PCPIN Template API (Application programming interface)

Contents

Package PcpinTpl Procedural Elements

pcpintpl.class.php

PCPIN Template engine   Copyright (C) 2007 Kanstantin Reznichak

- **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 overrided 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

```
<?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.
 *
 *    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/>.
 */


/**
 * PCPIN Template engine
 * @package PcpinTpl
 * @author Konstantin Reznichak <k.reznichak@pcpin.com>
 * @copyright Copyright &copy; 2007, Kanstantin Reznichak
 * @version 1.0
 */



/******************************************************************************
 *
 * CONFIGURATION
 *
 ******************************************************************************/

/**
 * PCPIN template elements namespace
 */
define('PCPIN_TPL_NS', 'PCPIN');

/**
 * Main template element name
 */
define('PCPIN_TPL_NAME_MAIN', 'TPL');

/**
 * Subtemplate element name
 */
define('PCPIN_TPL_NAME_SUB', 'SUB');

/**
 * Parse mode "overwrite" identifier
 */
define('PCPIN_TPL_PARSE_MODE_OVERWRITE', 'w');

/**
 * Parse mode "append" identifier
 */
define('PCPIN_TPL_PARSE_MODE_APPEND', 'a');

/**
 * Default parse mode
 */
define('PCPIN_TPL_DEFAULT_PARSE_MODE', PCPIN_TPL_PARSE_MODE_OVERWRITE);

/**
```

```
  * Wether to delete line break between root element and the next line and between the last line and root element's closing tag
  */
define('PCPIN_TPL_TRIM_ROOT', true);



/*******************************************************************************
 *
 * CONFIGURATION ENDS HERE
 * DO NOT EDIT ANYTHING BELOW UNTIL YOU ARE EXACTLY KNOW WHAT YOU ARE DOING!!!
 *
 ******************************************************************************/

/**
 * PCPIN template elements namespace full description
 */
define('PCPIN_TPL_FULL_NS', (PCPIN_TPL_NS!='')? (PCPIN_TPL_NS.':') : '');


/**
 * Class PcpinTpl
 * @package PcpinTpl
 */
class PcpinTpl {

  /**
   * Last raised error description
   * @var   string
   */
  var $last_error='';

  /**
   * The directory where the template files are stored
   * @var   string
   */
  var $basedir='';

  /**
   * Array with opened files' description (name=>byte_offset)
   * @var   array
   */
  var $files=null;

  /**
   * Template structure
   * @var   array
   */
  var $tpl_struct=null;

  /**
   * Template structure current depth
   * @var   int
   */
  var $tpl_depth=0;

  /**
   * 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
   * @var   array
   */
  var $tpl_name_ref=null;

  /**
   * Array with references to the subtemplates.
   * The subtemplates are referenced by the value of an attribute "name" of their parent template.
   * @var   array
   */
  var $sub_name_ref=null;

  /**
   * 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
   * @var   array
   */
```

```
    var $parsed_name_ref=null;

    /**
     * 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
     * @var   array
     */
    var $parsed_name_flags=null;

    /**
     * Template variables referenced by the template name.
     * @var   array
     */
    var $tpl_vars=null;

    /**
     * All variable names used in all loaded templates
     * @var   array
     */
    var $tpl_vars_plain=null;

    /**
     * Global variables.
     * These variables are overrided by template variables with the same name.
     * @var   array
     */
    var $global_vars=null;






    /**
     * Constructor
     */
    function PcpinTpl() {
      // Reset template structure
      $this->   resetTpl();
    }


    /**
     * Reset template structure
     */
    function resetTpl() {
      $this->   tpl_struct=array();
      $this->   tpl_depth=0;
      $this->   files=array();
      $this->   tpl_name_ref=array();
      $this->   sub_name_ref=array();
      $this->   parsed_name_ref=array();
      $this->   parsed_name_flags=array();
      $this->   tpl_vars=array();
      $this->   tpl_vars_plain=array();
      $this->   global_vars=array();
    }


    /**
     * Set error status
     * @param   string    $errortext    Error text
     */
    function setError($errortext='') {
      $this->   last_error=$errortext;
    }


    /**
     * Get last raised error description
     * @return  string
     */
    function getLastError() {
      return $this->   last_error;
    }


    /**
```

```php
   * Set new base directory
   * @param   string    $dir    Base directory
   * @return  boolean   TRUE, if directory exists and readable, otherwize FALSE
   */
  function setBasedir($dir='') {
    $result=false;
    // Reset error status
    $this->   setError();
    // Convert backslashes into forward slashes
    $dir=str_replace('\\', '/', $dir);
    // Check directory
    if (!file_exists($dir)) {
      // Specified directory does not exists
      $this->   setError('Directory "'     .$dir.'" does not exists'     );
    } elseif (!is_dir($dir)) {
      // Specified resource is not a directory
      $this->   setError('"'     .$dir.'" is not a directory'     );
    } elseif (!is_readable($dir)) {
      // Specified directory is not readable
      $this->   setError('Directory "'     .$dir.'" is not readable'     );
    } else {
      // Everything is OK.
