423     // Opening tag
424     // Call start element handler
425     if (false=== $result=$this-> startElement($tpl_struct, $tpl_depth, $name, $attrs)) {
426         // An error occurred
427         break;
428     } else {
429         // Get first part of CDATA
430         if (0< $cdata_length=$cdata_end-$cdata_start) {
431             if (false=== $result=$this-> characterData($tpl_struct, $tpl_depth, substr($tpl,
432 $cdata_start, $cdata_length))) {
433                 // An error occurred
434                 break;
435             }
436         }
437     }
438 } else if ($tag_type==1) {
439     // Closed tag (both start and end elements)
440     // Call start and end element handlers
441     if (false=== $result=$this-> startElement($tpl_struct, $tpl_depth, $name, $attrs)
442 && $this-> endElement($tpl_struct, $tpl_depth, $name)) {
443         // An error occurred
444         break;
445     } else {
446         // Get next part of CDATA
447         if (0< $cdata_length=$cdata_end-$cdata_start) {
448             if (false=== $result=$this-> characterData($tpl_struct, $tpl_depth, substr($tpl,
449 $cdata_start, $cdata_length))) {
450                 // An error occurred
451                 break;
452             }
453         }
454     }
455 }

```

```

450     }
451   }
452   } else {
453     // Closing tag
454     // Call end element handler
455     if (false=== $result= $this-> endElement($tpl_struct, $tpl_depth, $name)) {
456       // An error occurred
457       break;
458     } else {
459       // Get next part of CDATA
460       if (0< $cdata_length=$cdata_end-$cdata_start) {
461         if (false=== $result= $this-> characterData($tpl_struct, $tpl_depth, substr($tpl,
462 $cdata_start, $cdata_length))) {
463           // An error occurred
464           break;
465         }
466       }
467     }
468     $i++;
469     if ($total_elements==$i && $result) {
470       // Last element
471       $cdata_postfix=substr($tpl, $cdata_start);
472     }
473   }
474   if ($result===true && $tpl_depth!=0) {
475     // Error: tag is still open at the end of file
476     $this-> getLastFileData($fn, $offset);
477     $this-> setError('Error in file "' . $fn . '" : element
478 "<' . $ns.$tpl_struct[$tpl_depth-1]['name'] . '>" is still open at the end of file'
479 );
480     $result=false;
481   }
482 }
483 // Optimize $this->tpl_vars_plain array
484 $this-> tpl_vars_plain=array_unique($this-> tpl_vars_plain);
485 return $result;
486 }
487
488 /**
489  * Start element handler
490  * @param array $tpl_struct A reference to the template structure array
491  * @param array $tpl_depth A reference to the current template structure depth
492  * @param string $name Element name
493  * @param array $attrs Element attributes
494  * @return boolean TRUE on success or FALSE on error
495  */
496 function startElement(& $tpl_struct, & $tpl_depth, $name, $attrs) {
497   $result=false;
498   if ($tpl_depth==0 && !empty($tpl_struct)) {
499     // More than one root element detected
500     $this-> getLastFileData($fn, $offset);
501     $this-> setError('Error in file "' . $fn . '" at line ' . $this-> getLineNumber($fn,
502 $offset) . ': element "<' . PCPIN_TPL_FULL_NS.$name . '>" is not allowed here'
503 );
504   } elseif ($name!=PCPIN_TPL_NAME_MAIN && $name!=PCPIN_TPL_NAME_SUB) {
505     // Unknown element
506     $this-> getLastFileData($fn, $offset);
507     $this-> setError('Error in file "' . $fn . '" at line ' . $this-> getLineNumber($fn,
508 $offset) . ': unknown element "<' . PCPIN_TPL_FULL_NS.$name . '>"
509 );
510   } elseif ($name==PCPIN_TPL_NAME_MAIN && $tpl_depth> 0 &&
511 $tpl_struct[$tpl_depth-1]['template_type']=='condition') {
512     // Element with this name is not allowed here
513     $this-> getLastFileData($fn, $offset);
514     $this-> setError('Error in file "' . $fn . '" at line ' . $this-> getLineNumber($fn,
515 $offset) . ': element "<' . PCPIN_TPL_FULL_NS.$name . '>" is not allowed here'
516 );
517   } elseif ($name==PCPIN_TPL_NAME_SUB && ($tpl_depth==0 || $tpl_struct[$tpl_depth-
518 1]['name']==PCPIN_TPL_NAME_SUB)) {
519     // Element with this name is not allowed here
520     $this-> getLastFileData($fn, $offset);
521     $this-> setError('Error in file "' . $fn . '" at line ' . $this-> getLineNumber($fn,
522 $offset) . ': element "<' . PCPIN_TPL_FULL_NS.$name . '>" is not allowed here'
523 );
524   } else {
525     $result=true;
526     // Check SUBtemplate
527     if ($name==PCPIN_TPL_NAME_SUB) {
528       // Its a subtemplate
529       if (!array_key_exists('condition', $attrs)) {
530         // A subtemplate requires the "condition" attribute

```

