Smarty - die kompilierende PHP Template-Engine

von

Monte Ohrt <monte at ohrt dot com> und Andrei Zmievski <andrei@php.net>

Smarty - die kompilierende PHP Template-Engine

Veröffentlicht 14-12-2005

Inhaltsverzeichnis

Vorwort

Die Frage, wie man die Applikations-Logik eines PHP-Scriptes vom Layout trennt, ist unzweifelhaft eines der am häfigsten diskutierten Themen. Da PHP als "in HTML eingebettete Scripting-Sprache" angepriesen wird, ergibt sich nach einigen Projekten in denen man HTML und PHP gemischt hat schnell die Idee, Funktionalität und Darstellung zu trennen. Dazu kommt, dass in vielen Firmen Applikationsentwickler und Designer nicht die selbe Person sind. In Konsequenz beginnt die Suche nach einer Template-Lösung.

Als Beispiel: In unserer Firma funktioniert die Entwicklung einer Applikation wie folgt: Nachdem die Spezifikationen erstellt sind, entwickelt der Interface Designer einen Prototypen des Interfaces und übergibt dieses dem Programmierer. Der Programmierer implementiert die Geschäftslogik in PHP und verwendet den Interface-Prototypen zur Erstellung eines Template-Skeletts. Danach übergibt der Programmierer die Templates dem HTML/Webseiten-Designer welcher ihnen den letzten Schliff verleiht. Das Projekt kann mehrfach zwischen dem Programmieren und dem Designer ausgetauscht werden. Deshalb ist es wichtig, dass die Trennung von Logik und Design klar stattfindet. Der Programmierer will sich normalerweise nicht mit HTML herumschlagen müssen und möchte auch nicht, dass der Designer seinen PHP-Code verändert. Designer selbst benötigen Konfigurationsdateien, dynamische Blöcke und andere Interface spezifische Eigenheiten, möchten aber auch nicht direkt mit PHP in Berührung kommen.

Die meisten Template-Engines die heutzutage angeboten werden, bieten eine rudimentäre Möglichkeit Variablen in einem Template zu ersetzen und beherschen eine eingeschränkte Funktionalität für dynamische Blöcke. Unsere Anforderungen forderten jedoch ein wenig mehr. Wir wollten erreichen, dass sich Programmierer überhaupt nicht um HTML Layouts kümmern müssen. Dies war aber fast unumgänglich. Wenn ein Designer zum Beispiel alternierende Farben in einer Tabelle einsetzen wollte, musste dies vorhergehend mit dem Programmierer abgesprochen werden. Wir wollten weiter, dass dem Designer Konfigurationsdateien zur Verfügung stünden, aus denen er Variablen für seine Templates extrahieren kann. Die Liste ist endlos.

Wir begannen 1999 mit der Spezifikation der Template Engine. Nachdem dies erledigt war, fingen wir an eine Engine in C zu schreiben, die - so hofften wir - in PHP eingebaut würde. Nach einer hitzigen Debatte darüber was eine Template Engine können sollte und was nicht, und nachdem wir feststellen mussten, dass ein paar komplizierte technische Probleme auf uns zukommen würden, entschlossen wir uns die Template Engine in PHP als Klasse zu realisieren, damit sie von jederman verwendet und angepasst werden kann. So schrieben wir also eine Engine, die wir SmartTemplate™ nannten (anm: diese Klasse wurde nie veröffentlicht). SmartTemplate erlaubte uns praktisch alles zu tun was wir uns vorgenommen hatten: normale Variablen-Ersetzung, Möglichkeiten weitere Templates einzubinden, Integration von Konfigurationsdateien, Einbetten von PHP-Code, limitierte 'if'-Funktionalität und eine sehr robuste Implementation von dynamischen Blöcken die mehrfach verschachtelt werden konnten. All dies wurde mit Regulären Ausdrücken erledigt und der Sourcecode wurde ziemlich unübersichtlich. Für grössere Applikationen war die Klasse auch bemerkenswert langsam, da das Parsing bei jedem Aufruf einer Seite durchlaufen werden musste. Das grösste Problem aber war, dass der Programmierer das Setup, die Templates und dynamische Blöcke in seinem PHP-Skript definieren musste. Die nächste Frage war: wie können wir dies weiter vereinfachen?

Dann kam uns die Idee, aus der schließlich Smarty wurde. Wir wussten wie schnell PHP-Code ohne den Overhead des Template-Parsing ist. Wir wussten ebenfalls wie pedantisch PHP aus Sicht eines durchschnittlichen Designers ist und dass dies mit einer einfacheren Template-Syntax verborgen werden kann. Was wäre also, wenn wir diese beiden Stärken vereinten? Smarty war geboren...

Teil I. Erste Schritte

Inhaltsverzeichnis

Kapitel 1. Was ist Smarty?

Smarty ist eine Template-Engine für PHP. Genauer gesagt erlaubt es die einfache Trennung von Applikations-Logik und Design/Ausgabe. Dies ist vor allem wünschenswert, wenn der Applikationsentwickler nicht die selbe Person ist wie der Designer. Nehmen wir zum Beispiel eine Webseite die Zeitungsartikel ausgibt. Der Titel, die Einführung, der Author und der Inhalt selbst enthalten keine Informationen darüber wie sie dargestellt werden sollen. Also werden sie von der Applikation an Smarty übergeben, damit der Designer in den Templates mit einer Kombination von HTML- und Template-Tags die Ausgabe (Tabellen, Hintergrundfarben, Schriftgrössen, Stylesheets, etc.) gestalten kann. Falls nun die Applikation eines Tages angepasst werden muss, ist dies für den Designer nicht von Belang, da die Inhalte immer noch genau gleich übergeben werden. Genauso kann der Designer die Ausgabe der Daten beliebig verändern, ohne dass eine Änderung der Applikation vorgenommen werden muss. Somit können der Programmierer die Applikations-Logik und der Designer die Ausgabe frei anpassen, ohne sich dabei in die Quere zu kommen.

Was Smarty nicht kann: Smarty versucht nicht die gesamte Logik aus dem Template zu verbannen. Solange die verwendete Logik ausschließlich für die Ausgabe verwendet wird, kann sie auch im Template eingebettet werden. Ein Tip: versuchen Sie Applikations-Logik aus dem Template und Präsentations-Logik aus der Applikation herauszuhalten. Nur so bleibt die Applikation auf absehbere Zeit gut skalier- und wartbar.

Einer der einzigartigen Aspekte von Smarty ist die Kompilierung der Templates. Smarty liest die Template-Dateien und generiert daraus neue PHP-Skripte; von da an werden nur noch diese Skripte verwendet. Deshalb müssen Templates nicht für jeden Seitenaufruf performance-intensiv neu geparst werden und jedes Template kann voll von PHP Compiler-Cache Lösungen profitieren. (Zend, http://www.zend.com/; PHP Accelerator, http://www.php-accelerator.co.uk)

Ein paar Smarty Charakteristiken

- Sehr schnell.
- Sehr effizient, da der PHP-Parser die 'schmutzige' Arbeit übernimmt.
- Kein Overhead durch Template-Parsing, nur einmaliges kompilieren.
- Re-kompiliert nur gänderte Templates.
- Sie können die Engine um individuelle Funktionen und Variablen-Modifikatoren erweitern.
- Konfigurierbare Syntax für Template-Tags: Sie können {}, {{}}, <!--{}-->, etc. verwenden.
- 'if/elseif/else/endif'-Konstrukte werden direkt dem PHP-Parser übergeben. Somit können {if ...} Ausdrücke sowohl sehr einfach als auch sehr komplex sein.
- Unbegrenzte Verschachtelung von 'section', 'if' und anderen Blöcken.
- Ermöglicht die direkte Einbettung von PHP-Code. (Obwohl es weder benötigt noch empfohlen wird, da die Engine einfach erweiterbar ist.)
- Eingebauter Caching-Support
- Beliebige Template-Quellen
- Eigene Cache-Handling Funktionen
- Plugin Architektur

Kapitel 2. Installation

Inhaltsverzeichnis

Anforderungen

Smarty benötigt einen Webserver mit PHP >=4.0.6.

Basis Installation

> **Technische Bemerkung:** Dieser Leitfaden geht davon aus, dass Sie Ihr Webserver- und PHP-Setup kennen und mit den Namenskonventionen für Dateien und Verzeichnisse Ihres Betriebssystems vertraut sind. Im Folgenden wird ein Unix-Dateisystem verwendet, stellen Sie also sicher, dass sie die für Ihr Betriebssystem nötigen Änderungen vornehmen.
>
> Das Beispiel geht davon aus, dass '/php/includes' in Ihrem PHP-'include_path' liegt. Konsultieren Sie das PHP-Manual für weiterführende Informationen hierzu.

Installieren Sie als erstes die Smarty-Library Dateien (den /libs/-Ordner der Smarty Distribution). Diese Dateien sollten von Ihnen NICHT editiert und von allen Applikationen verwendet werden. Sie werden nur erneuert, wenn Sie eine neue Version von Smarty installieren.

> **Technische Bemerkung:** Wir empfehlen keine Änderungen an den Smarty-Library Dateien vorzunehmen. Dies macht ein mögliches Upgrade wesentlich einfacher. Sie müssen diese Dateien auch nicht anpassen, um Smarty zu konfigurieren! Benutzen Sie für diesen Zwecke eine Instanz der Smarty-Klasse.

Folgende Library Dateien werden mit Smarty geliefert und werden benötigt:

Beispiel 2.1. Benötigte Smarty-Library Dateien

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/internals/*.php (alle)
/plugins/*.php (alle)
```

Sie können diese Dateien entweder in Ihrem PHP-'include_path' oder auch in irgend einem anderen Verzeichnis ablegen, solange die Konstante [http://php.net/define] SMARTY_DIR auf den korrekten Pfad zeigt. Im Folgenden werden Beispiele für beide Fälle aufgezeigt. SMARTY_DIR **muss in jedem Fall** am Ende einen Slash ("/", unter Windows ggf. einen Backslash "\") enthalten.

So erzeugt man eine Instanz der Smarty-Klasse im PHP-Skript:

Beispiel 2.2. Smarty Instanz erstellen:

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
?>
```

Versuchen Sie das Skript auszuführen. Wenn Sie eine Fehlermeldung erhalten dass `Smarty.class.php` nicht gefunden werden konnte, versuchen Sie folgendes:

Beispiel 2.3. Manuelles setzen der SMARTY_DIR-Konstanten

```
<?php
// *nix-Stil
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

// Windows-Stil
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');

// Ein kleiner Hack der unter *nix und Windows funktioniert wenn Smarty
// in einem Verzeichnis 'includes/' unterhalb des Beispielskriptes liegt
define('SMARTY_DIR',str_replace("\\","/",getcwd()).'/includes/Smarty-v.e.r/libs/');

require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

Beispiel 2.4. Absoluter Pfad übergeben

```
<?php
require('/usr/local/lib/php/Smarty/Smarty.class.php');
$smarty = new Smarty;
?>
```

Beispiel 2.5. Library Verzeichnis dem Include-Pfad hinzufügen

```
<?php
// editieren Sie php.ini, f&uuml;gen Sie das Smarty
// Verzeichnis Ihrem include_path hinzu, und starten Sie den Webserver neu
// Danach sollte folgendes Funktionieren.
require('Smarty.class.php');
$smarty = new Smarty;

?>
```

Beispiel 2.6. SMARTY_DIR manuell setzen

```
<?php
define('SMARTY_DIR','/usr/local/lib/php/Smarty/');
require(SMARTY_DIR.'Smarty.class.php');
```

```
$smarty = new Smarty;
?>
```

Jetzt, wo die Library Dateien an ihrem Platz sind, wird es Zeit, die Smarty Verzeichnisse zu erstellen.

Für unser Beispiel werden wir die Smarty Umgebung für eine Gästebuch-Applikation konfigurieren. Wir verwenden den Applikationsnamen nur, um die Verzeichnis-Struktur zu verdeutlichen. Sie können die selbe Umgebung für alle Ihre Applikationen verwenden indem Sie 'guestbook' durch dem Namen Ihrer Applikation ersetzen.

Stellen Sie sicher, dass Sie die DocumentRoot Ihres Webservers kennen. In unserem Beispiel lautet sie '/web/www.domain.com/docs/'.

Die Smarty Verzeichnisse werden in den Klassen-Variablen $template_dir, $compile_dir, $config_dir und $cache_dir definiert. Die Standardwerte sind: `templates`, `templates_c`, `configs` und `cache`. Für unser Beispiel legen wir alle diese Verzeichnisse unter `/web/www.domain.com/smarty/guestbook/` an.

> **Technische Bemerkung:** Wir empfehlen, diese Verzeichnisse ausserhalb der DocumentRoot anzulegen, um mögliche Direktzugriffe zu verhindern.

In Ihrer DocumentRoot muss mindestens eine Datei liegen, die für Browser zugänglich ist. Wir nennen dieses Skript `index.php`, und legen es in das Verzeichnis `/guestbook/` in unserer DocumentRoot.

> **Technische Bemerkung:** Bequem ist es, den Webserver so zu konfigurieren, dass `index.php` als Standard-Verzeichnis-Index verwendet wird. Somit kann man das Skript direkt mit 'http://www.domain.com/guestbook/' aufrufen. Falls Sie Apache verwenden, lässt sich dies konfigurieren indem Sie `index.php` als letzten Eintrag für *DirectoryIndex* verwenden. (Jeder Eintrag muss mit einem Leerzeichen abgetrennt werden).

Die Dateistruktur bis jetzt:

Beispiel 2.7. Beispiel der Dateistruktur

```
/usr/local/lib/php/Smarty/Smarty.class.php
/usr/local/lib/php/Smarty/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty/Config_File.class.php
/usr/local/lib/php/Smarty/debug.tpl
/usr/local/lib/php/Smarty/plugins/*.php
/usr/local/lib/php/Smarty/core/*.php

/web/www.example.com/smarty/guestbook/templates/
/web/www.example.com/smarty/guestbook/templates_c/
/web/www.example.com/smarty/guestbook/configs/
/web/www.example.com/smarty/guestbook/cache/

/web/www.example.com/docs/guestbook/index.php
```

> **Technische Bemerkung:** Falls Sie kein Caching und keine Konfigurationsdateien verwenden, ist es nicht erforderlich die Verzeichnisse '$config_dir' und '$cache_dir' zu erstellen. Es wird jedoch trotzdem empfohlen, da diese Funktionalitäten eventuell später genutzt werden sollen.

Smarty benötigt **Schreibzugriff** auf die Verzeichnisse $compile_dir und $cache_dir. Stellen Sie also sicher, dass der Webserver-Benutzer (normalerweise Benutzer 'nobody' und Gruppe 'nogroup') in diese Verzeichnisse schreiben kann. (In OS X lautet der Benutzer normalerweise 'www' und ist in der Gruppe 'www'). Wenn Sie Apache verwenden, können Sie in der httpd.conf (gewöhnlich in '/usr/local/apache/conf/') nachsehen, unter welchem Benutzer Ihr Server läuft.

Beispiel 2.8. Dateirechte einrichten

```
chown nobody:nobody /web/www.example.com/smarty/guestbook/templates_c/
chmod 770 /web/www.example.com/smarty/guestbook/templates_c/

chown nobody:nobody /web/www.example.com/smarty/guestbook/cache/
chmod 770 /web/www.example.com/smarty/guestbook/cache/
```

> **Technische Bemerkung:** 'chmod 770' setzt ziemlich strenge Rechte und erlaubt nur dem Benutzer 'nobody' und der Gruppe 'nobody' Lese-/Schreibzugriff auf diese Verzeichnisse. Falls Sie die Rechte so setzen möchten, dass auch andere Benutzer die Dateien lesen können (vor allem für Ihren eigenen Komfort), so erreichen Sie dies mit 775.

Nun müssen wir die `index.tpl` Datei erstellen, welche Smarty laden soll. Die Datei wird in Ihrem $template_dir abgelegt.

Beispiel 2.9. Editieren von /web/www.example.com/smarty/guestbook/templates/index.tpl

```
{* Smarty *}

Hallo, {$name}!
```

> **Technische Bemerkung:** {* Smarty *} ist ein Template-Kommentar. Der wird zwar nicht benötigt, es ist jedoch eine gute Idee jedes Template mit einem Kommentar zu versehen. Dies erleichtert die Erkennbarkeit des Templates, unabhängig von der verwendeten Dateierweiterung. (Zum Beispiel für Editoren die Syntax-Highlighting unterstützen.)

Als nächstes editieren wir die Datei `index.php`. Wir erzeugen eine Smarty-Instanz, weisen dem Template mit assign() eine Variable zu und geben `index.tpl` mit display aus.

Beispiel 2.10. Editieren von /web/www.example.com/docs/guestbook/index.php

```
<?php

define('SMARTY_DIR','/usr/local/lib/php/Smarty/');
require(SMARTY_DIR.'Smarty.class.php');

$smarty = new Smarty();

$smarty->template_dir = '/web/www.example.com/smarty/guestbook/templates/';
$smarty->compile_dir = '/web/www.example.com/smarty/guestbook/templates_c/';
$smarty->config_dir = '/web/www.example.com/smarty/guestbook/configs/';
$smarty->cache_dir = '/web/www.example.com/smarty/guestbook/cache/';

$smarty->assign('name','Ned');