      $result=true;
      $this->   basedir=$dir;
    }
    return $result;
  }


  /**
   * Read template file and parses contained templates
   * @param   string    $file   File name (relative to the base directory)
   * @return  boolean   TRUE on success or FALSE on error
   */
  function readTemplatesFromFile($file='') {
    $result=false;
    // Reset error status
    $this->   setError();
    // Read file contents
    if (false!==$tpl=$this->   getFileContents($file)) {
      // Parse template string into structure
      $cdata_prefix='';
      $cdata_postfix='';
      if (false===$result=$this->   parseIntoStruct($this->   tpl_struct, $this->   tpl_depth, $cdata_prefix, $cdata_postfix, $tpl)) {
        // Template parser error
        echo htmlentities($this->   getLastError());
        die();
      }
      // Create new reference arrays
      $this->   makeRefs($this->   tpl_struct, '');
      array_pop($this->   files);
    }
    return $result;
  }


  /**
   * Read file into a string
   * @param   string    $file   File name (relative to the base directory)
   * @return  mixed   (string) File contents on success or (boolean) FALSE on error
   */
  function getFileContents($file='') {
    $result=false;
    // Reset error status
    $this->   setError();
    // Convert backslashes into forward slashes
    // Create file name with path
    $fn=realpath($this->   basedir.'/'.$file);
    // Check file
    if ($file=='') {
      // Empty filename
      $this->   setError('Template file name is empty');
    } elseif (!file_exists($fn)) {
      // File does not exists
      $this->   setError('Could not open file "'     .$this->   basedir.'/'.$file.'" for reading: file does not exists');
    }else if (!is_file($fn)) {
      // Specified resource is not a file
      $this->   setError('Could not open file "'     .$fn.'" for reading: it is not a file'     );
    }else if (!is_readable($fn)) {
      // File not readable
      $this->setError('Could not open file "'.$fn.'" for reading: file is not readable');
    } else {
      // File exists and readable.
      // Check wether the file were already read (avoiding unterminated recursion)
      $result=true;
      if (!empty($this->files)) {
        $tmp=$this->files;
        foreach ($tmp as $frec) {
          list($name,)=each($frec);
          if ($name==$fn) {
            // File has already been opened
            $this->getLastFileData($fn_old, $offset);
            $result=false;
            $this->setError('Error in file "'.$fn_old.'" at line '.$this->getLineNumber($fn_old, $offset).': the file "'.$fn.'" has already been opened: unterminated recursion detected!');
            break;
          }
        }
      }
      if ($result) {
        // Read file contents
        if (false===$result=file_get_contents($fn)) {
          // Failed to read file contents into string
          $this->setError('Could not open file "'.$fn.'" for reading: file read failure');
      } else {
          // File contents were successfully read
          array_push($this->files, array($fn=>0));
        }
      }
    }
    return $result;
  }


  /**
   * Parse template string into the internal structure
   * @param   array     $tpl_struct     A reference to the template structure array
   * @param   array     $tpl_depth      A reference to the current template structure depth
   * @param   array     $cdata_prefix   A reference to the variable where CDATA between offset 0
   *                                     and first element will be stored
   * @param   array     $cdata_postfix  A reference to the variable where CDATA between last element
   *                                     and end of the file will be stored
   * @param   string    $tpl            A reference to the template string
   * @return  boolean   TRUE on success or FALSE on error
   */
  function parseIntoStruct(&$tpl_struct, &$tpl_depth, &$cdata_prefix, &$cdata_postfix, &$tpl) {
    // Reset error status
    $this->setError();
    $result=false;
    if ($tpl!='') {
      $ns=((PCPIN_TPL_NS!='')? PCPIN_TPL_NS.':' : '');
      // REGEX pattern for matching PCPIN template tags
      $tag_pattern='/(<([ ])*'.$ns.'([A-Za-z0-9_])+([ ])*(([ ])+([A-Za-z0-9_]+)([ ])*=([ ])*"[^"]*"([ ])*([\/])?([ ])*)*>)|(<([ ])*[\/]([ ])*'.$ns.'([A-Za-z0-9_])+([ ])*>)/';
      // Parse tags
      if (false===preg_match_all($tag_pattern, $tpl, $matches, PREG_PATTERN_ORDER|PREG_OFFSET_CAPTURE)) {
        // An unknown error occured
        $this->setError('Template parser internal error');
      } else {
        if (empty($matches[0])) {
          // There are no template tags found.