```

522     $result=false;
523     $this-> getLastFileData($fn, $offset);
524     $this-> setError('Error in file "' . $fn.'" at line ' . $this-
>     getLineNumber($fn, $offset).': a subtemplate requires the "condition" attribute'
525     } else {
526         // The name of the parent template
527         $parent_name=(isset($tpl_struct[$tpl_depth-1]['attrs']['name']))? $tpl_struct[$tpl_depth-
1]['attrs']['name'] : '';
528     }
529 }
530 if ($result) {
531     // Get the value of "name" attribute
532     $tpl_name=(isset($attrs['name']))? $attrs['name'] : '';
533     // Check template type
534     $template_type=(isset($attrs['type']))? $attrs['type'] : '';
535     $tpl_vars=array();
536     if ($name!=PCPIN_TPL_NAME_SUB) {
537         // Check type
538         switch($template_type) {
539             default : // An unknown type
540                 $result=false;
541                 $this-> getLastFileData($fn, $offset);
542                 $this-> setError('Error in file "' . $fn.'" at
line '.$this-> getLineNumber($fn, $offset).': unknown template type "' . $template_type.'"
543                 break;
544             case '' : // Empty type
545                 // Check template name
546                 if ($tpl_name=='') {
547                     // Required attribute "name" is empty
548                     $result=false;
549                     $this-> getLastFileData($fn, $offset);
550                     $this-> setError('Error in file "' . $fn.'" at
line '.$this-> getLineNumber($fn, $offset).': template of this type requires non-empty "name"
attribute');
551                 }
552                 break;
553             case 'simplecondition' : // type="simplecondition"
554                 // Check template name
555                 if ($tpl_name=='') {
556                     // Required attribute "name" is empty
557                     $result=false;
558                     $this-> getLastFileData($fn, $offset);
559                     $this-> setError('Error in file "' . $fn.'" at
line '.$this-> getLineNumber($fn, $offset).': template of this type requires non-empty "name"
attribute');
560                 }
561                 // "simplecondition" template requires non-empty
"requiredvars" attribute
562                 if (isset($attrs['requiredvars'])) {
563                     $attrs['requiredvars']=strtoupper(trim($attrs['requiredvars']));
564                 }
565                 if (empty($attrs['requiredvars'])) {
566                     // "requiredvars" attribute is empty or not set
567                     $result=false;
568                     $this-> getLastFileData($fn, $offset);
569                     $this-> setError('Error in file "' . $fn.'" at
line '.$this-> getLineNumber($fn, $offset).': template of type "simplecondition" requires non-
empty "requiredvars" attribute'
570                     );
571                 } else {
572                     // Store variable names
573                     $tmp=explode(',', $attrs['requiredvars']);
574                     foreach ($tmp as $var) {
575                         $var=trim($var);
576                         if ($var!='') {
577                             $tpl_vars[$var]=null;
578                         }
579                     }
580                 }
581                 break;
582             case 'condition' : // type="condition"
583                 // "condition" template requires non-empty
"conditionvar" attribute
584                 if (isset($attrs['conditionvar'])) {
585                     $attrs['conditionvar']=strtoupper(trim($attrs['conditionvar']));
586                 }
587                 if (empty($attrs['conditionvar'])) {
588                     // "conditionvar" attribute is empty or not set
589                     $result=false;

```

