DirecTemplate
Template Author's Guide

Table of Contents

1. Introduction....4
   1.1. About DirecTemplate....4
   1.2. About this Guide....4
2. Template Basics....4
   2.1. Basic (Scalar) Variables....4
   2.2. Array and Object Variables....5
   2.3. Conditional Expressions....5
      2.3.1. Basic Conditional Expressions....5
      2.3.2. Advanced Conditional Expressions....5
      2.3.3. Compound Conditional Expressions....6
   2.4. Loops....7
3. Transformations....8
   3.1. Introduction....8
   3.2. Transformation Reference....8
      3.2.1. PHP Function Equivalents....8
      3.2.2. count....9
      3.2.3. yesno....9
      3.2.4. truefalse....9
      3.2.5. null....9
      3.2.6. noquotes....9
      3.2.7. quoteconvert....9
      3.2.8. nbsp....9
      3.2.9. nl2br....9
      3.2.10. base64_encode....10
      3.2.11. strpad....10
      3.2.12. currency....10
      3.2.13. chr....10
      3.2.14. ord....10
      3.2.15. mysqldatetime....10
      3.2.16. mysqltimestamp....11
      3.2.17. unixtimestamp....11
      3.2.18. timetext....11
      3.2.19. sprintf....11
      3.2.20. substr....12
      3.2.21. contains....12
      3.2.22. wordwrap....12
      3.2.23. str_replace....13
      3.2.24. preg_replace....13
      3.2.25. preg_match....13
      3.2.26. Arithmetic Transformations....14
      3.2.27. in_array....14
      3.2.28. in_keys....14
      3.2.29. in_set....14
      3.2.30. ellipsis....15
      3.2.31. midellipsis....15
      3.2.32. dateselect....16

3.3. Non-Variable Transformations....17
3.3.1. random....17
3.3.2. gettime....17
3.3.3. range....17
4. Includes....18
5. Multiple-Page Result Sets....18
6. Advanced Topics....19
6.1. Adding Custom Functionality....19
6.1.1. Custom Plugins....19
6.1.2. Custom Transformations....20
6.1.3. Custom Filters....20
6.2. Override and Fallback Template Directories....21
6.3. Assigning Variables from Within Templates....22
7. Credits....23

1. Introduction

1.1. About DirecTemplate

DirecTemplate is designed to be a fast, robust, regular-expression-based template engine.

Regular expressions offer great speed and flexibility while still allowing DirecTemplate to operate without disk write permissions (unlike other engines that take a compile-to-PHP-code approach to template parsing). While caching is supported, it is optional and DirecTemplate will happily operate without write access to any disk.

1.2. About this Guide

This guide is intended for template authors who are already familiar with template engines and HTML, and have at least some familiarity with the PHP scripting language. Users will find that experience with other template engines such as *Smarty* is an asset, as DirecTemplate uses many of the same conventions in its template language.

2. Template Basics

2.1. Basic (Scalar) Variables

The fundamental purpose of any template library is to take variable values from an application (in this case, a PHP script), and insert them into an HTML document (your template).

In DirecTemplate, variables are assigned in PHP using the `assign()` method of the `Template` object. For example:

```
$tpl->assign("myvar","value");
```

A variable assigned in this manner can be accessed as `{$myvar}` in a template. For example, the following might be used in a template file:

```
This is a sample {$myvar}.
```

This example would cause DirecTemplate to output the following:

```
This is a sample value.
```

2.2. Array and Object Variables

DirecTemplate supports the use of a variety of variable types in templates. Elements of associative arrays and member variables of objects can be accessed using the *dot operator*. For example, the following code could be used to assign an associative array to a template variable:

```
$myarray = array('name'=>'Fred');
$tpl->assign('myarray',$myarray)
```

Once the array is assigned to a variable as per above, the elements of this array can be accessed using the *dot operator*, by specifying the name of the variable, followed by a dot, followed by the key of the array element. For example:

```
{$myarray.name}
```

Member variables of objects assigned to template variables may be accessed in the same manner.

2.3. Conditional Expressions

2.3.1. Basic Conditional Expressions

Conditional expressions, along with the `if` control structure, allow the template author to selectively display or hide content based on the result of a comparison between two values. For example:

```
{if $users>300}
     Too many users.
{/if}
```

In this example, if the value of the `$users` variable is greater than 300, the message `Too many users` will be displayed.