$smarty->display('index.tpl');
?>
```

> **Technische Bemerkung:** In unserem Beispiel verwenden wir durchwegs absolute Pfadnamen zu den Smarty-Verzeichnissen. Falls `/web/www.example.com/smarty/guestbook/` in Ihrem PHP-'include_path' liegt, wäre dies nicht nötig. Es ist jedoch effizienter und weniger fehleranfällig die Pfade absolut zu setzen. Und es garantiert, dass Smarty die Templates aus dem geplanten Verzeichnis lädt.

Wenn Sie `index.php` nun in Ihrem Webbrowser öffnen, sollte 'Hallo, Ned!' ausgegeben werden.

Die Basis-Installation von Smarty wäre somit beendet.

Erweiterte Konfiguration

Dies ist eine Weiterführung der Basis Installation, bitte lesen Sie diese zuerst!

Ein flexiblerer Weg um Smarty aufzusetzen ist, die Klasse zu erweitern und eine eigene Smarty-Umgebung zu initialisieren. Anstatt immer wieder die Verzeichnisse zu definieren, kann diese Aufgabe auch in einer einzigen Datei erledigt werden. Beginnen wir, indem wir ein neues Verzeichnis namens '/php/includes/guestbook/' erstellen und eine Datei namens 'setup.php' darin anlegen.

Beispiel 2.11. Editieren von /php/includes/guestbook/setup.php

```
<?php

// Smarty Library Dateien laden
define('SMARTY_DIR','/usr/local/lib/php/Smarty/');
require(SMARTY_DIR.'Smarty.class.php');

// ein guter Platz um Applikations spezifische Libraries zu laden
// require('guestbook/guestbook.lib.php');

class Smarty_GuestBook extends Smarty {

   function Smarty_GuestBook()
   {
  // Konstruktor. Diese Werte werden f&uuml;r jede Instanz automatisch gesetzt

  $this->Smarty();

  $this->template_dir = '/web/www.example.com/smarty/guestbook/templates/';
  $this->compile_dir = '/web/www.example.com/smarty/guestbook/templates_c/';
  $this->config_dir = '/web/www.example.com/smarty/guestbook/configs/';
  $this->cache_dir = '/web/www.example.com/smarty/guestbook/cache/';

  $this->caching = true;
  $this->assign('app_name','Guest Book');
   }

}
?>
```

> **Technische Bemerkung:** In unserem Beispiel werden die Library Dateien ausserhalb der DocumentRoot abgelegt. Diese Dateien könnten sensitive Informationen enthalten, die wir nicht zugänglich machen möchten. Deshalb legen wir alle Library Dateien in '/php/includes/guestbook/' ab und laden sie in unserem 'setup.php' Skript, wie Sie im oben gezeigten Beispiel sehen können.

Nun passen wir `index.php` an, um 'setup.php' zu verwenden:

Beispiel 2.12. Editieren von /web/www.example.com/docs/guestbook/index.php

```
<?php

require('guestbook/setup.php');

$smarty = new Smarty_GuestBook;
$smarty->assign('name','Ned');
$smarty->display('index.tpl');

?>
```

Wie Sie sehen können, ist es sehr einfach eine Instanz von Smarty zu erstellen. Mit Hilfe von Smarty_GuestBook werden alle Variablen automatisch initialisiert.

Teil II. Smarty für Template Designer

Inhaltsverzeichnis

Kapitel 3. Grundlegende Syntax

Inhaltsverzeichnis

Alle Smarty Template-Tags werden mit Trennzeichen umschlossen. Normalerweise sind dies: { und }, sie können aber auch verändert werden.

Für die folgenden Beispiele wird davon ausgegangen, dass Sie die Standard-Trennzeichen verwenden. Smarty erachtet alle Inhalte ausserhalb der Trennzeichen als statisch und unveränderbar. Sobald Smarty auf Template-Tags stösst, versucht es diese zu interpretieren und die entsprechenden Ausgaben an deren Stelle einzufügen.

Kommentare

Kommentare werden von Asterisks umschlossen, und mit Trennzeichen umgeben. Beispiel: {* das ist ein Kommentar *} Smarty-Kommentare werden in der Ausgabe nicht dargestellt und vor allem dazu verwendet, die Templates verständlicher aufzubauen. Smarty Kommentare werden sind in der engültigen Ausgabe NICHT dargestellt. (im Gegensatz zu <!-- HTML Kommentaren -->). Sie sind nützlich um in den Templates interne Anmerkungen zu hinterlassen.

Beispiel 3.1. Kommentare

```
<body>
{* Dies ist ein einzeiliger Kommentar *}

{* dies ist ein mehrzeiliger
   Kommentar, der nicht zum
   Browser gesandt wird.
*}
</body>

{* einbinden des Header-Templates *}
{include file="header.tpl"}

{* Entwicklernotiz: $includeFile wurde in 'foo.php' zugewiesen *}
{include file=$includeFile}

{include file=#includeFile#}

{* Ausgabe der drop-down Liste *}
{* Dieser <select> Block ist überflüssig *}
{*
<select name=firma>
{html_options options=$vals selected=$selected}
</select>
*}
```

Variablen

Templatevariablennamen beginnen mit einem $dollar-Zeichen. Sie können Ziffer, Buchstaben und Unterstriche ('_') enthalten, sehr ähnlich den Variablen in PHP [http:/ / php.net/ language.variables]. Numerische Arrayindizes können referenziert werden und auch Nichtnumerische. Zugriff auf Objekteigenschaften und -methoden ist auch möglich. Konfigurationsvariablen sind einer Ausname was die Dollarzeichen-Syntax angeht. Diese werden durch umgebende #Doppelkreuze# oder über die Varible $smarty.config referenziert.

Beispiel 3.2. Variablen

```
{$foo}        <-- Zeigt einfache Variable an (kein Array oder Objekt)
{$foo[4]}     <-- Zeigt 5. Element von einem Array an, deren Schlussel bei 0 beginnen
{$foo.bar}    <-- Zeigt das Element zum Schlüssel "bar" des Arrays an (wie PHPs $foo['bar'])
{$foo.$bar}   <-- Zeigt das Element eines variablen Schlüssels an (wie PHPs $foo[$bar])
{$foo->bar}   <-- Zeigt eine Eigenschaft "bar" des Objekts $foo an
{$foo->bar()} <-- Zeigt den Rückgabewert der Objectmethode "bar" an
{#foo#}       <-- Zeift die Konfigurationsvariable "foo" an
{$smarty.config.foo} <-- Synonym für {#foo#}
{$foo[bar]}   <-- Syntax zum zugriff auf Element in einer Section-Schleife, siehe {section}
{assign var=foo value="baa"}{$foo} <--  Gibt "baa" aus, siehe {assign}

Viele weitere Kombinationen sind erlaubt

{$foo.bar.baz}
{$foo.$bar.$baz}
{$foo[4].baz}
{$foo[4].$baz}
{$foo.bar.baz[4]}
{$foo->bar($baz,2,$bar)} <-- Parameter übergeben
{"foo"}       <-- Statische (konstante) Werte sind auch erlaubt
```

Siehe auch: Die reservierte {$smarty} Variable und Verwendung von Variablen aus Konfigurationsdateien.

Funktionen

Jedes Smarty-Tag gibt entweder eine Variable aus oder ruft eine Funktion auf. Funktionen werden aufgerufen indem der Funktionsname und die Parameter mit Trennzeichen umschlossen werden. Beispiel: {funcname attr1="val" attr2="val"}.

Beispiel 3.3. Funktions-Syntax

```
{config_load file="colors.conf"}

{include file="header.tpl"}

{if $highlight_name}
       Welcome, <font color="{#fontColor#}">{$name}!</font>
{else}
       Welcome, {$name}!
{/if}

{include file="footer.tpl"}
```

Sowohl der Aufruf von eingebauten, als auch der von eigenen Funktionen folgt der gleichen Syntax.

Eingebaute Funktionen erlauben einige **Basis**-Operationen wie if, section und strip. Diese Funktionen können nicht verändert werden.

Individuelle Funktionen die die Fähigkeiten von Smarty erweitern werden als Plugins implementiert. Diese Funktionen können von Ihnen angepasst werden, oder Sie können selbst neue Plugins hinzufügen. {html_options} und {html_select_date} sind Beispiele solcher Funktionen.

Attribute / Parameter

Die meisten Funktionen nehmen Parameter entgegen, die das Verhalten der Funktion definieren beziehungsweise beeinflussen. Parameter für Smarty Funktionen sind HTML Attributen sehr ähnlich. Statische Werte müssen nicht in Anführungszeichen gesetzt werden, für literale Zeichenketten (literal strings) wird dies jedoch empfohlen.

Bestimmte Parameter verlangen logische Werte (true / false). Diese können auch ohne Anführungszeichen angegeben werden: `true`, `on` und `yes` - oder `false`, `off` und `no`.

Beispiel 3.4. Funktions-Parameter Syntax

```
{include file='header.tpl'}

{include file='header.tpl' attrib_name='attrib value'}

{include file=$includeFile}

{include file=#includeFile#}

{html_select_date display_days=yes}

{mailto address='smarty@example.com'}

<select name=firma>
{html_options values=$vals selected=$selected output=$output}
</select>
```

Variablen mit Doppelten Anführungszeichen

Smarty erkennt zugewiesene Variablen mit doppelten Anführungszeichen solange die Variablen nur Zahlen, Buchstaben, Understriche oder Klammern [] enthalten. Mit allen anderen Zeichen wie Punkt, Objekt Referenzen, etc muss die Vairable mit Backticks (``) umschlossen sein.

Beispiel 3.5. Syntax von eingebetteten Anfürungszeichen

```
SYNTAX BEISPIELE:
{func var="test $foo test"}       &lt;-- sieht $foo
{func var="test $foo_bar test"}   &lt;-- sieht $foo_bar
{func var="test $foo[0] test"}    &lt;-- sieht $foo[0]
{func var="test $foo[bar] test"}  &lt;-- sieht $foo[bar]
{func var="test $foo.bar test"}   &lt;-- sieht $foo (nicht $foo.bar)
{func var="test `$foo.bar` test"} &lt;-- sieht $foo.bar
{func var="test `$foo.bar` test"|escape} <-- Modifikatoren ausserhalb der Anführungsz.!

PRAKTISCHE BEISPIELE:
{include file="subdir/$tpl_name.tpl"} &lt;-- ersetzt $tpl_name durch wert
{cycle values="one,two,`$smarty.config.myval`"} &lt;-- muss Backticks enthalten</programlisting>
```

Siehe auch escape (Maskieren).

Math

Mathematische Operationen können direkt auf Variablen verwendet werden.

Beispiel 3.6. Mathematik Beispiele

```
{$foo+1}

{$foo*$bar}

{* kompliziertere Beispiele *}

{$foo-&gt;bar-$bar[1]*$baz-&gt;foo-&gt;bar()-3*7}

{if ($foo+$bar.test%$baz*134232+10+$b+10)}

{$foo|truncate:"`$fooTruncCount/$barTruncFactor-1`"}

{assign var="foo" value="`$foo+$bar`"}
```

Siehe auch die {math}-Funktion für komplexere Berechnungen.

Smarty Parsing umgehen

Manchmal ist es wünschenswert, dass Smarty Teile eines Templates nicht parst. Dies ist zum Beispiel der Fall, wenn Javascript oder CSS im Template eingebettet werden. Da diese Sprachen selbst { und } nutzen, erkennt Smarty diese als Start- beziehungsweise End-Tags.

Der einfachste Weg, dieses Problem zu umgehen, ist das Auslagern des betreffenden Javascript oder CSS Codes in eigene Dateien.

Um solche Inhalte trotzdem im gleichen Template einzubetten, können Sie {literal} .. {/literal} Blöcke verwenden. Die aktuell benutzten Trennzeichen können Sie mit {ldelim}, {rdelim}, {$smarty.ldelim} und {$smarty.rdelim} ausgeben.

Manchmal ist es auch einfacher, die Trennzeichen selbst zu ändern: $left_delimiter und $right_delimiter definieren diese.

Beispiel 3.7. Beispiel wie die Trennzeichen angepasst werden

```
<?php

$smarty = new Smarty;
$smarty->left_delimiter = '<!--{';
$smarty->right_delimiter = '}-->';
$smarty->assign('foo', 'bar');
$smarty->display('example.tpl');

?>
```

example.tpl würde somit wie folgt aussehen:

```
<script language="javascript">
  var foo = <!--{$foo}-->;
  function dosomething() {
    alert("foo is " + foo);
  }
  dosomething();
</script>
```

Siehe auch: Escape Modifikator

Kapitel 4. Variablen

Inhaltsverzeichnis

Smarty hat verschiedene Variablentypen, welche weiter unten detailliert beschrieben werden. Der Typ der Variable wird durch das Vorzeichen bestimmt.

Variablen können in Smarty direkt ausgegeben werden oder als Argumente für Funktionsparameter und Modifikatoren sowie in Bedingungen verwendet werden. Um eine Variable auszugeben, umschliessen Sie sie mit Trennzeichen, so dass die Variable das einzige enthaltene Element ist. Beispiele:

```
{$Name}

{$Kontakte[zeile].Telefon}

&lt;body bgcolor="{#bgcolor#}"&gt;
```

Aus einem PHP-Skript zugewiesene Variablen

Variablen die in einem PHP Skript assigned mit zugewiesen wurden, müssen mit eine Dollar Zeichen $ versehen werden. Auf die gleiche Art werden Variablen ausgegeben, die im Template mit {assign} zugewiesen wurden.

Beispiel 4.1. zugewiesene Variablen

PHP-Skript

```
<?php

$smarty = new Smarty;

$smarty->assign('vorname', 'Andreas');
$smarty->assign('nachname', 'Halter');
$smarty->assign('treffpunkt', 'New York');

$smarty->display('index.tpl');

?>
```

Mit folgendem index.tpl:

```
Hallo {$vorname} {$nachname}, schön, dass Du es einrichten kannst.
<br />
{*
  das hier funktioniert nicht, da bei Variablennamen auf
  Gross-Kleinschreibung geachtet werden muss:
*}
Diese Woche findet das Treffen in {$treffPunkt} statt.

{* aber das hier funktioniert: *}
Diese Woche findet das Treffen in {$treffpunkt} statt.
```

Ausgabe:

```
Hallo Andreas Halter, schön, dass Du es einrichten kannst.
<br />
Diese Woche findet das Treffen in  statt.
Diese Woche findet das Treffen in New York statt.
```

Assoziative Arrays

Sie können auch auf die Werte eines in PHP zugewiesenen assoziativen Arrays zugreifen, indem Sie den Schlüssel (Indexwert) nach einem '.'-Zeichen (Punkt) notieren.

Beispiel 4.2. Zugriff auf Variablen eines assoziativen Arrays

```
<?php
$smarty->assign('kontakte',
    array('fax' => '555-222-9876',
          'email' => 'zaphod@slartibartfast.example.com',
          'telefon' => array('privat' => '555-444-3333',
                             'mobil'  => '555-111-1234')
                       )
         );
$smarty->display('index.tpl');
?>
```

Bei folgender index.tpl:

```
{$kontakte.fax}<br />
{$kontakte.email}<br />
{* auch multidimensionale Arrays können so angesprochen werden *}
{$kontakte.telefon.privat}<br />
{$kontakte.telefon.mobil}<br />
```

Ausgabe:

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
555-444-3333<br />
555-111-1234<br />
```

Array Index

Arrays können - ähnlich der PHP-Syntax - auch über ihren Index angesprochen werden.

Beispiel 4.3. Zugriff über den Array Index

```
<?php
$smarty->assign('kontakte', array(
                            '555-222-9876',
                            'zaphod@slartibartfast.example.com',
                             array('555-444-3333',
                                   '555-111-1234')
                             ));
$smarty->display('index.tpl');
?>
```

Bei folgendem index.tpl:

```
{$kontakte[0]}<br />
{$kontakte[1]}<br />
{* auch hier sind multidimensionale Arrays möglich *}
{$kontakte[0][0]}<br />
{$kontakte[0][1]}<br />
```

Ausgabe:

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
555-444-3333<br />
555-111-1234<br />
```

Objekte

Attribute von aus PHP zugewiesenen Objekten können über das '->'-Symbol erreicht werden.

Beispiel 4.4. Zugriff auf Objekt-Attribute

```
name: {$person->name}<br />
email: {$person->email}<br />
```

Ausgabe:

```
name: Zaphod Beeblebrox<br />
email: zaphod@slartibartfast.example.com<br />
```

Verwendung von Variablen aus Konfigurationsdateien

Variablen, die aus einer Konfigurationsdatei geladen werden, referenziert man mit umschliessenden '#'-Zeichen (Raute).

Beispiel 4.5. Konfigurationsvariablen

```
<html>
<title>{#seitenTitel#}</title>
<body bgcolor="{#bodyHintergrundFarbe#}">
<table border="{#tabelleRahmenBreite#}" bgcolor="{#tabelleHintergrundFarbe#}">
<tr bgcolor="{#reiheHintergrundFarbe#}">
        <td>Vornamen</td>
        <td>Nachnamen</td>
        <td>Adresse</td>
</tr>
</table>
</body>
</html>
```

Variablen aus Konfigurationsdateien können erst verwendet werden, wenn sie aus der Datei geladen wurden. Dieser Vorgang wird im Abschnitt **config_load** weiter unten näher erläutert.

Die reservierte {$smarty} Variable

Die reservierte Variable {$smarty} wird verwendet, um auf spezielle Template-Variablen zuzugreifen. Im Folgenden die Liste der Variablen:

Request-Variablen

Aud die Request-Variablen [http:// php.net/ reserved.variables] $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV and $_SESSION (siehe $request_vars_order und $request_use_auto_globals) kann wie folgt zugegriffen werden.

Beispiel 4.6. Ausgabe der Requestvariablen (Anfragevariablen)

```
{* anzeigen der variable 'page' aus der URL oder dem FORM, welche mit GET übertragen wurde *}
{$smarty.get.page}

{* anzeigen der variable 'page' welche mit POST übertragen wurde *}
{$smarty.post.page}

{* anzeigen des cookies "benutzer" *}
{$smarty.cookies.benutzer}

{* anzeigen der Server-Variable "SERVER_NAME" *}
{$smarty.server.SERVER_NAME}

{* anzeigen der Environment-Variable "PATH" *}
{$smarty.env.PATH}

{* anzeigen der Session-Variable "id" *}
{$smarty.session.id}

{* anzeigen der Variable "benutzer" aus dem $_REQUEST Array (Zusammenstellung von get/post/cookie/serve
{$smarty.request.benutzer}
```

> **Anmerkung:** Aus historischen Gründen kann {$SCRIPT_NAME} verwendet werden, allerdings ist {$smarty.server.SCRIPT_NAME} die empfohlene Variante.

{$smarty.now}

Die momentane Unix-Timestamp kann über {$smarty.now} angefragt werden. Diese Zahl ist die Summe der verstrichenen Sekunden seit Beginn der UNIX-Epoche (1. Januar 1970) und kann zur Anzeige direkt dem 'date_format'-Modifikator übergeben werden.

Beispiel 4.7. Verwendung von {$smarty.now}

```
{* Verwendung des 'date_format'-Modifikators zur Anzeige der Zeit *}
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

{$smarty.const}

Hiermit kann auf PHP-Konstanten zugegriffen werden. Siehe auch smarty constants

Beispiel 4.8. Benutzung von {$smarty.const}

```
{$smarty.const._MY_CONST_VAL}
```

{$smarty.capture}

Auf die mit dem {capture}..{/capture} Konstrukt abgefangene Ausgabe kann via {$smarty} zugegriffen werden. Ein Beispiel dazu finden Sie im Abschnitt zu capture.