```

589                                     $this->    getLastFileData($fn, $offset);
590                                     $this->    setError('Error in file "' . $fn . '" at
line ' . $this->    getLineNumber($fn, $offset) . ': template of type "condition" requires non-empty
"conditionvar" attribute'
591                                     );
592                                     } else {
593                                     // Store variable name
594                                     $tpl_vars[$attrs['conditionvar']] = null;
595                                     }
596                                     break;
597             }
598         }
599         if ($result) {
600             // Template type is OK
601             $this->    tpl_vars[$tpl_name] = $tpl_vars;
602             $this->    parsed_name_ref[$tpl_name] = '';
603             $this->    parsed_name_flags[$tpl_name] = false;
604             $children = array();
605             $child_types = array();
606             if (isset($attrs['src'])) {
607                 // The template has an external source (included)
608                 if (false === $tpl = $this->    getFileContents($attrs['src'])) {
609                     // Failed to read template from file
610                     $result = false;
611                 } else {
612                     // Parse included template source
613                     $element_start_src = '<' . PCPIN_TPL_FULL_NS . $name;
614                     foreach ($attrs as $attr_key => $attr_val) {
615                         if ($attr_key != 'src') {
616                             $element_start_src .= ' ' . $attr_key . '=' . $attr_val . ' ';
617                         }
618                     }
619                     $element_start_src .= '>';
620                     $element_end_src = '</' . PCPIN_TPL_FULL_NS . $name . '>';
621                     $tpl = $element_start_src . $tpl . $element_end_src;
622                     $tpl_struct_sub = array();
623                     $tpl_depth_sub = 0;
624                     $cdata_prefix = '';
625                     $cdata_postfix = '';
626                     if (false !== $result = $this->    parseIntoStruct($tpl_struct_sub, $tpl_depth_sub,
$cdata_prefix, $cdata_postfix, $tpl)) {
627                         if ($cdata_prefix != '') {
628                             // Included template has CDATA before the first element
629                             array_push($children, $cdata_prefix);
630                             array_push($child_types, 1);
631                         }
632                         if (!empty($tpl_struct_sub)) {
633                             array_push($children, $tpl_struct_sub);
634                             array_push($child_types, 0);
635                         }
636                         if ($cdata_postfix != '') {
637                             // Included template has CDATA after the last element
638                             array_push($children, $cdata_postfix);
639                             array_push($child_types, 1);
640                         }
641                     }
642                     array_pop($this->    files);
643                 }
644             }
645             if ($result) {
646                 $node = array('name' =>    $name,
647                             'template_type' =>    $template_type, // If a template, then here is the value
of "type" attribute, if any
648                             'attrs' =>    $attrs,
649                             'child_types' =>    $child_types, // 0: template record (array), 1: cdata
(string)
650                             'children' =>    $children
651                             );
652                 $tpl_depth++;
653                 array_push($tpl_struct, $node);
654             }
655         }
656     }
657     return $result;
658 }
659
660 /**
661  * Characted data handler
662  * @param array $tpl_struct A reference to the template structure array

```