DirecTemplate supports the equality or “equal-to” operator (==), greater-than and greater-than-or-equal-to operators (> and >=), less-than and less-than-or-equal-to operators (< and <=) and the negative-equality or “not-equal-to” operator (!=). PHP's equality-and-same-type operator (===) is also supported.

2.3.2. Advanced Conditional Expressions

The `if` control structure also allows the use of `else` to display alternate content when the conditional expression evaluates to false. For example:

```
{if $age<18}
     Child
```

```
{else}
     Adult
{/if}
```

In this example, if the value of the `$age` variable is less than 18, the word `Child` will be displayed. Otherwise, the word `Adult` will be displayed.

Multiple conditions can also be evaluated through the use of `elseif`. For example:

```
{if $hour<12}
     Morning
{elseif $hour<18}
     Evening
{else}
     Night
{/if}
```

In this example, DirecTemplate first checks the value of `$hour` to determine if it is less than 12. If so, it displays the word `Morning` and ignores the rest of the `if` statement. If `$hour` is *not* less than 12, however, it proceeds to check the value of `$hour` to determine whether it is less than 18. If the value is less than 18, the word `Evening` is displayed and the rest of the `if` statement is ignored. Finally, if neither of the first two conditions evaluated true, DirecTemplate displays the word `Night` from the `else` clause.

Note that because DirecTemplate was designed to be based solely upon regular expressions, nested `if` statements are not supported. The `check` statement is a synonym for `if`, however, and a `check` statement can be nested inside an `if` statement and vice-versa. When using `check`, the `otherwisecheck` and `otherwise` statements are used instead of `elseif` and `else`, respectively.

2.3.3. Compound Conditional Expressions

DirecTemplate provides limited support for compound statements ("and" and "or"). For example:

```
{if ($var1==$var2) or ($var3==$var4)}
     match
{/if}
{if ($var1==$var2) and ($var3==$var4)}
     match
{/if}
```

Note that each condition MUST be enclosed in parentheses, and conditions are evaluated from left to right without any notion of operator precedence. Nested conditions (and nested parentheses) are not

supported.

2.4. Loops

Loops allow a portion of a template to be repeated a certain number of times – once for each element in an array. This is particularly useful for displaying tables or lists in a tmeplate.

Loops are structured as follows:

```
{loop $iterator=$myvar}
This element of myvar is: {$iterator}.
The key is: {$_loop.key}.
{/loop}
```

In the above example, `$myvar` represents an array variable that must be assigned from within PHP. For each element in `$myvar`, DirecTemplate will evaluate the content inside the `{$loop...}{/loop}` block one time. For each iteration of the loop, the `$iterator` variable will be assigned to the value of the next element in `$myvar`.

For example, `$myvar` might be assigned as follows from within PHP:

```
$myvar = array('name'=>'Jim Jones','age'=>32);
$tpl->assign('myvar',$myvar);
```

In this example, DirecTemplate would return the following output using the loop above:

```
This element of myvar is: Jim Jones.
The key is: name.
This element of myvar is: 32.
The key is: age.
```

Inside loops, the value of the current element is assigned to `$iterator`, and the `$_loop` variable contains an array of various information about the current iteration of the loop.

Elements of `$_loop` include:

- `$_loop.key` - current key in the array
- `$_loop.iteration` - current iteration of the loop (starting at 1)
- `$_loop.iterations` - total number of iterations to be completed
- `$_loop.odd` - false if iteration is evenly divisible by 2
- `$_loop.first` - true if this is the first iteration of the loop
- `$_loop.last` - true if this is the last iteration of the loop

Note that because DirecTemplate was designed to be based solely upon regular expressions, nested "loop" statements are not supported.

3. Transformations

3.1. Introduction

Transformations are similar to functions which return a modified version of a template variable. They are invoked by appending a pipe (|) to a variable name, followed by the name of the transformation to call:

```
{$variable|transformation}
```

For example:

```
{$myvar|strtoupper}
```

Using the above in a template would be equivalent to the following PHP code:

```
echo strtoupper($variable);
```

Some transformations accept arguments (similar to a PHP function) which will modify their behavior. Arguments are specified by adding a colon (:) after the transformation name, followed by the argument value. For example:

```
{$variable|substr:2:7}
```

Using the above in a template would be equivalent to the following PHP code:

```
echo substr($variable,2,7);
```

3.2. Transformation Reference

DirecTemplate supports a significant number of transformations. Each transformation is described below.