          $result=true;
          $cdata_prefix=$tpl;
          $tpl_struct=array();
          $cdata_postfix='';
        } else {
          // There are some template tags. Parse.
          $matches=$matches[0];
          $total_elements=count($matches);
          $tag_opened=false;
          $i=0;
          foreach ($matches as $match) {
```

```
$match_0=$match[0];
// This match' offset
$offset=$match[1];
// Is there CDATA before the first element?
if ($i==0 &&        $offset>   0) {
  // There is some CDATA before first element
  $cdata_prefix=substr($tpl, 0, $offset);
}
$thisfiles=array_pop($this->   files);
list($fn,)=each($thisfiles);
array_push($this->   files, array($fn=>   $offset));
// Get first CDATA borders
$cdata_start=$offset+strlen($match_0);
$cdata_end=isset($matches[$i+1])? $matches[$i+1][1] : $cdata_start;
// Prepare string
$match_0=trim(ltrim(rtrim(str_replace($ns, '', $match_0), '>'   ), '<'   ));
// Which tag? (opening|closing|closed)
$slashpos=strpos($match_0, '/');
if (false===$slashpos) {
  // Opening tag
  $tag_type=0;
} elseif ($slashpos>   0) {
  // Closed tag (both start and end elements)
  $tag_type=1;
  $match_0=trim(rtrim($match_0, '/'));
} else {
  // Closing tag
  $tag_type=2;
  $match_0=trim(ltrim($match_0, '/'));
}
// Tag name
$name=substr($match_0, 0, strpos($match_0.' ', ' '));
// Parse attributes
$attrs=array();
if ($tag_type==0 || $tag_type==1) {
  $pattern='/([ ])*[A-Za-z0-9_]*=([ ])*"[^"]*"([ ])*/'              ;
  if (false!==preg_match_all($pattern, $match_0, $attr_matches)) {
    if (!empty($attr_matches[0])) {
      $attr_pairs=$attr_matches[0];
    }
  }
  foreach ($attr_pairs as $attrstr) {
    if ($attrstr!='' &&        $attrstr!='/') {
      list($key, $val)=explode('=', $attrstr);
      $attrs[trim($key)]=substr(trim($val), 1, -1);
    }
  }
}
// Call handlers
if ($tag_type==0) {
  // Opening tag
  // Call start element handler
  if (false===$result=$this->   startElement($tpl_struct, $tpl_depth, $name, $attrs)) {
    // An error occured
    break;
  } else {
    // Get first part of CDATA
    if (0<   $cdata_length=$cdata_end-$cdata_start) {
      if (false===$result=$this->   characterData($tpl_struct, $tpl_depth, substr($tpl, $cdata_start, $cdata_length))) {
        // An error occured
        break;
      }
    }
  }
}else if ($tag_type==1) {
  // Closed tag (both start and end elements)
  // Call start and end element handlers
  if (false===$result=$this->   startElement($tpl_struct, $tpl_depth, $name, $attrs) &&        $this->   endElement($tpl_struct, $tpl_depth, $name)) {
    // An error occured
    break;
  } else {
    // Get next part of CDATA
    if (0<   $cdata_length=$cdata_end-$cdata_start) {
      if (false===$result=$this->   characterData($tpl_struct, $tpl_depth, substr($tpl, $cdata_start, $cdata_length))) {
        // An error occured
        break;
      }
```

```
                }
              }
            } else {
              // Closing tag
              // Call end element handler
              if (false===$result=$this->   endElement($tpl_struct, $tpl_depth, $name)) {
                // An error occured
                break;
              } else {
                // Get next part of CDATA
                if (0<   $cdata_length=$cdata_end-$cdata_start) {
                  if (false===$result=$this->   characterData($tpl_struct, $tpl_depth, substr($tpl,
$cdata_start, $cdata_length))) {
                    // An error occured
                    break;
                  }
                }
              }
            }
            $i++;
            if ($total_elements==$i &&         $result) {
              // Last element
              $cdata_postfix=substr($tpl, $cdata_start);
            }
          }
          if ($result===true &&         $tpl_depth!=0) {
            // Error: tag is still open at the end of file
            $this->   getLastFileData($fn, $offset);
            $this->   setError('Error in file "'     .$fn.'": element
"<'        .$ns.$tpl_struct[$tpl_depth-1]['name'].'>" is still open at the end of file'        );
            $result=false;
          }
        }
      }
    }
    // Optimize $this->tpl_vars_plain array
    $this->   tpl_vars_plain=array_unique($this->   tpl_vars_plain);
    return $result;
  }


  /**
   * Start element handler
   * @param   array     $tpl_struct     A reference to the template structure array
   * @param   array     $tpl_depth      A reference to the current template structure depth
   * @param   string    $name           Element name
   * @param   array     $attrs          Element attributes
   * @return  boolean   TRUE on success or FALSE on error
   */
  function startElement(&    $tpl_struct, &    $tpl_depth, $name, $attrs) {
    $result=false;
    if ($tpl_depth==0 && !empty(        $tpl_struct)) {
      // More than one root element detected
      $this->   getLastFileData($fn, $offset);
      $this->   setError('Error in file "'     .$fn.'" at line '     .$this->   getLineNumber($fn,
$offset).': element "<'        .PCPIN_TPL_FULL_NS.$name.'>" is not allowed here'        );