```

664 * @param array $tpl_depth A reference to the current template structure depth
665 * @param string $cdata Character data
666 * @return boolean TRUE on success or FALSE on error
667 */
668 function characterData(& $tpl_struct, & $tpl_depth, $cdata='') {
669     $result=false;
670     $tpl_depth_dec=$tpl_depth-1;
671     if (!empty($tpl_depth) && $tpl_struct[$tpl_depth_dec]['template_type']=='condition'
&& trim($cdata)!='') {
672         // Character data is not allowed here
673         $this-> getLastFileData($fn, $offset);
674         $this-> setError('Error in file " ' . $fn . '" at line ' . $this-> getLineNumber($fn,
$offset).': character data is not allowed here');
675     } else {
676         $result=true;
677         if ($tpl_depth_dec>= 0 &&
$tpl_struct[$tpl_depth_dec]['template_type']!='condition') {
678             // Condition template does needs even empty CDATA
679             $tpl_struct[$tpl_depth_dec]['child_types'][]=1; // CDATA
680             array_push($tpl_struct[$tpl_depth_dec]['children'], $cdata);
681         }
682     }
683     return $result;
684 }
685
686 /**
687 * End element handler
688 * @param array $tpl_struct A reference to the template structure array
689 * @param array $tpl_depth A reference to the current template structure depth
690 * @param string $name Element name
691 * @return boolean TRUE on success or FALSE on error
692 */
693 function endElement(& $tpl_struct, & $tpl_depth, $name) {
694     $result=false;
695     $hierarchy_error=false;
696     if (!isset($tpl_struct[0])) {
697         // Closing tag that is not opened
698         $this-> getLastFileData($fn, $offset);
699         $this-> setError('Error in file " ' . $fn . '" at line ' . $this-> getLineNumber($fn,
$offset).': closing tag that is not opened');
700     } else {
701         $node=array_pop($tpl_struct);
702         if ($node['name']!=$name) {
703             // Closing tag that is not opened
704             $this-> getLastFileData($fn, $offset);
705             $this-> setError('Error in file " ' . $fn . '" at line ' . $this->
getLineNumber($fn, $offset).': wrong closing tag
("</' . PCPIN_TPL_FULL_NS.$node['name'] . '>" expected)');
706         } else {
707             $result=true;
708             $tpl_depth--;
709             $tpl_depth_dec=$tpl_depth-1;
710             if ($tpl_depth> 0) {
711                 $tpl_struct[$tpl_depth_dec]['child_types'][]=0; // Node
712                 array_push($tpl_struct[$tpl_depth_dec]['children'], $node);
713             } else {
714                 // Root element
715                 // Trim root element CDATA, if needed
716                 if (PCPIN_TPL_TRIM_ROOT && !empty($node['child_types'])) {
717                     if ($node['child_types'][0]==1) {
718                         if ("\r\n" ==substr($node['children'][0], 0, 2)) {
719                             $node['children'][0]=substr($node['children'][0], 2);
720                         } elseif ("\r" ==substr($node['children'][0], 0, 1) ||
"\n" ==substr($node['children'][0], 0, 1)) {
721                             $node['children'][0]=substr($node['children'][0], 1, -1);
722                         }
723                     }
724                     $last_child=count($node['child_types'])-1;
725                     if ($last_child> 0 && $node['child_types'][$last_child]==1) {
726                         if ("\r\n" ==substr($node['children'][$last_child], -2)) {
727                             $node['children'][$last_child]=substr($node['children'][$last_child], 0, -2);
728                         } elseif ("\r" ==substr($node['children'][$last_child], -1) ||
"\n" ==substr($node['children'][$last_child], -1)) {
729                             $node['children'][$last_child]=substr($node['children'][$last_child], 1, -1);
730                         }
731                     }
732                 }
733             }
734             $tpl_struct=$node;
735         }
736     }

```

```

736 // Get variables from node's CDATA elements
737 $tpl_name='';
738 if ($name==PCPIN_TPL_NAME_SUB) {
739     // A subtemplate
740     $tpl_name=$tpl_struct[$tpl_depth_dec]['attrs']['name'];
741 } elseif (isset($node['attrs']['name'])) {
742     $tpl_name=$node['attrs']['name'];
743 }
744 if ($tpl_name!='') {
745     $cdata='';
746     if (!empty($node['child_types'])) {
747         foreach ($node['child_types'] as $key=> $type) {
748             if ($type==1) {
749                 $cdata.=$node['children'][$key];
750             }
751         }
752     }
753     if ($cdata!='') {
754         // Extract variables from CDATA
755         if (false!==preg_match_all('/{[^{}]*}/', $cdata, $matches)) {
756             if (!empty($matches[0])) {
757                 $matches=$matches[0];
758                 foreach ($matches as $match) {
759                     $var_name=trim(substr($match, 1, -1));
760                     if ($var_name!='') {
761                         $this-> tpl_vars[$tpl_name][$var_name]=null;
762                         $this-> tpl_vars_plain[]=$var_name;
763                     }
764                 }
765             }
766         }
767     }
768 }
769 }
770 }
771 return $result;
772 }
773
774
775 /**
776  * returns name and byte offset of the current opened file
777  * @param string $filename A reference to the variable where file name will be stored
778  * @param array $offset A reference to the variable where byte offset will be stored
779  */
780 function getLastFileData(& $filename, & $offset) {
781     if (!empty($this-> files)) {
782         $tmp=$this-> files;
783         list($filename, $offset)=each(end($tmp));
784     } else {
785         $filename='';
786         $offset=0;
787     }
788 }
789
790
791 /**
792  * Create new name reference arrays
793  * @param array Element to start with
794  */
795 function makeRefs(& $root, $parent_tpl_name='') {
796     if (!empty($root) && is_array($root)) {
797         if ($root['name']==PCPIN_TPL_NAME_SUB) {
798             // Element is a subtemplate
799             if (!isset($this-> sub_name_ref[$parent_tpl_name])) {
800                 $this-> sub_name_ref[$parent_tpl_name]=array();
801             }
802             $this-> sub_name_ref[$parent_tpl_name][]=& $root;
803         } else {
804             // Element is a template
805             $parent_tpl_name=$root['attrs']['name'];
806             $this-> tpl_name_ref[$parent_tpl_name]=& $root;
807         }
808         // Children?
809         foreach ($root['child_types'] as $key=> $type) {
810             if ($type==0) {
811                 $this-> makeRefs($root['children'][$key], $parent_tpl_name);
812             }
813         }
814     }
815 }

```