3.2.1. PHP Function Equivalents

The following transformations provide functionality identical to the PHP functions of the same names:

```
ucfirst
strtolower
strtoupper
ucwords
urlencode
urldecode
htmlentities
addslashes
stripslashes
abs
round
floor
ceil
striptags
trim
```

3.2.2. count

Returns the number of elements if variable is an array, or 0 if not an array.

3.2.3. yesno

Outputs `Yes` if contents are non-null, non-zero, or true, otherwise displays `No`.

3.2.4. truefalse

Identical to `yesno`, but outputs `True` or `False`.

3.2.5. null

Returns the unmodified value of the variable.

3.2.6. noquotes

Replaces all quotes (`"`) with: `&quot;`

3.2.7. quoteconvert

Replaces all quotes (`"`) with double-apostrophes (`''`).

3.2.8. nbsp

Replaces all spaces with non-breakable space entities (` `).

3.2.9. nl2br

Replaces all newlines with HTML break tags (`<br />`).

3.2.10. base64_encode

Encodes the variable using BASE64 encoding.

3.2.11. strpad

Returns the variable padded with the specified character to the specified number of places.

Accepts two arguments:
1. the character to use for padding
2. the padding width; use a positive value for left-padding, or negative for right-padding

Example:

```
{$var|strpad:X:10}
```

In this example, `$var` is padded to `10` spaces with the letter `X`. If `$var` contains the string `test`, the output would be:

```
testXXXXXX
```

3.2.12. currency

Returns the variable in currency format.

The default currency format is `%.2f`. Assign the `CURRENCY_FORMAT` constant in PHP to define an alternate currency format, using the same format as the PHP `sprintf` function.

3.2.13. chr

Returns the character value of the ordinal value in the variable.

3.2.14. ord

Returns the ordinal value of the variable.

3.2.15. mysqldatetime

Converts a string from a MySQL `DATETIME` field to a more presentable format.

Accepts 2 arguments:
1. the format method:
   - `verbose` = output date using words
   - `numeric` = output date using digits
   - `formatted` = use a format string (compatible with PHP's `date()` function)
2. the display method:

- `full` = display the date/time
- `date` = display the date only
- `time` = display the time only
- or, if the format method is `formatted`, this argument should specify the format string used to display the date

Example:

```
{$myvar|mysqldatetime:numeric:date}
{$myvar|mysqldatetime:formatted:M_d,_H:ia}
```

If `$myvar` contains a MySQL `DATETIME` value representing January 1st, 2007 at 22:00, the above example will return:

```
01/01/2007
Jan 01, 10:00pm
```

3.2.16. mysqltimestamp

Same as `mysqldatetime`, but accepts a MySQL `TIMESTAMP` value.

3.2.17. unixtimestamp

Same as `mysqldatetime`, but accepts a UNIX timestamp.

3.2.18. timetext

Returns a textual representation of a variable containing a number of seconds. The output is returned in days, hours, minutes, and seconds.

3.2.19. sprintf

Returns the variable value formatted with the specified format string.

Accepts one argument:

1. the format string; this should be in the same format accepted by the PHP `sprintf()` function

Example:

```
{$var|sprintf:"$%.2f"}
```

If `$var` contains a value of 27.5, the above example will output:

```
27.50
```

3.2.20. substr

Returns a (substring) portion of the variable value.

Accepts one or two arguments:
1. the starting position from which the substring will be copied
2. (optional) the number of characters to copy

Example:

```
{$var|substr:3}
{$var|substr:4:5}
```

If `$var` contains the string `incomprehensible`, the above example will output the following:

```
comprehensible
ompre
```

3.2.21. contains

Determines whether a substring appears in a string.

Accepts one argument:
1. the substring to test for

For example, `{$var|contains:"text"}` will return TRUE if `$var` contains the word `text`.

3.2.22. wordwrap

Returns the content variable wordwrapped at the specified width, using the specified break character.

Accepts up to three arguments:
1. (optional) the width at which to wrap the variable
2. (optional) the break character to append to each segment
3. (optional) if the third argument is set to the word “filename”, the variable will be wrapped at forward-slash (/) characters instead of spaces

Example:

```
{$var|wordwrap:10:"<br>"}
```

If `$var` contains the string `testing the word wrap feature`, the above example will

return:

```
testing the<br>word wrap<br>feature
```

3.2.23. str_replace

Performs a search/replace on the value of the variable. Be sure to use quotes around string literals.