{$smarty.config}

{$smarty} kann dazu genutzt werde, um auf Config-Variablen zuzugreifen. {$smarty.config.foo} ist ein Synonym for {#foo#}. Im Abschnitt {config_load} ist ein Beispiel.

{$smarty.section}, {$smarty.foreach}

{$smarty} wird auch verwendet, um auf Eigenschaften von {section} und foreach Schleifen zuzugreifen.

{$smarty.template}

Diese Variable enthält den Namen des gerade verarbeiteten Templates.

{$smarty.version}

Diese Variable enthält die Smarty Versionsnummer mit der das Template kompiliert wurde.

{$smarty.ldelim}, {$smarty.rdelim}

Diese Variablen dienen dazu den linken und rechten Trennzeichen wortwörtlich auszugeben. Siehe auch {ldelim},{rdelim}.

Siehe auch: Variables and Config Variables

Kapitel 5. Variablen-Modifikatoren

Inhaltsverzeichnis

Variablen-Modifikatoren können auf alle Variablen angewendet werden, um deren Inhalt zu verändern. Dazu hängen sie einfach ein | (Pipe-Zeichen) und den Modifikatornamen an die entsprechende Variable an. Ein Modifikator über Parameter in seiner Arbeitsweise beinflusst werden. Diese Parameter werden dem Modifikatorname angehängt und mit : getrennt.

Beispiel 5.1. Modifikator Beispiel

```
{* Modifikator auf eine Variable anwenden *}
{$titel|upper}
{* Modifikator mit Parametern *}
{$title|truncate:40:"..."}

{* Modifikator auf Funktionsparameter anwenden *}
{html_table loop=$myvar|upper}
{* mit Parametern *}
{html_table loop=$myvar|truncate:40:"..."}

{* formatierung einer Zeichenkette *}
{"foobar"|upper}

{* mit date_format das aktuelle Datum formatieren *}
{"now"|date_format:"%Y/%m/%d"}

{* modifier auf eigene Funktion anwenden *}
{mailto|upper address="me@domain.dom"}
```

Wenn Sie einen Modifikator auf ein Array anwenden, wird dieser auf jeden Wert angewandt. Um zu erreichen, dass der Modifkator auf den Array selbst angewendet wird, muss dem Modifikator ein @ Zeichen vorangestellt werden. Beispiel:

`{$artikelTitel|@count}` (gibt die Anzahl Elemente des Arrays $artikelTitel aus.)

Modifikatoren können aus Ihrem $plugins_dir automatisch geladen (sehen Sie dazu auch Naming Conventions) oder explizit registriert werden (register_modifier).

Zudem können alle PHP-Funktionen implizit als Modifikatoren verwendet werden. (Das Beispiel mit dem `@count` Modifier verwendet die Funktion 'count()' von PHP und keinen Smarty Modifikator) PHP Funktionen zu verwenden eröffnet zwei Probleme: erstens: manchmal ist die Parameter Reiehnfolge nicht erwünscht. (`{"%2.f"|sprintf:$float}` funktioniert zwar, sieht aber als `{$float|string_format:"%2.f"}` das durch Smarty geliefert wird, besser aus. Zweitens: wenn $security auf TRUE gesetzt ist, müssen alle verwendeten PHP Funktionen im $security_settings['MODIFIER_FUNCS']-Array enthalten sein.

Siehe auch register_modifier(), register_function(), Smarty durch Plugins erweitern und Variablen-Modifikatoren.

capitalize (in Grossbuchstaben schreiben)

Wird verwendet um den Anfangsbuchstaben aller Wörter in der Variable gross (upper case) zu schreiben.

Parameter Position	Typ	Benötigt	Standardwert	Beschreibung
1	boolean	Nein	false	Bestimmt ob Wörter die Ziffern enthalten auch in Großschreibung gewandelt werden

Beispiel 5.2. capitalize (in Grossbuchstaben schreiben)

```
<?php

$smarty->assign('articleTitle', 'diebe haben in norwegen 20 tonnen streusalz entwendet.');

?>
```

Wobei das Template wie folgt aussieht:

```
{$artikelTitel}
{$artikelTitel|capitalize}
```

AUSGABE:

```
diebe haben in norwegen 20 tonnen streusalz entwendet.
Diebe Haben In Norwegen 20 Tonnen Streusalz Entwendet.</programlisting>
```

Siehe auch lower (in Kleinbuchstaben schreiben) upper (in Grossbuchstaben umwandeln)

cat

Dieser Wert wird der aktuellen Variable hinzugefÃ¼gt.

Parameter Position	Typ	Benötigt	Standard	Beschreibung
1	string	Nein	*leer/empty*	Wert der an die Variable angefügt werden soll.

Beispiel 5.3. cat

```
<?php

$smarty->assign('articleTitle', "Psychics predict world didn't end");

?>
```

Bei folgendem index.tpl:

```
{$articleTitle|cat:" yesterday."}
```

Ausgabe:

```
Psychics predict world didn't end yesterday.
```

count_characters (Buchstaben zählen)

Parameter Position	Typ	Benötigt	Standard	Beschreibung
1	boolean	Nein	false	Definiert ob Leerzeichen mitgezählt werden sollen.

Wird verwendet um die Anzahl Buchstaben in einer Variable auszugeben.

Beispiel 5.4. count_characters (Buchstaben zählen)

```
{$artikelTitel}
{$artikelTitel|count_characters}
{$artikelTitel|count_characters:true}

AUSGABE:

20% der US-Amerikaner finden ihr Land (die USA) nicht auf der Landkarte.
61
72
```

count_paragraphs (Absätze zählen)

Wird verwendet, um die Anzahl der Absätze in einer Variable zu ermitteln.

Beispiel 5.5. count_paragraphs (Paragrafen zählen)

```
{$artikelTitel}
{$artikelTitel|count_paragraphs}

AUSGABE:

Britische Spezialeinheiten sind aufgrund eines "Navigationsfehlers" nicht wie beabsichtigt in Gibraltar
```

```
Ein spanischer Lokführer hat aus Protest gegen die Arbeitsbedingungen nach gearbeiteten acht Stunden ei
2
```

count_sentences (Sätze zählen)

Wird verwendet, um die Anzahl der Sätze in einer Variable zu ermitteln.

Beispiel 5.6. count_sentences (Sätze zählen)

```
{$artikelTitel}
{$artikelTitel|count_sentences}

AUSGABE:

Zwei Deutsche haben die sogenannte "Painstation" vorgestellt. Bei Fehlern im Spiel wird der Spieler dur
3
```

count_words (Wörter zählen)

Wird verwendet, um die Anzahl Wörter in einer Variable zu ermiteln.

Beispiel 5.7. count_words (Wörter zählen)

```
{$artikelTitel}
{$artikelTitel|count_words}

AUSGABE:

Südafrika: Eine Polizistin fesselte - mangels mitgebrachter Handschellen - drei Flüchtige mit ihrer Str
12
```

date_format (Datums Formatierung)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	string	Nein	%b %e, %Y	Das Format des ausgegebenen Datums.
2	string	Nein	n/a	Der Standardwert (Datum) wenn die Eingabe leer ist.

Formatiert Datum und Uhrzeit in das definierte 'strftime()'-Format. Daten können als Unix-Timestamps, MySQL-Timestamps und jeder Zeichenkette die aus 'Monat Tag Jahr' (von strtotime parsebar) besteht übergeben werden. Designer können 'date_format' verwenden, um vollständige Kontrolle über das Format des Datums zu erhalten. Falls das übergebene Datum leer ist und der zweite Parameter übergeben wurde, wird dieser formatiert und ausgegeben.

Beispiel 5.8. date_format (Datums Formatierung)

```
{$smarty.now|date_format}
{$smarty.now|date_format:"%A, %B %e, %Y"}
{$smarty.now|date_format:"%H:%M:%S"}

AUSGABE:

Feb 6, 2001
Tuesday, February 6, 2001
14:33:00
```

Beispiel 5.9. 'date_format' Konvertierungs Spezifikation

```
%a - abgekürzter Name des Wochentages, abhängig von der gesetzten Umgebung

%A - ausgeschriebener Name des Wochentages, abhängig von der gesetzten Umgebung

%b - abgekürzter Name des Monats, abhängig von der gesetzten Umgebung

%B - ausgeschriebener Name des Monats, abhängig von der gesetzten Umgebung

%c - Wiedergabewerte für Datum und Zeit, abhängig von der gesetzten Umgebung

%C - Jahrhundert (Jahr geteilt durch 100, gekürzt auf Integer, Wertebereich 00 bis 99)

%d - Tag des Monats als Zahl (Bereich 00 bis 31)

%D - so wie %m/%d/%y

%e - Tag des Monats als Dezimal-Wert, einstelligen Werten wird ein Leerzeichen voran gestellt (Werteber

%g - wie %G, aber ohne Jahrhundert.

%G - Das vierstellige Jahr entsprechend der ISO Wochennummer (siehe %V). Das gleiche Format und der gle

%h - so wie %b

%H - Stunde als Zahl im 24-Stunden-Format (Bereich 00 bis 23)

%I - Stunde als Zahl im 12-Stunden-Format (Bereich 01 bis 12)

%j - Tag des Jahres als Zahl (Bereich 001 bis 366)

%m - Monat als Zahl (Bereich 01 bis 12)

%M - Minute als Dezimal-Wert

%n - neue Zeile

%p - entweder `am' oder `pm' (abhängig von der gesetzten Umgebung) oder die entsprechenden Zeichenkette

%r - Zeit im Format a.m. oder p.m.

%R - Zeit in der 24-Stunden-Formatierung

%S - Sekunden als Dezimal-Wert

%t - Tabulator

%T - aktuelle Zeit, genau wie %H:%M:%S

%u - Tag der Woche als Dezimal-Wert [1,7], dabei ist 1 der Montag.

%U - Nummer der Woche des aktuellen Jahres als Dezimal-Wert, beginnend mit dem ersten Sonntag als erste

%V - Kalenderwoche (nach ISO 8601:1988) des aktuellen Jahres. Als Dezimal-Zahl mit dem Wertebereich 01

%w - Wochentag als Dezimal-Wert, Sonntag ist 0
```

```
%W - Nummer der Woche des aktuellen Jahres, beginnend mit dem ersten Montag als erstem Tag der ersten W

%x - bevorzugte Datumswiedergabe (ohne Zeit), abhängig von der gesetzten Umgebung.

%X - bevorzugte Zeitwiedergabe (ohne Datum), abhängig von der gesetzten Umgebung.

%y - Jahr als 2-stellige-Zahl (Bereich 00 bis 99)

%Y - Jahr als 4-stellige-Zahl inklusive des Jahrhunderts

%Z - Zeitzone, Name oder eine Abkürzung

%% - ein %-Zeichen

BEMERKUNG FÜR PROGRAMMIERER: 'date_format' ist ein wrapper für PHP's 'strftime()'-Funktion.
Je nachdem auf welchem System ihr PHP kompiliert wurde, ist es durchaus möglich, dass nicht alle
angegebenen Formatierungszeichen unterstützt werden. Beispielsweise stehen %e, %T, %R und %D
(eventuell weitere) auf Windowssystemen nicht zur Verfügung.
```

default (Standardwert)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	string	Nein	*leer*	Dieser Wert wird ausgegeben wenn die Variable leer ist.

Wird verwendet um den Standardwert einer Variable festzulegen. Falls die Variable leer ist oder nicht gesetzt wurde, wird dieser Standardwert ausgegeben. Default (Standardwert) hat 1 Parameter.

Beispiel 5.10. default (Standardwert)

```
{* gib "kein Titel" (ohne Anführungszeichen) aus, falls '$artikelTitel' leer ist *}
{$artikelTitel|default:"kein Titel"}

AUSGABE:

kein Titel
```

escape (Maskieren)

Parameter Position	Typ	Erforderlich	Mögliche (erlaubte) Werte	Standardwerte	Beschreibung
1	string	Nein	html, htmlall, url, quotes, hex, hexentity, javascript	html	Definiert die zu verwendende Maskierung.

Wird verwendet um eine Variable mit HTML, URL oder einfachen Anführungszeichen, beziehungsweise Hex oder Hex-Entitäten zu maskieren. Hex und Hex-Entity kann verwendet werden um "mailto:" -Links so zu verändern, dass sie von Web-Spiders (E-Mail Sammlern) verborgen bleiben und dennoch les-/linkbar für Webbrowser bleiben. Als Standard, wird 'HTML'-Maskierung verwendet.

Beispiel 5.11. escape (Maskieren)

```
<?php
index.php:

$smarty-&gt;assign('TitreArticle', "'Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohle
?>
```

Wobei im Template folgendes steht:

```
{$artikelTitel}
{$artikelTitel|escape}
{$artikelTitel|escape:"html"}    {* maskiert &amp; &quot; &#039; &lt; &gt; *}
{$artikelTitel|escape:"htmlall"} {* maskiert ALLE html Entit&auml;ten *}
{$artikelTitel|escape:"url"}
{$artikelTitel|escape:"quotes"}
&lt;a href="mailto:{$EmailAdresse|escape:"hex"}"&gt;{$EmailAdresse|escape:"hexentity"}&lt;/a&gt;
```

Ausgabe:

```
'Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohlen.'
&amp;#039;Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohlen.&amp;#039;
&amp;#039;Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohlen.&amp;#039;
&amp;#039;Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohlen.&amp;#039;
%27Zwei+Unbekannte+haben+im+Lidl+in+Monheim+24+Pakete+Kaffee+gestohlen.%27
\'Zwei Unbekannte haben im Lidl in Monheim 24 Pakete Kaffee gestohlen.\'
&lt;a href=&quot;mailto:%62%6f%62%40%6d%65%2e%6e%65%74&quot;&gt;&amp;#x62;&amp;#x6f;&amp;#x62;&amp;#x40
```

Siehe auch Smarty Parsing umgehen und Verschleierung von E-mail Adressen.

indent (Einrücken)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	integer	Nein	4	Definiert die Länge der Zeichenkette die verwendet werden soll um den Text einzurücken.
2	string	Nein	(ein Leerschlag)	Definiert das Zeichen, welches verwendet werden soll um den Text einzurücken.

Wird verwendet, um eine Zeichenkette auf jeder Zeile einzurücken. Optionaler Parameter ist die Anzahl der Zeichen, um die der Text eingerückt werden soll. Standardlänge ist 4. Als zweiten optionalen Parameter können sie ein Zeichen übergeben, das für die Einrückung verwendet werden soll (für Tabulatoren: '\t').

Beispiel 5.12. indent (Einrücken)

```
{$articleTitle}

{$articleTitle|indent}
```

```
{$articleTitle|indent:10}

{$articleTitle|indent:1:"\t"}
```

Ausgabe:

```
Nach einer feuchtfr&ouml;hlichen Nacht fand ein Brite sein Auto
nicht mehr und meldete es als gestohlen. Ein Jahr sp&auml;ter
besuchte er den Ort wieder und erinnerte sich, dass er
das Auto nur an einem anderen Ort abgestellt hatte -
dort stand das Fahrzeug nach einem Jahr auch noch.

    Nach einer feuchtfr&ouml;hlichen Nacht fand ein Brite sein Auto
    nicht mehr und meldete es als gestohlen. Ein Jahr sp&auml;ter
    besuchte er den Ort wieder und erinnerte sich, dass er
    das Auto nur an einem anderen Ort abgestellt hatte -
    dort stand das Fahrzeug nach einem Jahr auch noch.

          Nach einer feuchtfr&ouml;hlichen Nacht fand ein Brite sein Auto
          nicht mehr und meldete es als gestohlen. Ein Jahr sp&auml;ter
          besuchte er den Ort wieder und erinnerte sich, dass er
          das Auto nur an einem anderen Ort abgestellt hatte -
          dort stand das Fahrzeug nach einem Jahr auch noch.

        Nach einer feuchtfr&ouml;hlichen Nacht fand ein Brite sein Auto
        nicht mehr und meldete es als gestohlen. Ein Jahr sp&auml;ter
        besuchte er den Ort wieder und erinnerte sich, dass er
        das Auto nur an einem anderen Ort abgestellt hatte -
        dort stand das Fahrzeug nach einem Jahr auch noch.
```

lower (in Kleinbuchstaben schreiben)

Wird verwendet um eine Zeichenkette in Kleinbuchstaben auszugeben.