    } elseif ($name!=PCPIN_TPL_NAME_MAIN &&         $name!=PCPIN_TPL_NAME_SUB) {
      // Unknown element
      $this->   getLastFileData($fn, $offset);
      $this->   setError('Error in file "'     .$fn.'" at line '     .$this->   getLineNumber($fn,
$offset).': unknown element "<'        .PCPIN_TPL_FULL_NS.$name.'>"'        );
    } elseif ($name==PCPIN_TPL_NAME_MAIN &&         $tpl_depth>   0 &&
$tpl_struct[$tpl_depth-1]['template_type']=='condition') {
      // Element with this name is not allowed here
      $this->   getLastFileData($fn, $offset);
      $this->   setError('Error in file "'     .$fn.'" at line '     .$this->   getLineNumber($fn,
$offset).': element "<'        .PCPIN_TPL_FULL_NS.$name.'>" is not allowed here'        );
    } elseif ($name==PCPIN_TPL_NAME_SUB && (        $tpl_depth==0 || $tpl_struct[$tpl_depth-
1]['name']==PCPIN_TPL_NAME_SUB)) {
      // Element with this name is not allowed here
      $this->   getLastFileData($fn, $offset);
      $this->   setError('Error in file "'     .$fn.'" at line '     .$this->   getLineNumber($fn,
$offset).': element "<'        .PCPIN_TPL_FULL_NS.$name.'>" is not allowed here'        );
    } else {
      $result=true;
      // Check SUBtemplate
      if ($name==PCPIN_TPL_NAME_SUB) {
        // Its a subtemplate
        if (!array_key_exists('condition', $attrs)) {
          // A subtemplate requires the "condition" attribute
          $result=false;
          $this->   getLastFileData($fn, $offset);
          $this->   setError('Error in file "'     .$fn.'" at line '     .$this->   getLineNumber($fn, $offset).': a subtemplate requires the "condition" attribute'          );
        } else {
          // The name of the parent template
          $parent_name=(isset($tpl_struct[$tpl_depth-1]['attrs']['name']))? $tpl_struct[$tpl_depth-1]['attrs']['name'] : '';
        }
      }
      if ($result) {
        // Get the value of "name" attribute
        $tpl_name=(isset($attrs['name']))? $attrs['name'] : '';
        // Check template type
        $template_type=(isset($attrs['type']))? $attrs['type'] : '';
        $tpl_vars=array();
        if ($name!=PCPIN_TPL_NAME_SUB) {
          // Check type
          switch($template_type) {
            default                 :   // An unknown type
                                        $result=false;
                                        $this->   getLastFileData($fn, $offset);
                                        $this->   setError('Error in file "'     .$fn.'" at line '.$this->   getLineNumber($fn, $offset).': unknown template type "'     .$template_type.'"'     );
                                        break;
            case ''                 :   // Empty type
                                        // Check template name
                                        if ($tpl_name=='') {
                                          // Required attribute "name" is empty
                                          $result=false;
                                          $this->   getLastFileData($fn, $offset);
                                          $this->   setError('Error in file "'     .$fn.'" at line '.$this->   getLineNumber($fn, $offset).': template of this type requires non-empty "name" attribute');
                                        }
                                        break;
            case 'simplecondition'  :   // type="simplecondition"
                                        // Check template name
                                        if ($tpl_name=='') {
                                          // Required attribute "name" is empty
                                          $result=false;
                                          $this->   getLastFileData($fn, $offset);
                                          $this->   setError('Error in file "'     .$fn.'" at line '.$this->   getLineNumber($fn, $offset).': template of this type requires non-empty "name" attribute');
                                        }
                                        // "simplecondition" template requires non-empty "requiredvars" attribute
                                        if (isset($attrs['requiredvars'])) {
                                          $attrs['requiredvars']=strtoupper(trim($attrs['requiredvars']));
                                        }
                                        if (empty($attrs['requiredvars'])) {
                                          // "requiredvars" attribute is empty or not set
                                          $result=false;
                                          $this->   getLastFileData($fn, $offset);
                                          $this->   setError('Error in file "'     .$fn.'" at line '.$this->   getLineNumber($fn, $offset).': template of type "simplecondition" requires non-empty "requiredvars" attribute'          );
                                        } else {
                                          // Store variable names
                                          $tmp=explode(',', $attrs['requiredvars']);
                                          foreach ($tmp as $var) {