Accepts two arguments:
1. the string to search for
2. the replacement string

Example:
```
{$var|str_replace:"%%SOMETHING%%":"something else"}
{$var|str_replace:"%%SOMETHING%%":$somethingelse}
```

If $var contains the string `This is %%SOMETHING%%` and `$somethingelse` contains `excellent`, the above example will return:

```
This is something else
This is excellent
```

3.2.24. preg_replace

Performs a regular expression search/replace on the value of the variable.

Accepts two arguments:
1. the regular expression to search for
2. the replacement string

Example:
```
{$var|preg_replace:/beer[\.]+/:donuts!}
```

If $var contains the string `Mmm.... beer...`, the above example will return:

```
Mmmm.... donuts!
```

Note: the regular expression is urldecoded before being processed. As such, if you need to use a colon (:) character in your regular expression, use `%3A` instead. If you need a percent (%) character, use `%21` instead, and so-on.

3.2.25. preg_match

Identical in syntax to `preg_replace`, except it returns `1` if the variable value matched the

regular expression, or `0` if not.

3.2.26. Arithmetic Transformations

These transformations perform arithmetic operations on the variable value.

Each transformation accepts one argument:

1. the value to add/subtract/etc. to or from the variable value

The following transformations are available:

- mod – divides the variable value by the argument value, and returns the remainder
- add – adds the variable value to the argument value, and returns the sum
- sub – subtracts the argument value from the variable value, and returns the difference
- mul – multiplies the variable value by the argument value, and returns the product
- div – divides the variable value by the argument value, and returns the quotient
- and – returns the value of a bitwise and between the variable and argument values
- or – returns the value of a bitwise or between the variable and argument values
- xor – returns the value of a bitwise xor between the variable and argument values

Example:

```
{$var|add:2}
```

If the value of `$var` is 40, the above example will return:

```
42
```

3.2.27. in_array

Determines whether the variable value appears in the specified array.

Accepts one argument:

1. the array to search for the variable value

Returns true if the value exists in the array, otherwise false.

3.2.28. in_keys

Identical in syntax to `in_array` , but determines whether `$var` is a key in the specified array.

3.2.29. in_set

Determines whether the variable value appears in the list of arguments provided.

Accepts one or more arguments, indicating the list of values to search.

Example:

```
{$var|in_set:1:3:5}
```

The output of the above example is true if `$var` contains a value of `1`, `3`, or `5`.

3.2.30. ellipsis

Returns the variable value truncated to the specified length, with an ellipsis appended if the value was truncated.

Accepts one argument:
1. the maximum length for the output string

Example:

```
{$var|ellipsis:8}
```

If the value of `$var` is `Teststring`, the example above will return:

```
Tests...
```

3.2.31. midellipsis

Returns the variable value truncated to the specified length, with an ellipsis in the middle (if necessary), and the specified number of trailing characters intact

Accepts up to two arguments:
1. the maximum length for the output string
2. the number of characters to leave intact at the end of the string

Example:

```
{$var|midellipsis:8:2}
```

If the value of `$var` is `Teststring`, the example above will return:

```
Tes...ng
```

For filenames, you can optionally use the word `filename` instead of the trailing character count to keep the filename at the end intact.

Example:

```
{$var|midellipsis:18:filename}
```

If the value of `$var` is `/var/www/html/test.html`, the above example will return:

```
/var/.../test.html
```

3.2.32. dateselect

Returns a `<SELECT>` tag suitable for choosing a date/time value.

Accepts the following arguments:

- base name - specifies the `NAME` attribute for the `<SELECT>` tags
- date type - specifies the format of the input variable; one of:
  - `datetime` - a MySQL `DATETIME` value
  - `timestamp` - a MySQL `TIMESTAMP` value
  - `unix` - a Unix timestamp
  - `null` - an empty date (all `<SELECT>` fields set to "--")
  - `now` - the current time (input variable's value is ignored)

After the first two arguments, any of the following values may be appended as arguments in any order:

- `allownull` - indicates that users can specify null dates
- `showtime` - indicates that `<SELECT>` fields for time should be shown
- `nomonth` - hide the month `<SELECT>` field
- `noday` - hide the day `<SELECT>` field
- `noyear` - hide the year `<SELECT>` field
- `timeonly` - do not show date fields
- `houronly` - do not show minutes
- `ampm` - show AM/PM select field
- `minyear=x` - indicates the minimum year for the year `<SELECT>` field
- `maxyear=x` - indicates the maximum year for the year `<SELECT>` field

Example:

```
{$var|dateselect:basename:datetime:allownull:showtime:
     noday:minyear=2003:maxyear=2007}
```

The above example creates a `<SELECT>` tag whose `NAME` attribute begins with `basename`,

which is populated with the `datetime`-formatted date in `$var`. Null dates (eg: `0000-00-00`) are permitted, hour/minute fields are provided, no day field is shown, and the year field offers values from `2003` to `2007`.