Beispiel 5.13. lower (in Kleinbuchstaben schreiben)

```
{$artikelTitel}
{$artikelTitel|lower}

AUSGABE:

In Kalifornien wurde ein Hund in das Wählerverzeichnis eingetragen.
in kalifornien wurde ein hund in das wählerverzeichnis eingetragen.
```

nl2br

Konvertiert alle Zeilenschaltungen in
 Tags. Genau wie die PHP Funktion nl2br.

Beispiel 5.14. nl2br

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', "Sonne oder Regen erwartet,\nnachts dunkel.");
$smarty->display('index.tpl');

?>
```

Wobei index.tpl wie folgt aussieht:

```
{$articleTitle|nl2br}
```

Ausgabe:

```
Sonne oder Regen erwartet,<br />nachts dunkel.
```

regex_replace (Ersetzen mit regulären Ausdrücken)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	string	Ja	*n/a*	Definiert das zu ersetzende Suchmuster, als regulären Ausdruck.
2	string	Ja	*n/a*	Definiert die ersetzende Zeichenkette.

Suchen/Ersetzen mit regulären Ausdrücken. Folgt der Syntax von PHP's preg_replace().

Beispiel 5.15. regex_replace (Ersetzen mit regulären Ausdrücken)

```
{* Ersetzt jeden Zeilenumbruch-Tabulator-Neuezeile, durch ein Leerzeichen. *}

{$artikelTitel}
{$artikelTitel|regex_replace:"/[\r\t\n]/":" "}

AUSGABE:

Ein Bankangestellter in England zerkaut aus Stress
 bei der Arbeit wöchentlich 50 Kugelschreiber. Er ist deshalb in Behandlung.
Ein Bankangestellter in England zerkaut aus Stress bei der Arbeit wöchentlich 50 Kugelschreiber. Er ist
```

replace (Ersetzen)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	string	Ja	*n/a*	Die zu ersetzende Zeichenkette.
2	string	Ja	*n/a*	Die ersetzende Zeichenkette.

Einfaches suchen/ersetzen in einer Variable.

Beispiel 5.16. replace (Ersetzen)

```
{$artikelTitel}
{$artikelTitel|replace:"Fracht":"Lieferung"}
{$artikelTitel|replace:" ":"   "}
```

```
AUSGABE:

Ein Holsten-Laster hat in England seine komplette Fracht verloren, die nun von jedermann aufgesammelt w
Ein Holsten-Laster hat in England seine komplette Lieferung verloren, die nun von jedermann aufgesammel
Ein   Holsten-Laster   hat   in   England   seine   komplette   Fracht   verloren,   die   nun   von
```

spacify (Zeichenkette splitten)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	string	Nein	*ein Leerzeichen*	Definiert die zwischen allen Zeichen einzufügende Zeichenkette.

Fügt zwischen allen Zeichen einer Variablen ein Leerzeichen ein. Eine alternativ einzufügende Zeichenkette kann über den ersten Parameter definiert werden.

Beispiel 5.17. spacify (Zeichenkette splitten)

```
{$artikelTitel}
{$artikelTitel|spacify}
{$artikelTitel|spacify:"^^"}

AUSGABE:

Ein Mann flog 5000 km um sich die Haare schneiden zu lassen. Grund: Seine offensichtlich begnadete Fris
E i n   M a n n   f l o g   5 0 0 0   k m   u m   s i c h   d i e   H a a r e   s c h n e i d e n   z u
E^^i^^n^^ ^^M^^a^^n^^n^^ ^^f^^l^^o^^g^^ ^^5^^0^^0^^0^^ ^^k^^m^^ ^^u^^m^^ ^^s^^i^^c^^h^^ ^^d^^i^^e^^ ^^H
```

string_format (Zeichenkette formatieren)

Parameter Position	Typ	Erfoderlich	Standardwert	Beschreibung
1	string	Ja	*n/a*	Das zu verwendende Format (sprintf).

Wird verwendet um eine Zeichenkette, wie zum Beispiel dezimale Werte, zu formatieren. Folgt der Formatierungs-Syntax von sprintf.

Beispiel 5.18. string_format (Zeichenkette formatieren)

```
{$wert}
{$wert|string_format:"%.2f"}
{$wert|string_format:"%d"}

AUSGABE:

23.5787446
23.58
24
```

strip (Zeichenkette strippen)

Ersetzt mehrfache Leerzeichen, Zeilenumbrüche und Tabulatoren durch ein Leerzeichen oder eine alternative Zeichenkette.

Achtung: Falls Sie ganze Blöcke eines Templates 'strippen' möchten, verwenden Sie dazu strip.

Beispiel 5.19. strip (Zeichenkette strippen)

```
{$artikelTitel}
{$artikelTitel|strip}
{$artikelTitel|strip:" "}

AUSGABE:

Ein 18 Jahre alter Pappkarton
   erzielte  bei                 Ebay einen Erlös von
 536 Dollar. Es war der Karton, in dem der erste Apple verpackt war.
Ein 18 Jahre alter Pappkarton erzielte bei Ebay einen Erlös von 536 Dollar. Es war der Karton, in dem d
Ein 18 Jahre alter Pappkarton erzielte bei Ebay einen Erlö
```

strip_tags

Parameter Position	Typ	Benötigt	Standard	Beschreibung
1	bool	Nein	true	Definiert ob Tags durch ' ' oder " ersetzt werden sollen.

Entfernt alle Markup tags. - Eigentlich alles zwischen < und >.

Beispiel 5.20. strip_tags

```
<?php
$smarty = new Smarty;
$smarty->assign('articleTitle', "Da ein <font face="helvetica">betrunkener Mann</font> auf einem Flug a
$smarty->display('index.tpl');
?>
```

where index.tpl is:

```
{$articleTitle}
{$articleTitle|strip_tags} {* same as {$articleTitle|strip_tags:true} *}
{$articleTitle|strip_tags:false}
```

This will output:

```
Da ein <font face="helvetica">betrunkener Mann</font> auf einem Flug ausfallend wurde, musste <b>das Fl
Da ein betrunkener Mann auf einem Flug ausfallend wurde, musste das Flugzeug auf einer kleinen Insel zw
Da ein <font face="helvetica">betrunkener Mann</font> auf einem Flug ausfallend wurde, musste <b>das Fl
```

truncate (kürzen)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	integer	Nein	80	Länge, auf die die Zeichenkette gekürzt werden soll.
2	string	Nein	...	An die gekürzte Zeichenkette anzuhängende Zeichenkette.
3	boolean	Nein	false	Nur nach ganzen Worten (false) oder exakt an der definierten Stelle (true) kürzen.

Kürzt die Variable auf eine definierte Länge. Standardwert sind 80 Zeichen. Als optionaler zweiter Parameter kann eine Zeichenkette übergeben werden, welche der gekürzten Variable angehängt wird. Diese zusätzliche Zeichenkette wird bei der Berechnung der Länge berücksichtigt. Normalerweise wird 'truncate' versuchen, die Zeichenkette zwischen zwei Wörtern umzubrechen. Um die Zeichenkette exakt an der definierten Position abzuscheiden, können sie als dritten Parameter 'true' übergeben.

Beispiel 5.21. truncate (kürzen)

```
{$artikelTitel}
{$artikelTitel|truncate}
{$artikelTitel|truncate:30}
{$artikelTitel|truncate:30:""}
{$artikelTitel|truncate:30:"---"}
{$artikelTitel|truncate:30:"":true}
{$artikelTitel|truncate:30:"...":true}

AUSGABE:

George W. Bush will die frei gewählten Mitglieder der ICANN ("Internetregierung") durch Regierungsvertr
George W. Bush will die frei gewählten Mitglieder der ICANN ("Internetregierung") durch Regierungsvertr
George W. Bush will die frei...
George W. Bush will die frei
George W. Bush will die frei---
George W. Bush will die frei
George W. Bush will die fr...
```

upper (in Grossbuchstaben umwandeln)

Wandelt eine Zeichenkette in Grossbuchstaben um.

Beispiel 5.22. upper (in Grossbuchstaben umwandeln)

```
{$artikelTitel}
{$artikelTitel|upper}

AUSGABE:

Ein 58jähriger Belgier ist nach 35 Jahren zum Sieger der Weltmeisterschaft im Querfeldeinrennen 1967 er
EIN 58JÄHRIGER BELGIER IST NACH 35 JAHREN ZUM SIEGER DER WELTMEISTERSCHAFT IM QUERFELDEINRENNEN 1967 ER
```

wordwrap (Zeilenumbruch)

Parameter Position	Typ	Erforderlich	Standardwert	Beschreibung
1	integer	Nein	80	Definiert maximale Länge einer Zeile in der umzubrechenden Zeichenkette.
2	string	Nein	\n	Definiert das zu verwendende Zeichen.
3	boolean	Nein	false	Definiert ob die Zeichenkette nur zwischen Wörtern getrennt (false), oder auch abgeschnitten werden darf (true).

Bricht eine Zeichenkette an einer definierten Stelle (Standardwert 80) um. Als optionaler zweiter Parameter kann das Zeichen übergeben werden, welches zum Umbrechen verwendet werden soll (Standardwert '\n'). Normalerweise bricht wordwrap nur zwischen zwei Wörtern um. Falls Sie exakt an der definierten Stelle umbrechen wollen, übergeben Sie als optionalen dritten Parameter 'true'.

Beispiel 5.23. wordwrap (Zeilenumbruch)

```
{$artikelTitel}

{$artikelTitel|wordwrap:75}

{$artikelTitel|wordwrap:50}

{$artikelTitel|wordwrap:75:"<br>\n"}

{$artikelTitel|wordwrap:75:"\n":true}

AUSGABE:

Eine Frau stahl in einem Bekleidungsgeschäft eine Hose und kam kurz danach zurück, um die Hose umzutaus

Eine Frau stahl in einem Bekleidungsgeschäft eine Hose und kam kurz
danach zurück, um die Hose umzutauschen, weil die Grösse nicht
passte.

Eine Frau stahl in einem Bekleidungsgeschäft
eine Hose und kam kurz danach zurück, um die
Hose umzutauschen, weil die Grösse nicht
passte.

Eine Frau stahl in einem Bekleidungsgeschäft eine Hose und kam kurz<br>
danach zurück, um die Hose umzutauschen, weil die Grösse nicht<br>
passte.

Eine Frau stahl in einem Bekleidungsgeschäft eine Hose und kam kurz d
anach zurück, um die Hose umzutauschen, weil die Grösse nicht pass
te.
```

Kapitel 6. Kombinieren von Modifikatoren

Sie können auf eine Variable so viele Modifikatoren anwenden wie Sie möchten. Die Modifkatoren werden in der Reihenfolge angewandt, in der sie notiert wurden - von links nach rechts. Kombinierte Modifikatoren müssen mit einem `|`-Zeichen (pipe) getrennt werden.

Beispiel 6.1. Kombinieren von Modifikatoren

```
<?php

$smarty->assign('articleTitle',
    'Einem Stadtrat in Salem in Pennsylvania (USA) droht eine
     zweijährige Haftstrafe, da eine von ihm gehaltene Rede sechs
     Minuten länger dauerte, als erlaubt. Die Redezeit ist auf maximal
     fünf Minuten begrenzt.');

?>
```

Wobei das Template dann folgendes entält:

```
{$articleTitle}
{$articleTitle|upper|spacify}
{$articleTitle|lower|spacify|truncate}
{$articleTitle|lower|truncate:30|spacify}
{$articleTitle|lower|spacify|truncate:30:". . ."}
```

AUSGABE:

```
Einem Stadtrat in Salem in Pennsylvania (USA) droht eine (usw.)
EINEM STADTRAT IN SALEM IN PENNSYLVANIA (USA) DROHT EINE (usw.)
e i n e m   s t a d t r a t   i n   s a l e m   i n...
e i n e m   s t a d t r a t   i n   s a l e m   i n . . .
e i n e m   s t a d t r. . .
```

Kapitel 7. Eingebaute Funktionen

Inhaltsverzeichnis

Smarty enthält eine Reihe eingebauter Funktionen. Eingebaute Funktionen sind integral für die Template-Sprache. Sie können sie weder verändern noch eigene Funktionen unter selbem Namen erstellen.

{capture} (Ausgabe abfangen)

{capture} wird verwendet, um die Template-Ausgabe abzufangen und in einer Variable zu speichern. Der Inhalt zwischen {capture name="foo"} und {/capture} wird unter der im 'name' Attribut angegebenen Capture-Variablen abgelegt und kann über $smarty.capture.foo angesprochen werden. Falls kein 'name'-Attribut übergeben wurde, wird der Inhalt in 'default' (also $smarty.capture.default) abgelegt. Jede {capture} Sektion muss mit {/capture} beendet werden. {capture}-Blöcke können verschachtelt sein.

Attribut Name	Typ	Benötigt	Standardwert	Beschreibung
name	string	no	*default*	Der Name des abgefangenen Blocks
assign	string	No	*n/a*	Der Name der Variable welcher der Wert zugewiesen werden soll.

Achtung

Seien Sie vorsichtig, wenn sie die Ausgabe von {insert} abfangen wollen. Sie sollten die Ausgabe nicht abfangen, wenn Caching eingeschaltet ist und Sie einen {insert} Befehl verwenden, um Ausgaben vom Caching auszuschliessen.

Beispiel 7.1. Template-Inhalte abfangen

```
{* Tabellenzeile nur ausgeben, wenn Inhalt vorhanden *}
{capture name=banner}
{include file='get_banner.tpl'}
```

```
{/capture}
{if $smarty.capture.banner ne ""}
<table>
<tr>
  <td>
    {$smarty.capture.banner}
  </td>
</tr>
</table>
{/if}
```

Beispiel 7.2. Template-Inhalte abfangen

Hier ist ein Beispiel das das Zusammenspiel mit der Funktion {popup} demonstriert.

```
{capture name=some_content assign=popText}
    .... some content ....
  {/capture}

  <a href="#" {popup caption='Help' text=$popText}>help</a>
```

Siehe auch: $smarty.capture, {eval}, {fetch}, fetch() and {assign}.

{config_load} (Konfiguration laden)

Diese Funktion wird verwendet, um Variablen aus einer Konfigurationsdatei in das Template zu laden. Sehen sie Config Files (Konfigurationsdateien) für weitere Informationen.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
file	string	Ja	*n/a*	Definiert den Namen der einzubindenden Datei.
section	string	Nein	*n/a*	Definiert den Namen des zu ladenden Abschnitts.
scope	string	Nein	*local*	Definiert den Geltungsbereich der zu ladenden Variablen. Erlaubte Werte sind 'local','parent' und 'global'. 'local' bedeutet, dass die Variablen in den Context des lokalen Template geladen werden. 'parent' bedeutet, dass die Variablen sowohl in den lokalen Context, als auch in den Context des aufrufenden Templates eingebunden werden. 'global' bedeutet, dass die Variablen von allen

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				Templates zugänglich sind.
global	boolean	Nein	*No*	Definiert, ob die Variablen von allen Templates aus zugänglich sind. WICHTIG: Dieses Attribut wird von 'scope' abgelöst und sollte nicht mehr verwendet werden. Falls 'scope' übergeben wurde, wird 'global' ignoriert.

Beispiel 7.3. Funktion {config_load}

beispiel.conf

```
#Dies ist ein Konfigurationsdateikommentar

# globale Variablen
seitenTitel = "Hauptmenü"
bodyHintergrundFarbe = #000000
tabelleHintergrundFarbe = #000000
reiheHintergrundFarbe = #00ff00

# Kundenvariablen
[Kunden]
seitenTitel = "Kundeninfo"
```

and the template

```
{config_load file='example.conf'}

<html>
<title>{#seitenTitel#}</title>
<body bgcolor="{#bodyHintergrundFarbe#}">
<table border="{#tabelleRahmenBreite#}" bgcolor="{#tabelleHintergrundFarbe#}">
  <tr bgcolor="{#reiheHintergrundFarbe#}">
    <td>Vornamen</td>
    <td>Nachnamen</td>
    <td>Adresse</td>
  </tr>
</table>
</body>
</html>
```

Konfigurationsdateien können Abschnitte enthalten. Um Variablen aus einem Abschnitt zu laden, können Sie das Attribut *section* übergeben.

Bemerkung: *Konfigurationdatei-Abschnitte (sections)* und die eingebaute Template Funktion namens section haben ausser dem Namen nichts gemeinsam.

Beispiel 7.4. Funktion {config_load} mit Abschnitten