                                            $var=trim($var);
                                            if ($var!='') {
                                              $tpl_vars[$var]=null;
                                            }
                                          }
                                        }
                                        break;
            case 'condition'  :         // type="condition"
                                        // "condition" template requires non-empty "conditionvar" attribute
                                        if (isset($attrs['conditionvar'])) {
                                          $attrs['conditionvar']=strtoupper(trim($attrs['conditionvar']));
                                        }
                                        if (empty($attrs['conditionvar'])) {
                                          // "conditionvar" attribute is empty or not set
                                          $result=false;
```

```
                                                $this->   getLastFileData($fn, $offset);
                                                $this->   setError('Error in file "'     .$fn.'" at line '.$this->   getLineNumber($fn, $offset).': template of type "condition" requires non-empty "conditionvar" attribute'         );
                                              } else {
                                                // Store variable name
                                                $tpl_vars[$attrs['conditionvar']]=null;
                                              }
                                              break;
                papers}
              }
            }
            if ($result) {
              // Template type is OK
              $this->   tpl_vars[$tpl_name]=$tpl_vars;
              $this->   parsed_name_ref[$tpl_name]='';
              $this->   parsed_name_flags[$tpl_name]=false;
              $children=array();
              $child_types=array();
              if (isset($attrs['src'])) {
                // The template has an external source (included)
                if (false===$tpl=$this->   getFileContents($attrs['src'])) {
                  // Failed to read template from file
                  $result=false;
                } else {
                  // Parse included template source
                  $element_start_src='<'   .PCPIN_TPL_FULL_NS.$name;
                  foreach ($attrs as $attr_key=>   $attr_val) {
                    if ($attr_key!='src') {
                      $element_start_src.=' '.$attr_key.'="'     .$attr_val.'"'     ;
                    }
                  }
                  $element_start_src.='>'   ;
                  $element_end_src='</'   .PCPIN_TPL_FULL_NS.$name.'>'   ;
                  $tpl=$element_start_src.$tpl.$element_end_src;
                  $tpl_struct_sub=array();
                  $tpl_depth_sub=0;
                  $cdata_prefix='';
                  $cdata_postfix='';
                  if (false!==$result=$this->   parseIntoStruct($tpl_struct_sub, $tpl_depth_sub, $cdata_prefix, $cdata_postfix, $tpl)) {
                    if ($cdata_prefix!='') {
                      // Included template has CDATA before the first element
                      array_push($children, $cdata_prefix);
                      array_push($child_types, 1);
                    }
                    if (!empty($tpl_struct_sub)) {
                      array_push($children, $tpl_struct_sub);
                      array_push($child_types, 0);
                    }
                    if ($cdata_postfix!='') {
                      // Included template has CDATA after the last element
                      array_push($children, $cdata_postfix);
                      array_push($child_types, 1);
                    }
                  }
                  array_pop($this->   files);
                }
              }
              if ($result) {
                $node=array('name'=>   $name,
                            'template_type'=>   $template_type, // If a template, then here is the value of "type" attrbute, if any
                            'attrs'=>   $attrs,
                            'child_types'=>   $child_types, // 0: template record (array), 1: cdata (string)
                            'children'=>   $children
                            );
                $tpl_depth++;
                array_push($tpl_struct, $node);
              }
            }
          }
          return $result;
        }


        /**
         * Characted data handler
         * @param   array     $tpl_struct     A reference to the template structure array
```

```php
   * @param   array     $tpl_depth      A reference to the current template structure depth
   * @param   string    $cdata          Character data
   * @return  boolean   TRUE on success or FALSE on error
   */
  function characterData(&$tpl_struct, &$tpl_depth, $cdata='') {
   $result=false;
   $tpl_depth_dec=$tpl_depth-1;
   if (!empty($tpl_depth) && $tpl_struct[$tpl_depth_dec]['template_type']=='condition' && trim($cdata)!='') {
    // Character data is not allowed here
    $this->getLastFileData($fn, $offset);