```

816
817
818 /**
819  * Get number of line at which the character with specified offset is located
820  * @param string $filename File name to search in
821  * @param int $char Character offset
822  * @return int
823  */
824 function getLineNumber($filename='', $char=0) {
825     $result=0;
826     if (false!==$h=file($filename)) {
827         $i=0;
828         foreach ($h as $line_nr=> $str) {
829             $i+=strlen($str);
830             if ($i> $char) {
831                 $result=$line_nr+1;
832                 break;
833             }
834         }
835     }
836     return $result;
837 }
838
839 /**
840  * Add a variable to the template
841  * @param string $template Template name
842  * @param string $var_name Variable name
843  * @param mixed $var_val Variable value
844  */
845 function addVar($template='', $var_name='', $var_val=null) {
846     $var_name=strtoupper(trim($var_name));
847     if ($var_name!='' && isset($this->tpl_vars[$template]) &&
848     array_key_exists($var_name, $this->tpl_vars[$template])) {
849         $this->tpl_vars[$template][$var_name]=$var_val;
850     }
851 }
852
853 /**
854  * Add multiple variables to the template.
855  * The $vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)
856  * @param string $template Template name
857  * @param array $vars Variables to add
858  */
859 function addVars($template='', $vars=null) {
860     if (!empty($vars) && is_array($vars)) {
861         foreach ($vars as $key=> $val) {
862             $this->addVar($template, $key, $val);
863         }
864     }
865 }
866
867 /**
868  * Add a global variable
869  * @param string $var_name Variable name
870  * @param mixed $var_val Variable value
871  */
872 function addGlobalVar($var_name='', $var_val=null) {
873     $var_name=strtoupper(trim($var_name));
874     if ($var_name!='') {
875         $this->global_vars[$var_name]=$var_val;
876     }
877 }
878
879 /**
880  * Add multiple global variables
881  * The $vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)
882  * @param array $vars Variables to add
883  */
884 function addGlobalVars($vars=null) {
885     if (!empty($vars) && is_array($vars)) {
886         foreach ($vars as $key=> $val) {
887             $this->addGlobalVar($key, $val);
888         }
889     }
890 }
891
892
893
894

```