The `parse_date_select()` method can be used in the associated PHP code with the matching `basename` value to parse the submitted date and obtain a UNIX timestamp. For example:

```
$unixtimestamp = $tpl->parse_date_select('basename');
echo "Date submitted: ".date('Y-m-d',$unixtimestamp);
```

3.3. Non-Variable Transformations

Some transformations do not operate on variables, but instead just act as function calls. These transformations may be invoked using:

```
{$_|transformation}
```

Non-variable transformations currently available include:

3.3.1. random

Outputs a random number.

3.3.2. gettime

Same as `mysqldatetime`, but works with the current time.

3.3.3. range

Returns an array containing the specified numeric range.

Accepts two arguments:

1. the lower bound of the range
2. the upper bound of the range

Example:

```
{$_|range:5:10}
```

This example returns an array containing the values `(5,6,7,8,9,10)`.

This is useful for on-the-fly loops, eg:

```
{loop $counter=$_|range:1:5}{$counter}{/loop}
```

The above example would display the numbers 1 through 5.

4. Includes

Use `{include "filename.tpl"}` to parse the template `filename.tpl` and include its output inside the current template.

The included template must be in the same directory as the calling template.

5. Multiple-Page Result Sets

DirecTemplate has internal support for displaying sets of data (eg: result sets from a database server) spanning multiple pages, with a "search engine" style set of page navigation links.

Multiple page result sets must be prepared in PHP code using the `prepare_multipage()` method of the template object:

```
$tpl->prepare_multipage($pagelimit,$totalitems,$baseurl)
```

The `prepare_multipage()` function accepts the following arguments:

(int) `$pagelimit`	specifies the maximum number of items to be displayed per page
(int) `$totalitems`	specifies the total number of items that are available for display
(string) `$baseurl`	specifies the URL of the current page; the navigation links generated by DirecTemplate will point to this URL. DirecTemplate will automatically append a "pstart" query variable to the URL indicating the index of the first item to display on the page.

Inside the template, the `{$_multipage.nav}` variable is used to display the navigation links.

Example:

If variable $myitems contains a two-dimensional array of values from a database server, the following PHP code might be used to prepare the multipage result set:

```
// get the index of the first item
$firstitem = (int) $_REQUEST['pstart'];
// get the total number of items
$totalitems = count($myitems);
```

```
// we want to show 25 items per page
$pagelimit = 25;
// point links back to this page
$baseurl = $_SERVER['REQUEST_URI'];
// get the result set for the page
$pageitems = array_slice($myitems,$pstart,$pagelimit);
// assign the items and prepare the multipage result set
$tpl->assign('items',$pageitems);
$tpl->prepare_multipage($pagelimit,$totalitems,$baseurl);
```

The corresponding template might include the following to display the array items and associated navigation links:

```
{loop $item=$myitems}
{$item}
{loop}
Navigation links: {$_multipage.nav}
```

6. Advanced Topics

6.1. Adding Custom Functionality

6.1.1. Custom Plugins

Custom plugins can be defined by adding entries to the `plugins` array, which is a member of the template object. For example:

```
$tpl->plugins = array("plugin_name"=>"function_to_call");
```

The `function_to_call` must be a PHP function defined as:

```
function function_to_call($arguments) { }
```

DirecTemplate will pass the `$arguments` argument as an array containing all of the arguments passed to the plugin, wherein each key represents the name of the argument, and each value represents the value of the argument.

The plugin can then be called from within the template as:

```
{plugin_name argumentname=value}
```

6.1.2. Custom Transformations

Custom transformations can be defined by adding entries to the `transforms` array, which is a member of the template object. For example:

```
$tpl->transforms = array("transform_name"=>"function_to_call");
```

The `function_to_call` must be a PHP function defined as:

```
function function_to_call($data,$arguments) { }
```

DirecTemplate will pass the `$data` argument as the data to be transformed, and the `$arguments` argument as an array containing all of the arguments passed to the transformation.