    $this->setError('Error in file "'.$fn.'" at line '.$this->getLineNumber($fn, $offset).': character data is not allowed here');
   } else {
    $result=true;
    if ($tpl_depth_dec>=0 && $tpl_struct[$tpl_depth_dec]['template_type']!='condition') {
     // Condition template does needs even empty CDATA
     $tpl_struct[$tpl_depth_dec]['child_types'][]=1; // CDATA
     array_push($tpl_struct[$tpl_depth_dec]['children'], $cdata);
    }
   }
   return $result;
  }


  /**
   * End element handler
   * @param   array     $tpl_struct     A reference to the template structure array
   * @param   array     $tpl_depth      A reference to the current template structure depth
   * @param   string    $name           Element name
   * @return  boolean   TRUE on success or FALSE on error
   */
  function endElement(&$tpl_struct, &$tpl_depth, $name) {
   $result=false;
   $hierarchy_error=false;
   if (!isset($tpl_struct[0])) {
    // Closing tag that is not opened
    $this->getLastFileData($fn, $offset);
    $this->setError('Error in file "'.$fn.'" at line '.$this->getLineNumber($fn, $offset).': closing tag that is not opened');
   } else {
    $node=array_pop($tpl_struct);
    if ($node['name']!=$name) {
     // Closing tag that is not opened
     $this->getLastFileData($fn, $offset);
     $this->setError('Error in file "'.$fn.'" at line '.$this->getLineNumber($fn, $offset).': wrong closing tag ("</'.PCPIN_TPL_FULL_NS.$node['name'].'>" expected)');
    } else {
     $result=true;
     $tpl_depth--;
     $tpl_depth_dec=$tpl_depth-1;
     if ($tpl_depth>0) {
      $tpl_struct[$tpl_depth_dec]['child_types'][]=0; // Node
      array_push($tpl_struct[$tpl_depth_dec]['children'], $node);
     } else {
      // Root element
      // Trim root element' CDATA, if needed
      if (PCPIN_TPL_TRIM_ROOT && !empty($node['child_types'])) {
       if ($node['child_types'][0]==1) {
        if ("\r\n"==substr($node['children'][0], 0, 2)) {
         $node['children'][0]=substr($node['children'][0], 2);
        } elseif ("\r"==substr($node['children'][0], 0, 1) || "\n"==substr($node['children'][0], 0, 1)) {
         $node['children'][0]=substr($node['children'][0], 1);
        }
       }
       $last_child=count($node['child_types'])-1;
       if ($last_child>0 && $node['child_types'][$last_child]==1) {
        if ("\r\n"==substr($node['children'][$last_child], -2)) {
         $node['children'][$last_child]=substr($node['children'][$last_child], 0, -2);
        } elseif ("\r"==substr($node['children'][$last_child], -1) || "\n"==substr($node['children'][$last_child], -1)) {
         $node['children'][$last_child]=substr($node['children'][$last_child], 1, -1);
        }
       }
      }
      $tpl_struct=$node;
     }
```

```
        // Get variables from node's CDATA elements
        $tpl_name='';
        if ($name==PCPIN_TPL_NAME_SUB) {
          // A subtemplate
          $tpl_name=$tpl_struct[$tpl_depth_dec]['attrs']['name'];
        } elseif (isset($node['attrs']['name'])) {
          $tpl_name=$node['attrs']['name'];
        }
        if ($tpl_name!='') {
          $cdata='';
          if (!empty($node['child_types'])) {
            foreach ($node['child_types'] as $key=>   $type) {
              if ($type==1) {
                $cdata.=$node['children'][$key];
              }
            }
          }
          if ($cdata!='') {
            // Extract variables from CDATA
            if (false!==preg_match_all('/{[^{}]*}/', $cdata, $matches)) {
              if (!empty($matches[0])) {
                $matches=$matches[0];
                foreach ($matches as $match) {
                  $var_name=trim(substr($match, 1, -1));
                  if ($var_name!='') {
                    $this->   tpl_vars[$tpl_name][$var_name]=null;
                    $this->   tpl_vars_plain[]=$var_name;
                  }
                }
              }
            }
          }
        }
      }
    }
    return $result;
  }


  /**
   * returns name and byte offset of the current opened file
   * @param   string    $filename   A reference to the variable where file name will be stored
   * @param   array     $offset     A reference to the variable where byte offset will be stored
   */
  function getLastFileData(&    $filename, &    $offset) {
    if (!empty($this->   files)) {
      $tmp=$this->   files;
      list($filename, $offset)=each(end($tmp));
    } else {
      $filename='';
      $offset=0;
    }
  }


  /**
   * Create new name reference arrays
   * @param   array   Element to start with
   */
  function makeRefs(&    $root, $parent_tpl_name='') {
    if (!empty($root) &&         is_array($root)) {
      if ($root['name']==PCPIN_TPL_NAME_SUB) {
        // Element is a subtemplate
        if (!isset($this->   sub_name_ref[$parent_tpl_name])) {
          $this->   sub_name_ref[$parent_tpl_name]=array();
        }
        $this->   sub_name_ref[$parent_tpl_name][]=&    $root;
      } else {
        // Element is a template
        $parent_tpl_name=$root['attrs']['name'];
        $this->   tpl_name_ref[$parent_tpl_name]=&    $root;
      }
      // Children?