```

895
896 /**
897  * Parse template
898  * @param string $name Template name
899  * @param string $mode Parse mode
900  */
901 function parseTemplate($name='', $mode=PCPIN_TPL_DEFAULT_PARSE_MODE) {
902     if ($name!='' && isset($this-> tpl_name_ref[$name])) {
903         $this-> parsed_name_flags[$name]=true;
904         if ($mode==PCPIN_TPL_PARSE_MODE_OVERWRITE) {
905             $this-> parsed_name_ref[$name]='';
906         }
907         // Parse template
908         $this-> parsed_name_ref[$name].=$this-> parseIntoString($this-> tpl_name_ref[$name],
909 $this-> tpl_vars[$name]);
910     }
911 }
912
913 /**
914  * Parse template (if not parsed yet) and return it's parsed contents as string
915  * @param string $name Template name
916  * @return string
917  */
918 function getParsedTemplate($name='') {
919     $result='';
920     if ($name=='') {
921         if (isset($this-> tpl_struct['attrs'])) {
922             if (isset($this-> tpl_struct['attrs']['name'])) {
923                 $name=$this-> tpl_struct['attrs']['name'];
924             }
925         }
926     }
927     if ($name!='' && isset($this-> tpl_name_ref[$name])) {
928         if (false===$this-> parsed_name_flags[$name]) {
929             // Parse template
930             $this-> parseTemplate($name);
931         }
932         $result=$this-> parsed_name_ref[$name];
933     }
934     return $result;
935 }
936
937
938 /**
939  * Parse template structure and return it's parsed contents as a string
940  * @param array $tpl Template record
941  * @param array $vars Template variables
942  * @return string
943  */
944 function parseIntoString($tpl, $vars) {
945     $parsed_string='';
946     if (is_array($tpl) && !empty($tpl)) {
947         $child_key=-1;
948         if ($tpl['name']==PCPIN_TPL_NAME_SUB) {
949             // Subtemplate (must be parsed)
950             $parse=true;
951         } else {
952             // Check template
953             $parse=false;
954             $tpl_name=$tpl['attrs']['name'];
955             // Which type is the template of?
956             if (empty($tpl['template_type'])) {
957                 // A simple template. Parse without conditions.
958                 $parse=true;
959             } elseif ($tpl['template_type']=='simplecondition') {
960                 // A simple condition template. Check condition.
961                 $parse=true;
962                 if (!empty($tpl['attrs']['requiredvars'])) {
963                     $requiredvars=explode(',', $tpl['attrs']['requiredvars']);
964                     // Check each var
965                     foreach ($requiredvars as $var) {
966                         $var=trim($var);
967                         if ($var!='' && empty($this-> tpl_vars[$tpl_name][$var])) {
968                             // At least one of the required vars is empty. Do not parse.
969                             $parse=false;
970                             break;
971                         }
972                     }
973                 }
974             }
975         }
976     }

```

```

974     } elseif ($tpl['template_type']=='condition') {
975         // A conditional template. Check subtemplates.
976         $conditionvar=$this->tpl_vars[$tpl_name][$tpl['attrs']['conditionvar']];
977         if (!empty($tpl['child_types'])) {
978             foreach ($tpl['child_types'] as $key=> $type) {
979                 if ($type==0) {
980                     $child=$tpl['children'][$key];
981                     if (isset($child['attrs']['condition'])) {
982                         $child_condition=$child['attrs']['condition'];
983                         if ($child_condition=='default') {
984                             // One of subtemplates is a default template
985                             $parse=true;
986                             $child_key=$key;
987                         } elseif ($child_condition=='empty' && empty($conditionvar)) {
988                             // Condition variable has an empty value and one of subtemplates has an empty
condition.
989                             $parse=true;
990                             $child_key=$key;
991                             break;
992                         } elseif ($child_condition!='default' && $child_condition!='empty'
&& (string)$child_condition==(string)$conditionvar) {
993                             // One of the subtemplates has the same condition as the condition variable's
value
994                             $parse=true;
995                             $child_key=$key;
996                             break;
997                         }
998                     }
999                 }
1000             }
1001         }
1002     }
1003 }
1004 if ($parse) {
1005     // Parse template children.
1006     if ($child_key>= 0) {
1007         // Parse only one child (a subtemplate)
1008         $parsed_string=$this->parseIntoString($tpl['children'][$child_key], $vars);
1009     } else {
1010         // Parse all children
1011         if (!empty($tpl['child_types'])) {
1012             foreach ($tpl['child_types'] as $key=> $type) {
1013                 // Which type is the child of?
1014                 if ($type==1) {
1015                     // A CDATA
1016                     // Pass variables and store.
1017                     $parsed_string=$this->passVars($tpl['children'][$key], $vars);
1018                 } elseif ($type==0) {
1019                     // A template
1020                     $parsed_string=$this->
> getParsedTemplate($tpl['children'][$key]['attrs']['name']);
1021                 }
1022             }
1023         }
1024     }
1025 }
1026 }
1027 return $parsed_string;
1028 }
1029
1030
1031 /**
1032  * Pass variables to the parsed template string.
1033  * Global variables will be passed too.
1034  * @param string $parsed Parsed template string
1035  * @param array $vars Variables to pass
1036  * @return string
1037  */
1038 function passVars($parsed='', $vars=null) {
1039     if ($parsed!='' && !empty($vars) && is_array($vars)) {
1040         // Replace '{' characters in the values in order to avoid wrong name-value replacements
1041         $replacement=chr(0).'pcpin'.chr(0);
1042         $replaced=false;
1043         // Add global vars
1044         if (!empty($this->global_vars)) {
1045             foreach ($this->global_vars as $var_name=> $var_value) {
1046                 if (empty($vars[$var_name])) {
1047                     if (!is_scalar($var_value)) {
1048                         $var_value='';
1049                     } elseif (!empty($var_value) && false!==strpos($var_value, '{')) {

```