```
{config_load file="beispiel.conf" section="Kunde"}
<html>
<title>{#seitenTitel#}</title>
<body bgcolor="{#bodyHintergrundFarbe#}">
<table border="{#tabelleRahmenBreite#}" bgcolor="{#tabelleHintergrundFarbe#}">
  <tr bgcolor="{#reiheHintergrundFarbe#}">
    <td>Vornamen</td>
    <td>Nachnamen</td>
    <td>Adresse</td>
  </tr>
</table>
</body>
</html>
```

Siehe $config_overwrite bezüglich Arrays von Konfigurationsvariablen.

Siehe auch Konfigurationsdateien, Variablen aus Konfigurationsdateien, $config_dir, get_config_vars() und config_load().

{foreach}, {foreachelse}

Die *foreach* Schleife ist eine Alternative zu *section*. *foreach* wird verwendet, um ein assoziatives Array zu durchlaufen. Die Syntax von *foreach*-Schleifen ist viel einfacher als die von *section*. *{foreach}* Tags müssen mit *{/foreach}* tags kombiniert werden. Erforderliche Parameter sind: *from* und *item*. Der Name der {foreach}-Schleife kann frei vergeben werden und sowohl Buchstaben, Zahlen als auch Unterstriche enthalten. *foreach*-Schleifen können verschachtelt werden, dabei ist zu beachten, dass sich die definierten Namen voneinander unterscheiden. Die *from* Variable (normalerweise ein assoziatives Array) definiert die Anzahl der von *foreach* zu durchlaufenen Iterationen. *foreachelse* wird ausgeführt wenn keine Werte in der *from* Variable übergeben wurden.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
from	string	Ja	*n/a*	Name des zu durchlaufenden Array.
item	string	Ja	*n/a*	Name für das aktuelle Element.
key	string	Nein	*n/a*	Name für den aktuellen Schlüssel.
name	string	Nein	*n/a*	Name der 'foreach'-Schleife, für die Abfrage der 'foreach'-Eigenschaften .

Beispiel 7.5. {foreach} - item

```
<?php
$arr = array( 1001,1002,1003);
$smarty->assign('custid', $arr);
?>
```

```
<?php
{* dieses Beispiel gibt alle Werte aus dem $KundenId Array aus *}
{foreach from=$KundenId item=aktuelle_id}
 id: {$aktuelle_id}<br>
{/foreach}
```

Das obige Beispiel erzeugt folgende Ausgabe:

```
id: 1000<br>
id: 1001<br>
id: 1002<br>
```

Beispiel 7.6. {foreach} - item und key

```
// Der Schlüssel enthält den Schlüssel des jeweils iterierten Wertes
// die Zuweisung sieht wie folgt aus:
<?php
 $smarty->assign('kontakte', array(
                             array('phone' => '1',
                                   'fax' => '2',
                                   'cell' => '3'),
                             array('phone' => '555-4444',
                                   'fax' => '555-3333',
                                   'cell' => '760-1234')
                             ));
?>
```

```
{foreach name=aussen item=kontakt from=$kontakte}
  <hr />
  {foreach key=schluessel item=wert from=$kontakt}
    {$schluessel}: {$wert}<br>
  {/foreach}
{/foreach}
  </programlisting>
  <para>
   Das obige Beispiel erzeugt folgende Ausgabe:
  </para>
  <screen>
<![CDATA[
<hr />
   phone: 1<br>
   fax: 2<br>
   cell: 3<br>
<hr />
   phone: 555-4444<br>
   fax: 555-3333<br>
   cell: 760-1234<br>
```

Beispiel 7.7. {foreach} - Beispiel mit Datenbankzugriff (z.B. PEAR oder ADODB)

```
<?php
  $sql = 'SELECT contact_id, name, nick FROM contacts ORDER BY contact';
  $smarty->assign('kontakte', $db->getAssoc($sql));
?>
```

```
{foreach key=cid item=con from=$kontakte}
  <a href="kontact.php?contact_id={$cid}">{$con.name} - {$con.nick}</a><br />
{/foreach}
```

Foreach-Loops haben auch eigene Variablen welche die Foreach Eigenschaften enthalten. Diese werden wie folgt ausgewiesen: {$smarty.foreach.foreachname.varname}. foreachname ist der Name der als *name* Attribut von Foreach übergeben wurden.

iteration

gibt die aktuelle iteration aus

iteration beginnt immer mit 1 und wird danach bei jedem durchgang um 1 inkrementiert.

first

first ist TRUE wenn die aktuelle Iteration die erste ist

last

last ist TRUE wenn die aktuelle Iteration die letzte ist

show

show wird als Parameter von foreach verwedet und ist ein boolscher Wert, TRUE oder FALSE. Auf FALSE wird nichts ausgegeben und wenn foreachelse gefunden wird, dieser angezeigt.

total

total gibt die Anzahl Iterationen des Foreach Loops aus und kann in- oder nach- Foreach Blöcken verwendet werden.

{if},{elseif},{else}

{if}-Statements in Smarty erlauben die selbe Flexibilität wie in PHP, bis auf ein paar Erweiterungen für die Template-Engine. Jedes *{if}* muss mit einem *{/if}* kombiniert sein. *{else}* und *{elseif}* sind ebenfalls erlaubt. Alle PHP Vergleichsoperatoren und Funktionen, wie ||, *or*, *&&*, *and*, *is_array()*, etc. sind erlaubt.

Wenn $security angeschaltet wurde, dann müssen alle verwendeten PHP-Funktionen im *IF_FUNCS*-Array in dem $security_settings-Array deklariert werden.

Hier eine Liste der erlaubten Operatoren. Bedingungsoperatoren müssen von umgebenden Elementen mit Leerzeichen abgetrennt werden. PHP-Äquivalente sind, sofern vorhanden, angeben.

Operator	Alternativen	Syntax Beispiel	Bedeutung	PHP Äquivalent
==	eq	$a eq $b	ist gleich	==
!=	ne, neq	$a neq $b	ist ungleich	!=
>	gt	$a gt $b	größer als	>
<	lt	$a lt $b	kleiner als	<
>=	gte, ge	$a ge $b	größer oder gleich	>=
<=	lte, le	$a le $b	kleiner oder gleich	<=
===		$a === 0	identisch	===
!	not	not $a	Negation	!
%	mod	$a mod $b	Modulo	%
is [not] div by		$a is not div by 4	Ist [nicht] teilbar durch	$a % $b == 0
is [not] even		$a is not even	ist [k]eine gerade Zahl	$a % 2 == 0
is [not] even by		$a is [not] even by $b	[k]eine gerade Gruppierung	($a / $b) % 2 == 0

Operator	Alternativen	Syntax Beispiel	Bedeutung	PHP Äquivalent
is [not] odd		$a is not odd	ist [k]eine ungerade Zahl	$a % 2 != 0
is [not] odd by		$a is not odd by $b	[k]eine ungerade Gruppierung	($a / $b) % 2 != 0

Beispiel 7.8. if Anweisung

```
{* ein Beispiel mit 'eq' (gleich) *}
{if $name eq "Fred"}
        Willkommen der Herr.
{elseif $name eq "Wilma"}
        Willkommen die Dame.
{else}
        Willkommen, was auch immer Du sein magst.
{/if}

{* ein Beispiel mit 'or'-Logik *}
{if $name eq "Fred" or $name eq "Wilma"}
        ...
{/if}

{* das selbe *}
{if $name == "Fred" || $name == "Wilma"}
        ...
{/if}

{*
  die foldende Syntax ist nicht korrekt, da die Elemente welche die
  Bedingung umfassen nicht mit Leerzeichen abgetrennt sind
*}
{if $name=="Fred" || $name=="Wilma"}
        ...
{/if}


{* Klammern sind erlaubt *}
{if ( $anzahl < 0 or $anzahl > 1000 ) and $menge >= #minMengeAmt#}
        ...
{/if}


{* einbetten von php Funktionsaufrufen ('gt' steht für 'grösser als') *}
{if count($var) gt 0}
        ...
{/if}

{* Auf "ist array" überprüfen. *}
{if is_array($foo) }
   .....
{/if}

{* Auf "ist nicht null" überprüfen. *}
{if isset($foo) }
   .....
{/if}



{* testen ob eine Zahl gerade (even) oder ungerade (odd) ist *}
{if $var is even}
        ...
{/if}
{if $var is odd}
        ...
{/if}
```

```
{if $var is not odd}
        ...
{/if}


{* testen ob eine Zahl durch 4 teilbar ist (div by) *}
{if $var is div by 4}
        ...
{/if}


{* testen ob eine Variable gerade ist, gruppiert nach 2
   0=gerade, 1=gerade, 2=ungerade, 3=ungerade, 4=gerade, 5=gerade, etc *}
{if $var is even by 2}
        ...
{/if}

{* 0=gerade, 1=gerade, 2=gerade, 3=ungerade, 4=ungerade, 5=ungerade, etc *}
{if $var is even by 3}
        ...
{/if}
```

include (einbinden)

{include}-Tags werden verwendet, um andere Templates in das aktuelle Template einzubinden. Alle Variablen des aktuellen Templates sind auch im eingebundenen Template verfügbar. Das {include}-Tag muss ein 'file' Attribut mit dem Pfad zum einzubindenden Template enthalten.

Optional kann mit dem *assign* Attribut definiert werden, in welcher Variable die Ausgabe des mit *include* eingebundenen Templates abgelegt werden soll statt sie auszugeben.

Die Werte aller zugewiesenen Variablen werden wiederhergestellt, sobald ein eingebundenes Template wieder verlassen wurde. Das bedeutet, dass in einem eingebundenen Template alle Variablen des einbindenden Template verwendet und verändert werden können, diese Änderungen aber verloren sind, sobald das {include} abgearbeitet wurde.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
file	string	Ja	*n/a*	Name der Template-Datei, die eingebunden werden soll.
assign	string	Nein	*n/a*	Variable, welcher der eingebundene Inhalt zugewiesen werden soll.
[var ...]	[var typ]	Nein	*n/a*	Variablen welche dem Template lokal übergeben werden sollen.

Beispiel 7.9. function include (einbinden)

```
<html>
<head>
  <title>{$title}</title>
</head>
<body>
{include file='page_header.tpl'}
```

```
{* hier kommt der body des Templates *}
{include file="$tpl_name.tpl"} <-- $tpl_name wird durch eine Wert ersetzt

{include file='page_footer.tpl'}
</body>
</html>
```

Sie können dem einzubindenden Template Variablen als Attribute übergeben. Alle explizit übergebenen Variablen sind nur im Anwendungsbereich (scope) dieses Template verfügbar. Attribut-Variablen überschreiben aktuelle Template-Variablen, falls sie den gleichen Namen haben.

Beispiel 7.10. include-Funktion und Variablen Übergabe

```
{include file='header.tpl' title='Hauptmenu' table_bgcolor='#c0c0c0'}

{* hier kommt der body des Templates *}

{include file='footer.tpl' logo='http://my.domain.com/logo.gif'}
```

Benutzen sie die Syntax von template resources, um Templates ausserhalb des '$template_dir' einzubinden:

Beispiel 7.11. Beispiele für Template-Ressourcen bei der 'include'-Funktion

```
{* absoluter Dateipfad *}
{include file='/usr/local/include/templates/header.tpl'}

{* absoluter Dateipfad (gleich) *}
{include file='file:/usr/local/include/templates/header.tpl'}

{* absoluter Dateipfad unter Windows ("file:"-Prefix MUSS übergeben werden) *}
{include file='file:C:/www/pub/templates/header.tpl'}

{* einbinden aus Template-Ressource namens 'db' *}
{include file='db:header.tpl'}

{* einbinden eines Variablen Templates - z.B. $module = 'contacts' *}
{include file="$module.tpl"}
{*
  Dies hier Funktioniert nicht, da Variablen innerhalb einfacher
  Anführungszeichen nicht interpoliert werden.
*}
{include file='$module.tpl'}
```

Siehe auch {include_php}, {php}, Template Ressourcen und Template/Skript Komponenten.

include_php (PHP-Code einbinden)

Die Verwendung von {include_php} wird nicht mehr empfohlen, die gleiche funktionalität kann auch mit Template/Script Komponenten erreicht werden.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
file	string	Ja	*n/a*	Der Name der einzubindenden PHP-Datei.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
once	boolean	Nein	*true*	Definiert ob die Datei mehrmals geladen werden soll, falls sie mehrmals eingebunden wird.
assign	string	Nein	*n/a*	Der Name der Variable, der die Ausgabe von include_php zugewiesen wird.

Falls Sicherheit aktiviert ist, muss das einzubindende Skript im $trusted_dir Pfad liegen. {include_php} muss das Attribut 'file' übergeben werden, das den Pfad - entweder relativ zu $trusted_dir oder absolut - zum Skript enthält.

Normalerweise wird ein PHP-Skript nur einmal pro Aufruf geladen, selbst wenn es mehrfach eingebunden wird. Sie können dieses Verhalten durch die Verwendung des *once* Attributs steuern. Wenn Sie 'once' auf 'false' setzen, wird die Datei immer wenn sie eingebunden wird auch neu geladen.

Optional kann das *assign* Attribut übergeben werden. Die Ausgabe von *include_php* wird dann nicht direkt eingefügt, sondern in der durch assign benannten Template-Variable abgelegt.

Das Objekt '$smarty' kann in dem eingebundenen PHP-Script über '$this' angesprochen werden.

Beispiel 7.12. Funktion include_php

lade_nav.php

```
<?php

 // lade die Variablen aus einer MySQL-Datenbank und weise sie dem Template zu
 require_once("MySQL.class.php");
 $sql = new MySQL;
 $sql->query("select * from site_nav_sections order by name",SQL_ALL);
 $this->assign($sections,$sql->record);

?>
```

Bei folgendem index.tpl:

```
{* absoluter Pfad, oder relativ zu '$trusted_dir' *}
{include_php file="/pfad/zu/lade_nav.php"}

{foreach item=$aktuelle_section from=$sections}
 <a href="{$aktuelle_section.url}">{$aktuelle_section.name}</a><br>
{/foreach}
```

Siehe auch {include}, {php}, {capture}, Template Ressourcen and Template/Script Komponenten

insert (einfügen)

{insert}-Tags funktionieren ähnlich den {include}-Tags, werden aber nicht gecached, falls caching eingeschaltet ist. Sie werden bei jedem Aufruf des Templates ausgeführt.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Ja	*n/a*	Der Name der Insert-Funktion
assign	string	Nein	*n/a*	Name der Template-Variable, in der die Ausgabe der 'insert'-Funktion optional abgelegt wird.
script	string	Nein	*n/a*	Name des PHP-Skriptes, das vor Aufruf der 'insert'-Funktion eingebunden werden soll.
[var ...]	[var typ]	Nein	*n/a*	Variablen die der 'insert'-Funktion übergeben werden sollen.

Stellen Sie sich vor, sie hätten ein Template mit einem Werbebanner. Dieser Banner kann verschiedene Arten von Inhalten haben: Bilder, HTML, Flash, etc. Deshalb können wir nicht einfach einen statischen Link verwenden und müssen vermeiden, dass dieser Inhalt gecached wird. Hier kommt das {insert}-Tag ins Spiel. Das Template kennt die Variablen '#banner_location_id#' und '#site_id#' (zum Beispiel aus einer Konfigurationsdatei) und soll eine Funktion aufrufen, die den Inhalt des Banners liefert.

Beispiel 7.13. Funktion 'insert'

```
{* erzeugen des Banners *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

In diesem Beispiel verwenden wir die Funktion 'getBanner' und übergeben die Parameter '#banner_location_id#' und '#site_id#'. Smarty wird daraufhin in Ihrer Applikatiopn nach einer Funktion namens 'getBanner' suchen und diese mit den Parametern '#banner_location_id#' und '#site_id#' aufrufen. Allen 'insert'-Funktionen in Ihrer Applikation muss 'insert_' vorangestellt werden, um Konflikte im Namensraum zu vermeiden. Ihre 'insert_getBanner()'-Funktion sollte etwas mit den übergebenen Parametern unternehmen und das Resultat zurückgeben. Dieses Resultat wird an der Stelle des 'insert'-Tags in Ihrem Template ausgegeben. In diesem Beispiel würde Smarty folgende Funktion aufrufen: insert_getBanner(array("lid" => "12345","sid" => "67890")) und die erhaltenen Resultate an Stelle des 'insert'-Tags ausgeben.

Falls Sie das 'assign'-Attribut übergeben, wird die Ausgabe des 'insert'-Tags in dieser Variablen abgelegt. Bemerkung: dies ist nicht sinnvoll, wenn Caching eingeschaltet ist.

Falls Sie das 'script'-Attribut übergeben, wird das angegebene PHP-Skript vor der Ausführung der {insert}-Funktion eingebunden. Dies ist nützlich, um die {insert}-Funktion erst in diesem Skript zu definieren. Der Pfad kann absolut oder relativ zu $trusted_dir angegeben werden. Wenn Sicherheit eingeschaltet ist, muss das Skript in $trusted_dir liegen.

Als zweites Argument wird der {insert}-Funktion das Smarty-Objekt selbst übergeben. Damit kann dort auf die Informationen im Smarty-Objekt zugegriffen werden.

> **Technische Bemerkung:** Es gibt die Möglichkeit, Teile des Templates nicht zu cachen. Wenn Sie caching eingeschaltet haben, werden {insert}-Tags nicht gecached. Sie werden jedesmal ausgeführt, wenn die Seite erstellt wird - selbst innerhalb gecachter Seiten. Dies funktioniert gut für Dinge wie Werbung (Banner), Abstimmungen, Wetterberichte, Such-Resultate, Benutzer-Feedback-Ecke, etc.

ldelim,rdelim (Ausgabe der Trennzeichen)

ldelim und rdelim werden verwendet, um die Trennzeichen auszugeben - in unserem Fall "{" oder "}" - ohne dass Smarty versucht, sie zu interpretieren. Um text im Template vor dem Interpretieren zu schÃ¼tzen kann auch {literal}{/literal} verwendet werden. Siehe auch {$smarty.ldelim}.