      foreach ($root['child_types'] as $key=>   $type) {
        if ($type==0) {
          $this->   makeRefs($root['children'][$key], $parent_tpl_name);
        }
      }
    }
  }
```

```php


    /**
     * Get number of line at which the character with specified offset is located
     * @param   string    $filename   File name to search in
     * @param   int       $char       Character offset
     * @return  int
     */
    function getLineNumber($filename='', $char=0) {
      $result=0;
      if (false!==$h=file($filename)) {
        $i=0;
        foreach ($h as $line_nr=>   $str) {
          $i+=strlen($str);
          if ($i>   $char) {
            $result=$line_nr+1;
            break;
          }
        }
      }
      return $result;
    }


    /**
     * Add a variable to the template
     * @param   string    $template   Template name
     * @param   string    $var_name   Variable name
     * @param   mixed     $var_val    Variable value
     */
    function addVar($template='', $var_name='', $var_val=null) {
      $var_name=strtoupper(trim($var_name));
      if ($var_name!='' && isset(        $this->   tpl_vars[$template]) &&
array_key_exists($var_name, $this->   tpl_vars[$template])) {
        $this->   tpl_vars[$template][$var_name]=$var_val;
      }
    }


    /**
     * Add multiple variables to the template.
     * The $vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)
     * @param   string    $template   Template name
     * @param   array     $vars       Variables to add
     */
    function addVars($template='', $vars=null) {
      if (!empty($vars) &&         is_array($vars)) {
        foreach ($vars as $key=>   $val) {
          $this->   addVar($template, $key, $val);
        }
      }
    }


    /**
     * Add a global variable
     * @param   string    $var_name   Variable name
     * @param   mixed     $var_val    Variable value
     */
    function addGlobalVar($var_name='', $var_val=null) {
      $var_name=strtoupper(trim($var_name));
      if ($var_name!='') {
        $this->   global_vars[$var_name]=$var_val;
      }
    }


    /**
     * Add multiple global variables
     * The $vars array elements have variable name as KEY and it's value as VAL (KEY=>VAL)
     * @param   array     $vars       Variables to add
     */
    function addGlobalVars($vars=null) {
      if (!empty($vars) &&         is_array($vars)) {
        foreach ($vars as $key=>   $val) {
          $this->   addGlobalVar($key, $val);
        }
      }
    }


    /**
     * Parse template
     * @param   string    $name     Template name
     * @param   string    $mode     Parse mode
     */
    function parseTemplate($name='', $mode=PCPIN_TPL_DEFAULT_PARSE_MODE) {
      if ($name!='' && isset(        $this->   tpl_name_ref[$name])) {
        $this->   parsed_name_flags[$name]=true;
        if ($mode==PCPIN_TPL_PARSE_MODE_OVERWRITE) {
          $this->   parsed_name_ref[$name]='';
        }
        // Parse template
        $this->   parsed_name_ref[$name].=$this->   parseIntoString($this->   tpl_name_ref[$name],
$this->   tpl_vars[$name]);
      }
    }


    /**
     * Parse template (if not parsed yet) and return it's parsed contents as string
     * @param   string    $name     Template name
     * @return  string
     */
    function getParsedTemplate($name='') {
      $result='';
      if ($name=='') {
        if (isset($this->   tpl_struct['attrs'])) {
          if (isset($this->   tpl_struct['attrs']['name'])) {
            $name=$this->   tpl_struct['attrs']['name'];
          }
        }
      }
      if ($name!='' && isset(        $this->   tpl_name_ref[$name])) {
        if (false===$this->   parsed_name_flags[$name]) {
          // Parse template
          $this->   parseTemplate($name);
        }
        $result=$this->   parsed_name_ref[$name];
      }
      return $result;
    }


    /**
     * Parse template structure and return it's parsed contents as a string
     * @param   array     $tpl          Template record
     * @param   array     $vars         Template variables
     * @return  string
     */
    function parseIntoString($tpl, $vars) {
      $parsed_string='';
      if (is_array($tpl) && !empty(        $tpl)) {
        $child_key=-1;
        if ($tpl['name']==PCPIN_TPL_NAME_SUB) {
          // Subtemplate (must be parsed)
          $parse=true;
        } else {
          // Check template
          $parse=false;
          $tpl_name=$tpl['attrs']['name'];
          // Which type is the template of?
          if (empty($tpl['template_type'])) {
            // A simple template. Parse without conditions.
            $parse=true;
          } elseif ($tpl['template_type']=='simplecondition') {
            // A simple condition template. Check condition.