```

1050         $replaced=true;
1051         $var_value=str_replace('{', $replacement, $var_value);
1052     }
1053     $parsed=str_replace('{'.$var_name.'}', $var_value, $parsed);
1054 }
1055 }
1056 }
1057 // Add local vars
1058 foreach ($vars as $var_name=> $var_value) {
1059     if (!is_scalar($var_value)) {
1060         $var_value='';
1061     } elseif (!empty($var_value) && false!==strpos($var_value, '{')) {
1062         $replaced=true;
1063         $var_value=str_replace('{', $replacement, $var_value);
1064     }
1065     $parsed=str_replace('{'.$var_name.'}', $var_value, $parsed);
1066 }
1067 if ($replaced) {
1068     $parsed=str_replace($replacement, '{', $parsed);
1069 }
1070 }
1071 return $parsed;
1072 }
1073
1074
1075 /**
1076  * Display parsed template. If the template is not parsed yet, then it will be parsed.
1077  * @param string $name Template name
1078  */
1079 function displayParsedTemplate($name='') {
1080     echo $this-> getParsedTemplate($name);
1081 }
1082
1083
1084 /**
1085  * Clear template variables and parsed contents of a template
1086  * @param string $name Template name
1087  */
1088 function clearTemplate($name='') {
1089     if ($name=='') {
1090         if (isset($this-> tpl_struct['attrs'])) {
1091             if (isset($this-> tpl_struct['attrs']['name'])) {
1092                 $name=$this-> tpl_struct['attrs']['name'];
1093             }
1094         }
1095     }
1096     if ($name!='' && isset($this-> tpl_name_ref[$name])) {
1097         $this-> parsed_name_ref[$name]='';
1098         $this-> parsed_name_flags[$name]=false;
1099         foreach ($this-> tpl_vars[$name] as $key=> $val) {
1100             $this-> tpl_vars[$name][$key]=null;
1101         }
1102     }
1103 }
1104
1105 }
1106 }
1107 ?>

```

# Index

## C

<a href="#">constructor PcpinTpl::PcpinTpl()</a> . . . . .	6
<i>Constructor</i>	

## P

<a href="#">PcpinTpl::getFileContents()</a> . . . . .	9
<i>Read file into a string</i>	
<a href="#">PcpinTpl::getLastError()</a> . . . . .	9
<i>Get last raised error description</i>	
<a href="#">PcpinTpl::getLastFileData()</a> . . . . .	9
<i>returns name and byte offset of the current opened file</i>	
<a href="#">PcpinTpl::getLineNumber()</a> . . . . .	10
<i>Get number of line at which the character with specified offset is located</i>	
<a href="#">PcpinTpl::endElement()</a> . . . . .	9
<i>End element handler</i>	
<a href="#">PcpinTpl::displayParsedTemplate()</a> . . . . .	8
<i>Display parsed template.</i>	
<a href="#">PcpinTpl::addVar()</a> . . . . .	7
<i>Add a variable to the template</i>	
<a href="#">PcpinTpl::addVars()</a> . . . . .	7
<i>Add multiple variables to the template.</i>	
<a href="#">PcpinTpl::characterData()</a> . . . . .	8
<i>Characted data handler</i>	
<a href="#">PcpinTpl::clearTemplate()</a> . . . . .	8
<i>Clear template variables and parsed contents of a template</i>	
<a href="#">PcpinTpl::getParsedTemplate()</a> . . . . .	10
<i>Parse template (if not parsed yet) and return it's parsed contents as string</i>	
<a href="#">PcpinTpl::makeRefs()</a> . . . . .	10
<i>Create new name reference arrays</i>	
<a href="#">PcpinTpl::setBasedir()</a> . . . . .	12