Beispiel 7.14. ldelim, rdelim

```
{* gibt die konfigurierten Trennzeichen des Templates aus *}

{ldelim}funktionsname{rdelim} Funktionen sehen in Smarty so aus!
```

Das obige Beispiel ergibt als Ausgabe:

```
{funktionsname} Funktionen sehen in Smarty so aus!</programlisting>
```

Ein weiteres Beispiel (diesmal mit javascript)

```
<script language="JavaScript">
function foo() {ldelim}
    ... code ...
{rdelim}
</script>
```

Ausgabe:

```
<script language="JavaScript">
function foo() {
    .... code ...
}
</script>
```

Siehe auch Smarty Parsing umgehen

literal

{literal}-Tags erlauben es, einen Block wörtlich auszugeben, d.h. von der Interpretation durch Smarty auszuschliessen. Dies ist vor allem für Javascript- oder andere Blöcke nützlich, die geschwungene Klammern verwenden. Alles was zwischen den {literal}{/literal} Tags steht, wird direkt angezeigt. Wenn in einem {literal}-Block temlate-Tags verwendet werden sollen, is es manchmal sinnvoller {ldelim}{rdelim} statt {literal} zu verwenden.

Beispiel 7.15. literal-Tags

```
{literal}
<script language=javascript>
<!--
  function isblank(field) {
    if (field.value == '') {
      return false;
    } else {
      document.loginform.submit();
      return true;
    }
  }
// -->
</script>
```

```
{/literal}
```

Siehe auch Smarty Parsing umgehen.

php

{php}-Tags erlauben es, PHP-Code direkt in das Template einzubetten. Der Inhalt wird nicht 'escaped', egal wie $php_handling konfiguriert ist. Dieses Tag ist nur für erfahrene Benutzer gedacht und wird auch von diesen normalerweise nicht benötigt.

Beispiel 7.16. {php}-Tags

```
{php}
  // php Skript direkt von Template einbinden
  include('/pfad/zu/zeige_weather.php');
  {/php}
```

> **Technical Note:** Um auf PHP-Variablen in {php}-Blöcken zugreifen zu können, kann es nötig sein, die Variable als global [http://php.net/global] zu deklarieren. Der {php}-Blöck läuft nämlich nicht in einem globalen Kontext, sondern im Kontext der method des laufenden $smarty-Objektes.

Beispiel 7.17. {php} mit Verwendung von global

```
{php}
   global $foo, $bar;
   if($foo == $bar){
     // tue irgendwas
   }
{/php}
```

Siehe auch $php_handling, {include_php}, {include} und Template/Script Komponenten.

section,sectionelse

Template-{sections} werden verwendet, um durch **Arrays** zu iterieren (ähnlich wie {foreach}). Jedes *section*-Tag muss mit einem */section*-Tag kombiniert werden. *name* und *loop* sind erforderliche Parameter. Der Name der 'section' kann frei gewählt werden, muss jedoch aus Buchstaben, Zahlen oder Unterstrichen bestehen. {sections} können verschachtelt werden. Dabei ist zu beachten, dass sich ihre Namen unterscheiden. Aus der 'loop'-Variable (normalerweise ein Array von Werten) resultiert die Anzahl der Iterationen, die durchlaufen werden. Wenn ein Wert aus der 'loop'-Variable innerhalb der {section} ausgegeben werden soll, muss der 'section-name' umschlossen mit [] angefügt werden. *sectionelse* wird ausgeführt, wenn keine Werte in der 'loop'-Variable enthalten sind.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Ja	*n/a*	Der Name der 'section'
loop	[$variable_name]	Ja	*n/a*	Der Name des Zählers für die Iterationen.
start	integer	Nein	*0*	Definiert die Startposition. Falls ein negativer Wert

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				übergeben wird, berechnet sich die Startposition ausgehend vom Ende des Arrays. Wenn zum Beispiel 7 Werte in einem Array enthalten sind und die Startposition -2 ist, ist die berechnete Startposition 5. Unerlaubte Werte (Werte ausserhalb der Grösse des Arrays) werden automatisch auf den nächstmöglichen Wert gesetzt.
step	integer	Nein	*1*	Definiert die Schrittweite mit welcher das Array durchlaufen wird. 'step=2' iteriert durch 0, 2, 4, etc. Wenn ein negativer Wert übergeben wurde, wird das Array rückwärts durchlaufen.
max	integer	Nein	*n/a*	Maximale Anzahl an Iterationen, die Durchlaufen werden.
show	boolean	Nein	*true*	Definiert ob diese 'section' angezeigt werden soll oder nicht.

Beispiel 7.18. section

```
<?php

$data = array(1000,1001,1002);
$smarty->assign('custid',$data);

?>
```

```
{* dieses Beispiel gibt alle Werte des $KundenId Arrays aus *}
{section name=kunde loop=$KundenId}
id: {$KundenId[kunde]}<br />
{/section}
{* alle Werte in umgekehrter Reihenfolge ausgeben: *}
{section name=kunde loop=$KundenId step=-1}
id: {$KundenId[kunde]}<br />
{/section}
```

Ausgabe des obigen Beispiels:

```
id: 1000<br />
```

```
id: 1001<br />
id: 1002<br />
<hr />
id: 1002<br />
id: 1001<br />
id: 1000<br />
```

Ein weiteres Beispiel, diesmal ohne ein zugewiesenes Array.

```
{section name=foo start=10 loop=20 step=2}
  {$smarty.section.foo.index}
{/section}
<hr />
{section name=bar loop=21 max=6 step=-2}
  {$smarty.section.bar.index}
{/section}
```

Ausgabe des obigen Beispiels:

```
10 12 14 16 18
<hr />
20 18 16 14 12 10
```

Beispiel 7.19. section loop Variable

```
{* die 'loop'-Variable definiert nur die Anzahl der Iterationen,
   Sie können in dieser 'section' auf jeden Wert des Templates
   zugreifen. Dieses Beispiel geht davon aus, dass $KundenId, $Namen und
   $Adressen Arrays sind, welche die selbe Anzahl Werte enthalten *}
{section name=kunde loop=$KundenId}
id: {$KundenId[kunde]}<br>
name: {$Namen[kunde]}<br>
address: {$Adressen[kunde]}<br>
<p>
{/section}
```

Ausgabe des obigen Beispiels:

```
id: 1000<br>
name: Peter Müller <br>
adresse: 253 N 45th<br>
<p>
id: 1001<br>
name: Fritz Muster<br>
adresse:: 417 Mulberry ln<br>
<p>
id: 1002<br>
name: Hans Meier<br>
adresse:: 5605 apple st<br>
<p>
```

Beispiel 7.20. section names

```
{*
  die 'name'-Variable definiert den Namen der verwendet werden soll,
  um Daten aus dieser 'section' zu referenzieren
*}
{section name=meinedaten loop=$KundenId}
```

```
<p>
  id: {$KundenId[meinedaten]}<br>
  name: {$Namen[meinedaten]}<br>
  address: {$Adressen[meinedaten]}
</p>
{/section}
```

Beispiel 7.21. nested sections (verschachtelte 'sections')

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

$types = array(
           array( 'home phone', 'cell phone', 'e-mail'),
           array( 'home phone', 'web'),
           array( 'cell phone')
         );
$smarty->assign('contact_type', $types);

$info = array(
           array('555-555-5555', '666-555-5555', 'john@myexample.com'),
           array( '123-456-4', 'www.example.com'),
           array( '0457878')
        );
$smarty->assign('contact_info', $info);

?>
```

```
{*
  Sections können unbegrenzt tief verschachtelt werden.  Mit
  verschachtelten 'sections' können Sie auf komplexe Datenstrukturen
  zugreifen (wie zum Beispiel multidimensionale Arrays). Im folgenden
  Beispiel ist $contact_type[customer] ein Array mit Kontakttypen des
  aktuellen Kunden.
*}
{section name=customer loop=$custid}
<hr />
  id: {$custid[customer]}<br />
  name: {$name[customer]}<br />
  address: {$address[customer]}<br />
  {section name=contact loop=$contact_type[customer]}
    {$contact_type[customer][contact]}: {$contact_info[customer][contact]}<br />
  {/section}
{/section}
```

Ausgabe des obigen Beispiels:

```
<hr />
  id: 1000<br />
  name: John Smith<br />
  address: 253 N 45th<br />
    home phone: 555-555-5555<br />
    cell phone: 666-555-5555<br />
    e-mail: john@myexample.com<br />
<hr />
  id: 1001<br />
```

```
  name: Jack Jones<br />
  address: 417 Mulberry ln<br />
    home phone: 123-456-4<br />
    web: www.example.com<br />
<hr />
  id: 1002<br />
  name: Jane Munson<br />
  address: 5605 apple st<br />
    cell phone: 0457878<br />
```

Beispiel 7.22. sections und assoziative Arrays

```
<?php

$data = array(
          array('name' => 'John Smith', 'home' => '555-555-5555',
                'cell' => '666-555-5555', 'email' => 'john@myexample.com'),
          array('name' => 'Jack Jones', 'home' => '777-555-5555',
                'cell' => '888-555-5555', 'email' => 'jack@myexample.com'),
          array('name' => 'Jane Munson', 'home' => '000-555-5555',
                'cell' => '123456', 'email' => 'jane@myexample.com')
        );
$smarty->assign('contacts',$data);

?>
```

```
{*
  Dies ist ein Beispiel wie man einen assoziativen Array in einer
  'section' ausgeben kann.
*}
{section name=customer loop=$contacts}
<p>
name: {$contacts[customer].name}<br />
home: {$contacts[customer].home}<br />
cell: {$contacts[customer].cell}<br />
e-mail: {$contacts[customer].email}
</p>
{/section}

{* Anm. d. übersetzers: Oft ist die Anwendung von 'foreach' kürzer. *}

{foreach item=customer from=$contacts}
<p>
name: {$customer.name}<br />
home: {$customer.home}<br />
cell: {$customer.cell}<br />
e-mail: {$customer.email}
</p>
{/foreach}
```

Ausgabe des obigen Beispiels:

```
<p>
name: John Smith<br />
home: 555-555-5555<br />
cell: 555-555-5555<br />
e-mail: john@mydomain.com
</p>
<p>
name: Jack Jones<br />
home phone: 555-555-5555<br />
cell phone: 555-555-5555<br />
e-mail: jack@mydomain.com
<p>
```

```
name: Jane Munson<br />
home phone: 555-555-5555<br />
cell phone: 555-555-5555<br />
e-mail: jane@mydomain.com
</p>
```

Beispiel 7.23. sectionelse

```
{*
  sectionelse wird aufgerufen, wenn keine $custid Werte vorhanden sind
*}
{section name=customer loop=$custid}

id: {$custid[customer]}<br>
{sectionelse}
keine Werte in $custid gefunden
{/section}
```

Die Eigenschaften der 'section' werden in besonderen Variablen abgelegt. Diese sind wie folgt aufgebaut: {$smarty.section.sectionname.varname}

> **Anmerkung:** Bermerkung: Seit Smarty 1.5.0 hat sich die Syntax der 'section' Eigenschaften von {%sectionname.varname%} zu {$smarty.section.sectionname.varname} geändert. Die alte Syntax wird noch immer unterstützt, die Dokumentation erwähnt jedoch nur noch die neue Schreibweise.

index

'index' wird verwendet, um den aktuellen Schleifen-Index anzuzeigen. Er startet bei 0 (beziehungsweise der definierten Startposition) und inkrementiert in 1-er Schritten (beziehungsweise der definierten Schrittgrösse).

> **Technische Bemerkung:** Wenn 'step' und 'start' nicht übergeben werden, verhält sich der Wert wie die 'section'-Eigenschaft 'iteration', ausser dass er bei 0 anstatt 1 beginnt.

Beispiel 7.24. 'section'-Eigenschaft 'index'

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
```

Ausgabe des obigen Beispiels:

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
```

index_prev

'index_prev' wird verwendet um den vorhergehenden Schleifen-Index auszugeben. Bei der ersten Iteration ist dieser Wert -1.

Beispiel 7.25. section'-Eigenschaft 'index_prev'

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* zur Information, $custid[customer.index] und $custid[customer] bedeuten das selbe *}
{if $custid[customer.index_prev] ne $custid[customer.index]}
    Die Kundennummer hat sich geändert.<br>
{/if}
{/section}
```

Ausgabe des obigen Beispiels:

```
0 id: 1000<br>
    Die Kundennummer hat sich geändert.<br>
1 id: 1001<br>
    Die Kundennummer hat sich geändert.<br>
2 id: 1002<br>
    Die Kundennummer hat sich geändert.<br>
```

index_next

'index_next' wird verwendet um den nächsten 'loop'-Index auszugeben. Bei der letzten Iteration ist dieser Wert um 1 grösser als der aktuelle 'loop'-Index (inklusive dem definierten 'step' Wert).

Beispiel 7.26. section'-Eigenschaft 'index_next'

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* zur Information, $custid[customer.index] und $custid[customer] bedeuten das selbe *}
{if $custid[customer.index_next] ne $custid[customer.index]}
    Die Kundennummer wird sich ändern.<br>
{/if}
{/section}
```

Ausgabe des obigen Beispiels:

```
0 id: 1000<br>
    Die Kundennummer wird sich ändern.<br>
1 id: 1001<br>
    Die Kundennummer wird sich ändern.<br>
2 id: 1002<br>
    Die Kundennummer wird sich ändern.<br>
]]¦
   </programlisting>
  </example>
  </sect2>
  <sect2 id="section.property.iteration">
   <title>iteration</title>
   <para>
    'iteration' wird verwendet um die aktuelle Iteration auszugeben.
   </para>
   <para>
    Bemerkung: Die Eigenschaften 'start', 'step' und 'max'
    beeinflussen 'iteration' nicht, die Eigenschaft 'index' jedoch
    schon. 'iteration' startet im gegensatz zu 'index' bei 1. 'rownum'
    ist ein Alias für 'iteration' und arbeitet identisch.
   </para>
  <example>
   <title>'section'-Eigenschaft 'iteration'</title>
   <programlisting>
<![CDATA[
{section name=customer loop=$custid start=5 step=2}
aktuelle loop iteration: {$smarty.section.customer.iteration}<br>
{$smarty.section.customer.index} id: {$custid[customer]}<br>
```

```
{* zur Information, $custid[customer.index] und $custid[customer] bedeuten das gleiche *}
{if $custid[customer.index_next] ne $custid[customer.index]}
    Die Kundennummer wird sich ändern.<br>
{/if}
{/section}
```

Ausgabe des obigen Beispiels:

```
aktuelle loop iteration: 1
5 id: 1000<br>
    Die Kundennummer wird sich ändern.<br>
aktuelle loop iteration: 2
7 id: 1001<br>
    Die Kundennummer wird sich ändern.<br>
aktuelle loop iteration: 3
9 id: 1002<br>
    Die Kundennummer wird sich ändern.<br>
```

first

'first' ist 'true', wenn die aktuelle Iteration die erste dieser 'section' ist.

Beispiel 7.27. 'section'-Eigenschaft 'first'

```
{section name=customer loop=$custid}
{if $smarty.section.customer.first}
    <table>
{/if}

<tr><td>{$smarty.section.customer.index} id:
        {$custid[customer]}</td></tr>

{if $smarty.section.customer.last}
    </table>
{/if}
{/section}
```

Ausgabe des obigen Beispiels:

```
<table>
<tr><td>0 id: 1000</td></tr>
<tr><td>1 id: 1001</td></tr>
<tr><td>2 id: 1002</td></tr>
</table>
```

last

'last' ist 'true' wenn die aktuelle Iteration die letzte dieser 'section' ist.

Beispiel 7.28. 'section'-Eigenschaft 'last'

```
{section name=customer loop=$custid}
{if $smarty.section.customer.first}
    <table>
{/if}
```

```
<tr><td>{$smarty.section.customer.index} id:
        {$custid[customer]}</td></tr>

{if $smarty.section.customer.last}
    </table>
{/if}
{/section}
```

Ausgabe des obigen Beispiels:

```
<table>
<tr><td>0 id: 1000</td></tr>
<tr><td>1 id: 1001</td></tr>
<tr><td>2 id: 1002</td></tr>
</table>
```

rownum

'rownum' wird verwendet um die aktuelle Iteration (startend bei 1) auszugeben. 'rownum' ist ein Alias für 'iteration' und arbeitet identisch.

Beispiel 7.29. 'section'-Eigenschaft 'rownum'

```
{section name=customer loop=$custid}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br />
{/section}
   </programlisting>
   <para>
    Ausgabe des obigen Beispiels:
   </para>
   <programlisting>
<![CDATA[
1 id: 1000<br />
2 id: 1001<br />
3 id: 1002<br />
```

loop

'loop' wird verwendet, um die Nummer letzte Iteration der 'section' auszugeben. Dieser Wert kann inner- und ausserhalb der 'section' verwendet werden.

Beispiel 7.30. 'section'-Eigenschaft 'loop'

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}

Es wurden {$smarty.section.customer.loop} Kunden angezeigt.
```

Ausgabe des obigen Beispiels:

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
```

```
Es wurden 3 Kunden angezeigt.
```

show

show kann die Werte 'true' oder 'false' haben. Falls der Wert 'true' ist, wird die 'section' angezeigt. Falls der Wert 'false' ist, wird die 'section' - ausser dem 'sectionelse' - nicht ausgegeben.

Beispiel 7.31. 'section'-Eigenschaft 'show'