            $parse=true;
            if (!empty($tpl['attrs']['requiredvars'])) {
              $requiredvars=explode(',', $tpl['attrs']['requiredvars']);
              // Check each var
              foreach ($requiredvars as $var) {
                $var=trim($var);
                if ($var!='' && empty(        $this->   tpl_vars[$tpl_name][$var])) {
                  // At least one of the required vars is empty. Do not parse.
                  $parse=false;
                  break;
                }
              }
            }
```

```php
          } elseif ($tpl['template_type']=='condition') {
            // A conditional template. Check subtemplates.
            $conditionvar=$this->tpl_vars[$tpl_name][$tpl['attrs']['conditionvar']];
            if (!empty($tpl['child_types'])) {
              foreach ($tpl['child_types'] as $key=>$type) {
                if ($type==0) {
                  $child=$tpl['children'][$key];
                  if (isset($child['attrs']['condition'])) {
                    $child_condition=$child['attrs']['condition'];
                    if ($child_condition=='default') {
                      // One of subtemplates is a default template
                      $parse=true;
                      $child_key=$key;
                    } elseif ($child_condition=='empty' && empty($conditionvar)) {
                      // Condition variable has an empty value and one of subtemplates has an empty condition.
                      $parse=true;
                      $child_key=$key;
                      break;
                    } elseif ($child_condition!='default' && $child_condition!='empty' && (string) $child_condition==(string)$conditionvar) {
                      // One of the subtemplates has the same condition as the condition variable's value
                      $parse=true;
                      $child_key=$key;
                      break;
                    }
                  }
                }
              }
            }
          }
        }
        if ($parse) {
          // Parse template children.
          if ($child_key>=0) {
            // Parse only one child (a subtemplate)
            $parsed_string.=$this->parseIntoString($tpl['children'][$child_key], $vars);
          } else {
            // Parse all children
            if (!empty($tpl['child_types'])) {
              foreach ($tpl['child_types'] as $key=>$type) {
                // Which type is the child of?
                if ($type==1) {
                  // A CDATA
                  // Pass variables and store.
                  $parsed_string.=$this->passVars($tpl['children'][$key], $vars);
                } elseif ($type==0) {
                  // A template
                  $parsed_string.=$this->getParsedTemplate($tpl['children'][$key]['attrs']['name']);
                }
              }
            }
          }
        }
      }
      return $parsed_string;
    }


    /**
     * Pass variables to the parsed template string.
     * Global variables will be passed too.
     * @param   string    $parsed   Parsed template string
     * @param   array     $vars     Variables to pass
     * @return  string
     */
    function passVars($parsed='', $vars=null) {
      if ($parsed!='' && !empty($vars) && is_array($vars)) {
        // Replace '{' characters in the values in order to avoid wrong name-value replacements
        $replacement=chr(0).'pcpin'.chr(0);
        $replaced=false;
        // Add global vars
        if (!empty($this->global_vars)) {
          foreach ($this->global_vars as $var_name=>$var_value) {
            if (empty($vars[$var_name])) {
              if (!is_scalar($var_value)) {
                $var_value='';
              } elseif (!empty($var_value) && false!==strpos($var_value, '{')) {
```

```
              $replaced=true;
              $var_value=str_replace('{', $replacement, $var_value);
            }
            $parsed=str_replace('{'.$var_name.'}', $var_value, $parsed);
          }
        }
      }
      // Add local vars
      foreach ($vars as $var_name=>   $var_value) {
        if (!is_scalar($var_value)) {
          $var_value='';
        } elseif (!empty($var_value) &&        false!==strpos($var_value, '{')) {
          $replaced=true;
          $var_value=str_replace('{', $replacement, $var_value);
        }
        $parsed=str_replace('{'.$var_name.'}', $var_value, $parsed);
      }
      if ($replaced) {
        $parsed=str_replace($replacement, '{', $parsed);
      }
    }
    return $parsed;
  }


  /**
   * Display parsed template. If the template is not parsed yet, then it will be parsed.
   * @param   string    $name     Template name
   */
  function displayParsedTemplate($name='') {
    echo $this->   getParsedTemplate($name);
  }


  /**
   * Clear template variables and parsed contents of a template
   * @param   string    $name     Template name
   */
  function clearTemplate($name='') {
    if ($name=='') {
      if (isset($this->   tpl_struct['attrs'])) {
        if (isset($this->   tpl_struct['attrs']['name'])) {
          $name=$this->   tpl_struct['attrs']['name'];
        }
      }
    }
    if ($name!='' && isset(        $this->   tpl_name_ref[$name])) {
      $this->   parsed_name_ref[$name]='';
      $this->   parsed_name_flags[$name]=false;
      foreach ($this->   tpl_vars[$name] as $key=>   $val) {
        $this->   tpl_vars[$name][$key]=null;
      }
    }
  }


}
?>
```

Index