<i>Set new base directory</i>	
<a href="#">PcpinTpl::setError()</a> . . . . .	13
<i>Set error status</i>	
<a href="#">PcpinTpl::startElement()</a> . . . . .	13
<i>Start element handler</i>	
<a href="#">pcpintpl.class.php</a> . . . . .	18
<i>Source code</i>	
<a href="#">PcpinTpl::resetTpl()</a> . . . . .	12
<i>Reset template structure</i>	
<a href="#">PcpinTpl::readTemplatesFromFile()</a> . . . . .	12
<i>Read template file and parses contained templates</i>	
<a href="#">PcpinTpl::parseIntoString()</a> . . . . .	10
<i>Parse template structure and return it's parsed contents as a string</i>	

<a href="#">PcpinTpl::parseIntoStruct()</a>	11
<i>Parse template string into the internal structure</i>	
<a href="#">PcpinTpl::parseTemplate()</a>	11
<i>Parse template</i>	
<a href="#">PcpinTpl::passVars()</a>	12
<i>Pass variables to the parsed template string.</i>	
<a href="#">PcpinTpl::addGlobalVars()</a>	7
<i>Add multiple global variables</i>	
<a href="#">PcpinTpl::addGlobalVar()</a>	6
<i>Add a global variable</i>	
<a href="#">PCPIN_TPL_PARSE_MODE_OVERWRITE</a>	3
<i>Parse mode "overwrite" identifier</i>	
<a href="#">PCPIN_TPL_TRIM_ROOT</a>	3
<i>Wether to delete line break between root element and the next line and between the last line and root element's closing tag</i>	
<a href="#">PcpinTpl</a>	4
<i>Class PcpinTpl</i>	
<a href="#">PcpinTpl::\$basedir</a>	4
<i>The directory where the template files are stored</i>	
<a href="#">PCPIN_TPL_PARSE_MODE_APPEND</a>	3
<i>Parse mode "append" identifier</i>	
<a href="#">PCPIN_TPL_NS</a>	3
<i>PCPIN template elements namespace</i>	
<a href="#">PCPIN_TPL_DEFAULT_PARSE_MODE</a>	2
<i>Default parse mode</i>	
<a href="#">PCPIN_TPL_FULL_NS</a>	2
<i>PCPIN template elements namespace full description</i>	
<a href="#">PCPIN_TPL_NAME_MAIN</a>	2
<i>Main template element name</i>	
<a href="#">PCPIN_TPL_NAME_SUB</a>	3
<i>Subtemplate element name</i>	
<a href="#">PcpinTpl::\$files</a>	4
<i>Array with opened files' description (name=&gt;byte_offset)</i>	
<a href="#">PcpinTpl::\$global_vars</a>	4
<i>Global variables.</i>	
<a href="#">PcpinTpl::\$tpl_name_ref</a>	6
<i>Array with references to the templates.</i>	
<a href="#">PcpinTpl::\$tpl_struct</a>	6
<i>Template structure</i>	
<a href="#">PcpinTpl::\$tpl_vars</a>	6
<i>Template variables referenced by the template name.</i>	
<a href="#">PcpinTpl::\$tpl_vars_plain</a>	6
<i>All variable names used in all loaded templates</i>	
<a href="#">PcpinTpl::\$tpl_depth</a>	5
<i>Template structure current depth</i>	
<a href="#">PcpinTpl::\$sub_name_ref</a>	5
<i>Array with references to the subtemplates.</i>	
<a href="#">PcpinTpl::\$last_error</a>	5
<i>Last raised error description</i>	
<a href="#">PcpinTpl::\$parsed_name_flags</a>	5
<i>Parsed template flags (true, if the template is parsed)</i>	
<i>The templates are referenced by the value of an attribute "name".</i>	
<a href="#">PcpinTpl::\$parsed_name_ref</a>	5

*Parsed template contents*

*The templates are referenced by the value of an attribute "name".*

[pcpintpl.class.php](#) . . . . . 2

*PCPIN Template engine*

*Copyright (C) 2007 Kanstantin Reznichak*

*This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.*