```
{section name=customer loop=$custid show=$show_customer_info}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br />
{/section}

{if $smarty.section.customer.show}
die 'section' wurde angezeigt
{else}
die 'section' wurde nicht angezeigt
{/if}
```

Ausgabe des obigen Beispiels:

```
1 id: 1000<br />
2 id: 1001<br />
3 id: 1002<br />

die 'section' wurde angezeigt
```

total

Wird verwendet um die Anzahl der durchlaufenen Iterationen einer 'section' auszugeben. Kann innerhalb oder ausserhalb der 'section' verwendet werden.

Beispiel 7.32. 'section'-Eigenschaft 'total'

```
{section name=customer loop=$custid step=2}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{/section}

Es wurden {$smarty.section.customer.total} Kunden angezeigt.
```

Ausgabe des obigen Beispiels:

```
0 id: 1000<br>
2 id: 1001<br>
4 id: 1002<br>

Es wurden 3 Kunden angezeigt.
```

strip

Webdesigner haben oft das Problem, dass Leerzeichen und Zeilenumbrüche die Ausgabe des erzeugten HTML im Browser

beeinflussen. Oft werden deshalb alle Tags aufeinanderfolgend im Template notiert, was aber zu einer schlechten Lesbarkeit führt.

Aus dem Inhalt zwischen den {strip}{/strip}-Tags werden alle Leerzeichen und Zeilenumbrüche entfernt. So können Sie Ihre Templates lesbar halten, ohne sich Sorgen um die Leerzeichen zu machen.

> **Technische Bemerkung:** {strip}{/strip} ändert nicht den Inhalt einer Template-Variablen. Dafür gibt es den strip Modifikator.

Beispiel 7.33. strip tags

```
{* der folgende Inhalt wird in einer Zeile ausgegeben *}
{strip}
<table border=0>
 <tr>
  <td>
   <a href="{$url}">
    <font color="red">Das ist ein Test.</font>
   </a>
  </td>
 </tr>
</table>
{/strip}
```

Ausgabe des obigen Beispiels:

```
<table border=0><tr><td><a href="http://my.domain.com"><font color="red">Das ist ein Test.</font></a></
```

Achtung: im obigen Beispiel beginnen und enden alle Zeilen mit HTML-Tags. Falls Sie Abschnitte haben, die nur Text enthalten, werden diese ebenfalls zusammengeschlossen. Das kann zu unerwünschten Resultaten führen.

Siehe auch strip-Modifikator (Zeichenkette strippen)

Kapitel 8. Eigene Funktionen

Inhaltsverzeichnis

Smarty wird mit verschiedenen massgeschneiderten Funktionen geliefert, welche Sie in Ihren Templates verwenden können.

{assign} (zuweisen)

{assign} wird verwendet um einer Template-Variable **innerhalb eines Templates** einen Wert zuzuweisen.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
var	string	Ja	*n/a*	Der Name der zuzuweisenden Variable.
value	string	Ja	*n/a*	Der zuzuweisende Wert.

Beispiel 8.1. {assign} (zuweisen)

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

Der Wert von $name ist {$name}.
```

Ausgabe des obiges Beispiels:

```
Der Wert von $name ist Bob.</programlisting>
```

Beispiel 8.2. Zugriff auf mit {assign} zugwiesene Variablen von PHP aus.

Um auf zugewiesene Variablen von php aus zuzugreifen nimmt man get_template_vars(). Die zugewiesenen variablen sind jedoch nur wärhend bzw. nach der Ausgabe des Template verfügbar.

```
{* index.tpl *}
{assign var="foo" value="Smarty"}
```

```
<?php

// Keine Ausgabe, das das Template noch nicht ausgegeben wurde:
echo $smarty->get_template_vars('foo');

// das Template in eine ungenutzte Variable ausgeben
$nix = $smarty->fetch('index.tpl');

// Gibt 'smarty' aus, da die {assign} anweisung im Template ausgeführt
// wurde
echo $smarty->get_template_vars('foo');

$smarty->assign('foo','Even smarter');

// Ausgabe 'Even smarter'
echo $smarty->get_template_vars('foo');

?>
```

Folgende Funktionen haben *optionale* assign-Attribute:

{capture}, {include}, {include_php}, {insert}, {counter}, {cycle}, {eval}, {fetch}, {math}, {textformat}

Siehe auch assign() und get_template_vars().

{counter} (Zähler)

{counter} wird verwendet um eine Zahlenreihe auszugeben. Sie können den Initialwert bestimmen, den Zählinterval, die Richtung in der gezählt werden soll und ob der Wert ausgegeben wird. Sie können mehrere Zähler gleichzeitig laufen lassen, in dem Sie ihnen einmalige Namen geben. Wenn Sie keinen Wert für 'name' übergeben, wird 'default' verwendet.

Wenn Sie das spezielle 'assign'-Attribut verwenden, wird die Ausgabe des Zählers dieser Template-Variable zugewiesen anstatt ausgegeben zu werden.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Nein	*default*	Der Name des Zählers.
start	number	Nein	*1*	Der Initialwert.
skip	number	Nein	*1*	Der Interval.
direction	string	Nein	*up*	Die Richtung (up/down).
print	boolean	Nein	*true*	Definiert ob der Wert ausgegeben werden soll.
assign	string	Nein	*n/a*	Die Template-Variable welcher der Wert zugewiesen werden soll.

Beispiel 8.3. {counter} (Zähler)

```
{* zähler initialisieren *}
{counter start=0 skip=2}<br />
{counter}<br />
{counter}<br />
{counter}<br />
```

AUSGABE:

```
0<br />
2<br />
4<br />
6<br />
```

{cycle} (Zyklus)

{cycle} wird verwendet um durch ein Set von Werten zu zirkulieren. Dies vereinfacht die Handhabung von zwei oder mehr Farben in einer Tabelle, oder um einen Array zu durchlaufen.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Nein	*default*	Der Name des Zyklus.
values	mixed	Ja	*N/A*	Die Werte durch die zirkuliert werden soll, entweder als Komma separierte Liste (siehe 'delimiter'-Attribut), oder als Array.
print	boolean	Nein	*true*	Definiert ob die Werte ausgegeben werden sollen oder nicht.
advance	boolean	Nein	*true*	Definiert ob der nächste Wert automatisch angesprungen werden soll.
delimiter	string	Nein	,	Das zu verwendende Trennzeichen.
assign	string	Nein	*n/a*	Der Name der Template-Variable welcher die Ausgabe zugewiesen werden soll.
reset	boolean	No	*false*	Der Zyklus wird auf den ersten Wert zurückgesetzt.

Sie können durch mehrere Sets gleichzeitig iterieren, indem Sie den Sets einmalige Namen geben.

Um den aktuellen Wert nicht auszugeben, kann das 'print' Attribut auf 'false' gesetzt werden. Dies könnte sinnvoll sein, wenn man einen einzelnen Wert überspringen möchte.

Das 'advance'-Attribut wird verwendet um einen Wert zu wiederholen. Wenn auf 'false' gesetzt, wird bei der nächsten Iteration der selbe Wert erneut ausgegeben.

Wenn sie das spezielle 'assign'-Attribut übergeben, wird die Ausgabe der {cycle}-Funktion in dieser Template-Variable abgelegt, anstatt ausgegeben zu werden.

Beispiel 8.4. {cycle} (Zyklus)

```
{section name=rows loop=$data}
<tr bgcolor="{cycle values="#eeeeee,#d0d0d0"}">
   <td>{$data[rows]}</td>
</tr>
{/section}
```

```
<tr bgcolor="#eeeeee">
   <td>1</td>
</tr>
<tr bgcolor="#d0d0d0">
   <td>2</td>
</tr>
<tr bgcolor="#eeeeee">
   <td>3</td>
</tr>
```

{debug}

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
output	string	Nein	*javascript*	Ausgabe-Typ, entweder HTML oder Javascript.

{debug} zeigt die 'debugging'-Konsole auf der Seite an. $debug hat darauf keinen Einfluss. Da die Ausgabe zur Laufzeit geschieht, können die Template-Namen hier nicht ausgegeben werden. Sie erhalten jedoch eine Liste aller mit assigned zugewiesenen Variablen und deren Werten.

Siehe auch Debugging Konsole

{eval} (auswerten)

{eval} wird verwendet um eine Variable als Template auszuwerten. Dies kann verwendet werden um Template-Tags/Variablen in einer Variable oder einer Konfigurationsdatei abzulegen.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
var	mixed	Ja	*n/a*	Variable oder Zeichenkette die ausgewertet werden soll.
assign	string	Nein	*n/a*	Die Template-Variable welcher die Ausgabe zugewiesen werden soll.

Wenn Sie das spezielle 'assign'-Attribut übergeben, wird die Ausgabe von 'eval' in dieser Template-Variable gespeichert und nicht ausgegeben.

Technische Bemerkung: Evaluierte Variablen werden gleich wie Template-Variablen verwendet und folgen den selben Maskierungs- und Sicherheits-Features.

Technische Bemerkung: Evaluierte Variablen werden bei jedem Aufruf neu ausgewertet. Die kompilierten Versionen werden dabei nicht abgelegt! Falls sie caching eingeschaltet haben, wird die Ausgabe jedoch mit dem Rest des Templates gecached.

Beispiel 8.5. eval (auswerten)

```
setup.conf
----------

emphstart = <b>
emphend = </b>
title = Willkommen auf {$company}'s home page!
ErrorCity = Bitte geben Sie einen {#emphstart#}Stadtnamen{#emphend#} ein.
ErrorState = Bitte geben Sie einen {#emphstart#}Provinznamen{#emphend#} ein.
```

index.tpl:

```
{config_load file="setup.conf"}

{eval var=$foo}
{eval var=#title#}
{eval var=#ErrorCity#}
{eval var=#ErrorState# assign="state_error"}
{$state_error}
```

Ausgabe des obigen Beispiels:

```
Dies ist der Inhalt von foo:

Willkommen auf Pub & Grill's home page!
Bitte geben Sie einen <b>Stadtnamen</b> ein.
Bitte geben Sie einen <b>Provinznamen</b> ein.
```

{fetch}

{fetch} wird verwendet um lokale oder via HTTP beziehungsweise FTP verfügbare Inhalte auszugeben. Wenn der Dateiname mit 'http://' anfängt, wird die angegebene Webseite geladen und angezeigt. Wenn der Dateiname mit 'ftp://' anfängt wird die Datei vom FTP-Server geladen und angezeigt. Für lokale Dateien muss der absolute Pfad, oder ein Pfad relativ zum ausgeführten Skript übergeben werden.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
file	string	Ja	*n/a*	Die Datei, FTP oder HTTP Seite die geliefert werden soll.
assign	string	Nein	*n/a*	Die Template-Variable welcher die Ausgabe zugewiesen werden soll.

Wenn Sie das spezielle 'assign'-Attribut übergeben, wird die Ausgabe der {fetch}-Funktion dieser Template-Variable zugewiesen, anstatt ausgegeben zu werden (seit Smarty 1.5.0).

Technische Bemerkung: HTTP-Redirects werden nicht unterstützt, stellen Sie sicher, dass die aufgerufene URL falls nötig durch ein '/'-Zeichen (slash) beendet wird.

Technische Bemerkung: Wenn Sicherheit eingeschaltet ist, und Dateien vom lokalen System geladen werden sollen, ist dies nur für Dateien erlaubt welche sich in einem definierten sicheren Verzeichnis befinden. ($secure_dir)

Beispiel 8.6. fetch

```
{* einbinden von javascript *}
{fetch file="/export/httpd/www.domain.com/docs/navbar.js"}

{* Wetter Informationen aus einer anderen Webseite bei uns anzeigen *}
{fetch file="http://www.myweather.com/68502/"}

{* News Datei via FTP auslesen *}
{fetch file="ftp://user:password@ftp.domain.com/path/to/currentheadlines.txt"}

{* die Ausgabe einer Template variable zuweisen *}
{fetch file="http://www.myweather.com/68502/" assign="weather"}
{if $weather ne ""}
    <b>{$weather}</b>
{/if}
```

Siehe auch {capture}, {eval} und fetch().

{html_checkboxes} (Ausgabe von HTML-Checkbox Tag)

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Nein	*checkbox*	Name der checkbox Liste
values	array	ja, ausser wenn das option Attribut verwendet wird	*n/a*	ein Array mit Werten für die checkboxes
output	array	ja, ausser wenn das option Attribut verwendet wird	*n/a*	ein Array mit Werten für checkbox Knöpfe
selected	string/array	No	*empty*	das/die ausgewählten checkbox Elemente
options	assoziatives array	Ja, ausser values/output wird verwendet	*n/a*	ein assoziatives Array mit Werten und Ausgaben
separator	string	No	*empty*	Zeichenkette die zwischen den checkbox Elementen eingefügt werden soll
labels	boolean	No	*true*	fügt der Ausgabe <label>-Tags hinzu

html_checkboxes ist eine Funktion die aus den übergebenen Daten html checkbox Elemente erstellt und kümmert sich darum welche Elemente ausgewählt sind. Erforderliche Attribute sind Wert/Ausgabe oder Options. Die Ausgabe ist XHTML kompatibel

Alle Parameter die nicht in der Liste erwähnt werden, werden ausgegeben.

Beispiel 8.7. {html_checkboxes}

```
<?php

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array('Joe Schmoe',
                                    'Jack Smith',
                                    'Jane Johnson',
                                    'Charlie Brown'));
$smarty->assign('customer_id', 1001);

?>
```

Wobei index.tpl wie folgt aussieht:

```
{html_checkboxes name="id" values=$cust_ids selected=$customer_id output=$cust_names separator="<br />"
```

Oder mit folgendem PHP-Code:

```
<?php

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_checkboxes', array(
            1000 => 'Joe Schmoe',
            1001 => 'Jack Smith',
            1002 => 'Jane Johnson',
            1003 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
?>
```

Wobei index.tpl wie folgt aussieht:

```
{html_checkboxes name="id" options=$cust_checkboxes selected=$customer_id separator="<br />"}
```

Das Ergebnis beider Listings:

```
<label><input type="checkbox" name="id[]" value="1000" />Joe Schmoe</label><br />
<label><input type="checkbox" name="id[]" value="1001" checked="checked" />Jack Smith</label><br />
<label><input type="checkbox" name="id[]" value="1002" />Jane Johnson</label><br />
<label><input type="checkbox" name="id[]" value="1003" />Charlie Brown</label><br />
```

html_image (Ausgabe von HTML-IMG Tag)

{html_image} ist eine eigene Funktion die ein HTML Tag für ein Bild erzeugt. Die Höhe und Breite der Ausgabe wird automatisch aus der Bilddatei berechnet wenn die Werte nicht übergeben werden.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
file	string	Ja	*n/a*	Name/Pfad zum Bild
height	string	Nein	*Normale Höhe des Bildes*	Höhe des Bildes
width	string	Nein	*Normale Breite des Bildes*	Breite des Bildes
basedir	string	Nein	*DOCUMENT_ROOT*	Basisverzeichnis für relative Pfadangaben
alt	string	Nein	*""*	Alternative Beschreibung des Bildes
href	string	Nein	*n/a*	Link für das Bild
path_prefix	string	Nein	*n/a*	Präfix für den Pfad zum Bild

basedir ist der Basispfad der für die Verlinkung verwendet werden soll. Wenn kein Wert übergeben wird, wird die Umgebungsvariable DOCUMENT_ROOT verwendet. Wenn Sicherheit eingeschaltet ist, muss das Bild in einem sicheren Verzeichnis liegen.

href ist das href Attribut für das Image-Tag. Wenn dieser Wert übergeben wird, wird um das Bild ein <a href="LINKVALUE"><a> Tag erzeugt.

path_prefix ist ein optionaler Präfix der dem Bildpfad vorangestellt wird. Die ist nützlich wenn zum Beispiel für den Bildpfad ein anderer Servername verwendet werden soll.

Alle weiteren Parameter werden als Name/Wert Paare (Attribute) im <img>-Tag ausgegeben.

> **Technische Bemerkung:** {html_image} greift auf das Dateisystem zu um Höhe und Breite zu errechnen. Wenn Sie caching nicht verwenden sollten Sie normalerweise auf diese Funktion aus performance Gründen verzichten.

Beispiel 8.8. html_image

```
Wobei index.tpl wie folgt aussieht:
-----------------------------------
{html_image file="pumpkin.jpg"}
{html_image file="/path/from/docroot/pumpkin.jpg"}
{html_image file="../path/relative/to/currdir/pumpkin.jpg"}
```

Mögliche Ausgabe:

```
<img src="pumpkin.jpg" alt="" width="44" height="68" />
<img src="/path/from/docroot/pumpkin.jpg" alt="" width="44" height="68" />
<img src="../path/relative/to/currdir/pumpkin.jpg" alt="" width="44" height="68" />
```

html_options (Ausgabe von HTML-Options)

{html_options} wird verwendet um HTML-Options Listen mit den übergebenen Daten zu erzeugen. Die Funktion kümmert sich ebenfalls um das setzen des ausgewählten Standardwertes. Die Attribute 'values' und 'output' sind erforderlich, ausser man verwendet das Attribut 'options'.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
values	array	Ja, ausser 'options'-Attribut wird verwendet.	*n/a*	Array mit Werten für die dropdown-Liste.
output	array	Ja, ausser 'options'-Attribut wird verwendet.	*n/a*	Arrays mit Namen für die dropdown-Liste.
selected	string	Nein	*empty*	Das ausgewählte Array Element.
options	associative array	Ja, ausser wenn das 'values'- und das 'output'-Attribut verwendet werden.	*n/a*	Assoziatives Array mit Werten die ausgegeben werden sollen.