The transformation can then be called from within the template as:

```
{$data|transform_name:argument:...}
```

Alternately, transforms and plugins may be defined using the `TPL_CUSTOM_TRANSFORMS` and `TPL_CUSTOM_PLUGINS` string constants. For example:

```
define(
     "TPL_CUSTOM_TRANSFORMS",
     "transform_name:function_to_call|..."
);
```

6.1.3. Custom Filters

Custom filters can be defined by adding entries to the `filters` array, which is a member of the template object. For example:

```
$tpl->filters = array("filter_name"=>"function_to_call");
```

The `function_to_call` must be a PHP function defined as:

```
function function_to_call($data) { }
```

DirecTemplate will pass the `$data` argument as the final version of the page *after* parsing, but *before* being returned or displayed. The filter function should simply return the filtered version of the data.

Filters do not need to be invoked from within the template; DirecTemplate will automatically call every defined filter prior to returning.

6.2. Override and Fallback Template Directories

Sometimes it may be desirable to keep different versions of certain templates in separate directories (for example, to allow clients to keep their customized versions of default application templates in a separate directory to ease any future upgrades).

DirecTemplate provides two solutions for this scenario: Override and fallback directories.

If one or more override directories are specified, they will be searched for a matching template *before* the main template directory is searched. If a template with a matching filename is found in an override directory, it will be used unconditionally and the main template directory will be ignored.

For example, say DirecTemplate has been prepared as follows:

```
$tpl = &Template::singleton();
$tpl->template_dir = '/var/www/templates/';
$tpl->set_override('/var/www/custom_templates/');
$tpl->display('page.tpl');
```

In this scenario, DirecTemplate will first check for the existence of `/var/www/custom_templates/page.tpl`. If it exists, it will be used; otherwise, DirecTemplate will try `/var/www/templates/page.tpl`.

Relative override directories can also be specified as follows:

```
$tpl = &Template::singleton();
$tpl->template_dir = '/var/www/templates/';
$tpl->set_relative_override('custom/');
$tpl->display('page.tpl');
```

In this scenario, DirecTemplate will first check for the existence of `/var/www/templates/custom/page.tpl`. If it exists, it will be used; otherwise, DirecTemplate will try `/var/www/templates/page.tpl`. If `$tpl->template_dir` is lated changed to point to a different path, DirecTemplate will automatically search for a `custom/` directory within the new template path.

Fallback directories are similar to override directories, except that they are only consulted if the specified template does *not* exist in the main template directory.

For example, say DirecTemplate has been prepared as follows:

```
$tpl = &Template::singleton();
$tpl->template_dir = '/var/www/client_templates/';
$tpl->set_fallback('/var/www/default_templates/');
$tpl->display('page.tpl');
```

In this scenario, DirecTemplate will first check for the existence of `/var/www/client_templates/page.tpl`. If it exists, it will be used; otherwise, DirecTemplate will try `/var/www/default_templates/page.tpl`.

6.3. Assigning Variables from Within Templates

Occasionally it may be necessary to set variables from within a template, rather than from the script itself. The procedure for doing this is somewhat convoluted due to the non-linear, regular-expressions-based technique that DirecTemplate uses (for the sake of speed) to parse templates.

Use of these techniques is not recommended due to their complexity, but the instructions below are provided for use where necessary.

Note variables cannot be set conditionally within an `{if}` or `{check}` clause; `{set}`, `{postset}`, and `{loopset}` tokens are *always* processed no matter where they are placed in the code.

The following syntax is used to assign a variable from within a template:

```
{set $varname}Value for variable, {$vars} permitted{/set}
```

Note that variables set in this manner will be set *before* loops are executed. Thus, any variables created or modified during a loop will *not* be accessible through `{set}` tags. To perform a `{set}` operation AFTER all loops are executed, the following syntax may be used:

```
{postset $varname}Value for variable, {$vars}
permitted{/postset}
```

This latter notation should also be used if variables are intended to be set from *inside* a loop, such that any embedded variables will be parsed after the loop has been executed.

And finally, if you a variable needs to be set from *inside* a loop, which will also be *used* inside the loop, the following syntax may be used:

```
{loopset $varname}Value for variable, {$vars}
permitted{/loopset}
```

Note that variables set using this final notation will *only* be valid within the loop in which they were

defined; before and after the loop, their values should be treated as undefined.

7. Credits

DirecTemplate Template Library

The latest version of DirecTemplate may always be downloaded from:
http://code.blitzaffe.com/pages/phpclasses/files/directemplate_52-2