Wenn ein Wert als Array erkannt wird, wird er als HTML-OPTGROUP ausgegeben und die Werte werden in Gruppen dargestellt. Rekursion wird unterstützt. Die Ausgabe ist XHTML kompatibel.

Wenn das (optionale) Attribute *name* angegeben wurde, wird um die <option>-Liste von <select name="groupname"></select>-Tags umschlossen

Alle Parameter die deren Namen nicht in der obigen Liste genannt wurde, werden dem <select>-Tag als Name/Wert-Paare hinzugefügt. Die Parameter werden ignoriert, wenn kein *name*-Attribute angegeben wurde.

Beispiel 8.9. html_options

Beispiel 1:

```
<?php

$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array(
                              'Joe Schmoe',
                              'Jack Smith',
                              'Jane Johnson',
                              'Charlie Brown'));
$smarty->assign('customer_id', 1001);

?>
```

Wobei das Template wie folgt aussieht:

```
<select name="customer_id">
   {html_options values=$cust_ids output=$cust_names selected=$customer_id}
</select>
```

Beispiel 2:

```
<?php

$smarty->assign('cust_options', array(
                                1000 => 'Joe Schmoe',
                                1001 => 'Jack Smith',
                                1002 => 'Jane Johnson',
                                1003 => 'Charlie Brown')
                                );
$smarty->assign('customer_id', 1001);
```

```
?>
```

Wobei das Template wie folgt aussieht:

```
{html_options name=customer_id options=$cust_options selected=$customer_id}
```

Beide Beispiele ergeben folgende Ausgabe:

```
<select name="customer_id" size="4">
    <option label="Joe Schmoe" value="1000">Joe Schmoe</option>
    <option label="Jack Smith" value="1001" selected="selected">Jack Smith</option>
    <option label="Jane Johnson" value="1002">Jane Johnson</option>
    <option label="Charlie Brown" value="1003">Charlie Brown</option>
</select>
```

Siehe auch {html_checkboxes} und {html_radios}

Beispiel 8.10. {html_options} - Beispiel mit Datenbank (z.B. PEAR oder ADODB):

```
<?php

$sql = 'select type_id, types from types order by type';
$smarty->assign('types',$db->getAssoc($sql));

$sql = 'select contact_id, name, email, type_id
        from contacts where contact_id='.$contact_id;
$smarty->assign('contact',$db->getRow($sql));

?>
```

Wobei das Template wie folgt aussieht:

```
<select name="type_id">
    <option value='null'>-- none --</option>
    {html_options name="type" options=$types selected=$contact.type_id}
</select>
```

Siehe auch {html_checkboxes} und {html_radios}

html_radios (Ausgabe von HTML-RADIO Tags)

html_radio ist eine Funktion die aus den Ã¼bergebenen Daten html radio Elemente erstellt und kÃ¼mmert sich darum welche Elemente ausgewÃ¤hlt sind. Erforderliche Attribute sind Wert/Ausgabe oder Options. Die Ausgabe ist XHTML kompatibel

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
name	string	Nein	*radio*	Name der Radio Liste
values	array	Ja, ausser 'options'-Attribut wird verwendet.	*n/a*	Array mit Werten fÃ¼r die dropdown-Liste.
output	array	Ja, ausser 'options'-Attribut wird verwendet.	*n/a*	Arrays mit Namen fÃ¼r die dropdown-Liste.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
selected	string	Nein	*empty*	Das ausgewÃ¤hlte Array Element.
options	associative array	Ja, ausser wenn das 'values'- und das 'output'-Attribut verwendet werden.	*n/a*	Assoziatives Array mit Werten die ausgegeben werden sollen.
separator	string	No	*empty*	Die Zeichenkette die zwischen 2 Radioelemente eingefÃ¼gt werden soll.

Alle weiteren Parameter werden als Name/Wert Paare (Attribute) in jedem der <input>-Tags ausgegeben.

Beispiel 8.11. html_radios

```
<?php

$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array(
                                    'Joe Schmoe',
                                    'Jack Smith',
                                    'Jane Johnson',
                                    'Carlie Brown')
                );
$smarty->assign('customer_id', 1001);
?>
```

Mit folgendem index.tpl:

```
{html_radios values=$cust_ids checked=$customer_id output=$cust_names separator="<br />"}
```

Beispiel 8.12. {html_radios} : Example 2

```
<?php

$smarty->assign('cust_radios', array(
                                1000 => 'Joe Schmoe',
                                1001 => 'Jack Smith',
                                1002 => 'Jane Johnson',
                                1003 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);

?>
```

Mit folgendem index.tpl:

```
{html_radios name="id" options=$cust_radios selected=$customer_id separator="<br />"}
```

Ausgabe beider Beispiele:

```
<label for="id_1000">
<input type="radio" name="id" value="1000" id="id_1000" />Joe Schmoe</label><br />
<label for="id_1001"><input type="radio" name="id" value="1001" id="id_1001" checked="checked" />Jack S
```

```
<label for="id_1002"><input type="radio" name="id" value="1002" id="id_1002" />Jane Johnson</label><br
<label for="id_1003"><input type="radio" name="id" value="1003" id="id_1003" />Charlie Brown</label><br
```

Beispiel 8.13. {html_radios}-Datenbankbeispiel (z.B. mit PEAR oder ADODB):

```
<?php

$sql = 'select type_id, types from types order by type';
$smarty->assign('types',$db->getAssoc($sql));

$sql = 'select contact_id, name, email, type_id
        from contacts where contact_id='.$contact_id;
$smarty->assign('contact',$db->getRow($sql));

?>
```

Mit folgendem index.tpl:

```
{html_radios name="type" options=$types selected=$contact.type_id separator="<br />"}
```

Siehe auch {html_checkboxes} und {html_options}

html_select_date (Ausgabe von Daten als HTML-'options')

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
prefix	string	Nein	Date_	Prefix für die Namen.
time	timestamp/ YYYY-MM-DD	Nein	Aktuelle Zeit als Unix-Timestamp, oder in YYYY-MM-DD format.	Das zu verwendende Datum.
start_year	string	Nein	aktuelles Jahr	Das erste Jahr in der dropdown-Liste, entweder als Jahreszahl oder relativ zum aktuellen Jahr (+/- N).
end_year	string	Nein	Gegenteil von start_year	Das letzte Jahr in der dropdown-Liste, entweder als Jahreszahl oder relativ zum aktuellen Jahr (+/- N).
display_days	boolean	Nein	true	Definiert ob Tage ausgegeben sollen oder nicht.
display_months	boolean	Nein	true	Definiert ob Monate ausgegeben werden sollen oder nicht.
display_years	boolean	Nein	true	Definiert ob Jahre ausgegeben werden sollen oder nicht.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
month_format	string	Nein	%B	Format in welchem der Monat ausgegeben werden soll. (strftime)
day_format	string	Nein	%02d	Definiert das Format in welchem der Tag ausgegeben werden soll. (sprintf)
year_as_text	boolean	Nein	false	Definiert ob das Jahr als Text ausgegeben werden soll oder nicht.
reverse_years	boolean	Nein	false	Definiert ob die Daten in verkehrter Reihenfolge ausgegeben werden sollen.
field_array	string	Nein	null	Wenn ein Namen übergeben wird, werden die Daten in der Form name[Day], name[Year], name[Month] an PHP zurückgegeben.
day_size	string	Nein	null	Fügt dem 'select'-Tag das Attribut 'size' hinzu.
month_size	string	Nein	null	Fügt dem 'select'-Tag das Attribut 'size' hinzu.
year_size	string	Nein	null	Fügt dem 'select'-Tag das Attribut 'size' hinzu.
all_extra	string	Nein	null	Fügt allen 'select'-Tags zusätzliche Attribute hinzu.
day_extra	string	Nein	null	Fügt 'select'-Tags zusätzliche Attribute hinzu.
month_extra	string	Nein	null	Fügt 'select'-Tags zusätzliche Attribute hinzu.
year_extra	string	Nein	null	Fügt 'select'-Tags zusätzliche Attribute hinzu.
field_order	string	Nein	MDY	Die Reihenfolge in der die Felder ausgegeben werden.
field_separator	string	Nein	\n	Zeichenkette die zwischen den Feldern ausgegeben werden soll.
month_value_format	string	Nein	%m	Format zur Ausgabe der Monats-Werte,

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				Standardwert ist %m. (strftime)
year_empty	string	Nein	null	Definiert, einen Namen für das erste Element der Jahres Select-Box und dessen Wert "". Dies is hilfreich, wenn Sie eine Select-Box machen wollen, die die Zeichenkette "Bitte wählen Sie ein Jahr" als erstes Element enthält. Beachten Sie, dass Sie Werte wie "-MM-DD" als 'time' Attribut definieren können, um ein unselektiertes Jahr anzuzeigen.
month_empty	string	Nein	null	Definiert, einen Namen für das erste Element der Monats Select-Box und dessen Wert "". Dies is hilfreich, wenn Sie eine Select-Box machen wollen, die die Zeichenkette "Bitte wählen Sie einen Monat" als erstes Element enthält. Beachten Sie, dass Sie Werte wie "YYYY--DD" als 'time' Attribut definieren können, um einen unselektierten Monat anzuzeigen.
day_empty	string	No	null	Definiert, einen Namen für das erste Element der Tages Select-Box und dessen Wert "". Dies is hilfreich, wenn Sie eine Select-Box machen wollen, die die Zeichenkette "Bitte wählen Sie einen Tag" als erstes Element enthält. Beachten Sie, dass Sie Werte wie "YYYY-MM-" als 'time' Attribut definieren können, um einen unselektierten Tag anzuzeigen.

'html_select_date' wird verwendet um Datums-Dropdown-Listen zu erzeugen, und kann einen oder alle der folgenden Werte

darstellen: Jahr, Monat und Tag

Beispiel 8.14. html_select_date

```
{html_select_date}


AUSGABE:

<select name="Date_Month">
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12" selected>December</option>
</select>
<select name="Date_Day">
<option value="1">01</option>
<option value="2">02</option>
<option value="3">03</option>
<option value="4">04</option>
<option value="5">05</option>
<option value="6">06</option>
<option value="7">07</option>
<option value="8">08</option>
<option value="9">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13" selected>13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
</select>
<select name="Date_Year">
<option value="2001" selected>2001</option>
</select>
```

Beispiel 8.15. html_select_date

```
{* Start- und End-Jahr können relativ zum aktuellen Jahr definiert werden. *}
{html_select_date prefix="StartDate" time=$time start_year="-5" end_year="+1" display_days=false}
```

```
AUSGABE: (aktuelles Jahr ist 2000)

<select name="StartDateMonth">
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12" selected>December</option>
</select>
<select name="StartDateYear">
<option value="1999">1995</option>
<option value="1999">1996</option>
<option value="1999">1997</option>
<option value="1999">1998</option>
<option value="1999">1999</option>
<option value="2000" selected>2000</option>
<option value="2001">2001</option>
</select>
```

html_select_time (Ausgabe von Zeiten als HTML-'options')

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
prefix	string	Nein	Time_	Prefix des Namens.
time	timestamp	Nein	Aktuelle Uhrzeit.	Definiert die zu verwendende Uhrzeit.
display_hours	boolean	Nein	true	Definiert ob Stunden ausgegeben werden sollen.
display_minutes	boolean	Nein	true	Definiert ob Minuten ausgegeben werden sollen.
display_seconds	boolean	Nein	true	Definiert ob Sekunden ausgegeben werden sollen.
display_meridian	boolean	Nein	true	Definiert ob der Meridian (am/pm) ausgegeben werden soll.
use_24_hours	boolean	Nein	true	Definiert ob die Stunden in 24-Stunden Format angezeigt werden sollen oder nicht.
minute_interval	integer	Nein	1	Definiert den Interval in der Minuten-Dropdown-Liste.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
second_interval	integer	Nein	1	Definiert den Interval in der Sekunden-Dropdown-Liste.
field_array	string	Nein	n/a	Gibt die Daten in einen Array dieses Namens aus.
all_extra	string	Nein	null	Fügt allen 'select'-Tags zusätzliche Attribute hinzu.
hour_extra	string	Nein	null	Fügt dem Stunden-'select'-Tag zusätzliche Attribute hinzu.
minute_extra	string	Nein	null	Fügt dem Minuten-'select'-Tag zusätzliche Attribute hinzu.
second_extra	string	Nein	null	Fügt dem Sekunden-'select'-Tag zusätzliche Attribute hinzu.
meridian_extra	string	No	null	Fügt dem Meridian-'select'-Tag zusätzliche Attribute hinzu.

'html_select_time' wird verwendet um Zeit-Dropdown-Listen zu erzeugen. Die Funktion kann alle oder eines der folgenden Felder ausgeben: Stunde, Minute, Sekunde und Meridian.

Beispiel 8.16. html_select_time

```
{html_select_time use_24_hours=true}
```

Ausgabe:

```
<select name="Time_Hour">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09" selected>09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
</select>
```

```
<select name="Time_Minute">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20" selected>20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Second">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
```

```
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23" selected>23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Meridian">
<option value="am" selected>AM</option>
<option value="pm">PM</option>
</select>
```

html_table (Ausgabe von HTML-TABLE Tag)

Attribut Name	Typ	Erforderlich	Standartwert	Beschreibung
loop	array	Ja	*n/a*	Array mit den Daten für den Loop
cols	integer	Nein	*3*	Anzahl Spalten in einer Tabelle

Attribut Name	Typ	Erforderlich	Standartwert	Beschreibung
table_attr	string	No	*border="1"*	Attribute für das Table-Tag
tr_attr	string	No	*empty*	Attribute für das tr-Tag (Arrays werden durchlaufen)
td_attr	string	No	*empty*	Attribute für das tr-Tag (Arrays werden durchlaufen)
trailpad	string	No	* *	Wert für leere Zellen
hdir	string	No	*right*	Richtung in der die Zeilen gerendered werden. Mögliche Werte: *left*/*right*
vdir	string	No	*down*	Richtung in der die Spalten gerendered werden. Mögliche Werte: *up*/*down*

html_table ist eine eigene Funktion die einen Array als Tabelle ausgibt. Das *cols* Attribut definiert die Menge von Spalten die ausgegeben werden sollen. *table_attr*, *tr_attr* und *td_attr* definieren die Attribute für die HTML Tags. Wenn *tr_attr* oder *td_attr* Arrays sind, werden diese durchlaufen. *trailpad* wird in leere Zellen eingefügt.

Beispiel 8.17. html_table

```
index.php:

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('data',array(1,2,3,4,5,6,7,8,9));
$smarty->assign('tr',array('bgcolor="#eeeeee"','bgcolor="#dddddd"'));
$smarty->display('index.tpl');

index.tpl:

{html_table loop=$data}
{html_table loop=$data cols=4 table_attr='border="0"'}
{html_table loop=$data cols=4 tr_attr=$tr}

AUSGABE:

<table border="1">
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td><td>6</td></tr>
<tr><td>7</td><td>8</td><td>9</td></tr>
</table>
<table border="0">
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>9</td><td> </td><td> </td><td> </td></tr>
</table>
<table border="1">
<tr bgcolor="#eeeeee"><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr bgcolor="#dddddd"><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr bgcolor="#eeeeee"><td>9</td><td> </td><td> </td><td> </td></tr>
</table>
```

mailto

Attribut Name	Typ	Benötigt	Standard	Beschreibung
Adresse	string	Ja	*n/a*	Die EMail Adresse
Text	string	Nein	*n/a*	Der Text der angezeigt werden soll. Standardwert ist die EMail Adresse
encode	string	Nein	*none*	Wie die EMail Adresse verschlüsselt werden soll. Erlaubt sind 'none', 'hex' und 'javascript'.
CC	string	Nein	*n/a*	Komma separierte Liste der EMail Adressen, die eine Kopie der Nachricht erhalten sollen.
BCC	string	Nein	*n/a*	Komma separierte Liste der EMail Adressen, die eine blinde Kopie der Nachricht erhalten sollen.
Titel	string	Nein	*n/a*	Titel der Nachricht.
Newsgroups	string	Nein	*n/a*	Komma separierte Liste der Newsgroups, die eine Kopie der Nachricht erhalten sollen.
FollowupTo	string	Nein	*n/a*	Komma separierte Liste der Followup Adressen.
Extra	string	Nein	*n/a*	Zusätzliche Attribute, die sie dem Link geben wollen.

mailto vereinfach den Einsatz von mailto-Links und verschlüsselt die Links. Verschlüsselte Links können von WebSpiders schlechter ausgelesen werden.

> **Technische Bemerkung:** Javascript ist wahrscheinlich die beste Methode, die Daten für WebSpider unzugänglich zu machen.

Beispiel 8.18. mailto

```
{mailto address="me@domain.com"}
{mailto address="me@domain.com" text="Der angezeigte Linktext"}
{mailto address="me@domain.com" encode="javascript"}
{mailto address="me@domain.com" encode="hex"}
{mailto address="me@domain.com" subject="Hallo!"}
{mailto address="me@domain.com" cc="you@domain.com,they@domain.com"}
{mailto address="me@domain.com" extra='class="email"'}
```

```
OUTPUT:

<a href="mailto:me@domain.com" >me@domain.com</a>
<a href="mailto:me@domain.com" >Der angezeigte Linktext</a>
<script type="text/javascript" language="javascript">eval(unescape('%64%6f%63%75%6d%65%6e%74%2e%77%72%6
9%74%65%28%27%3c%61%20%68%72%65%66%3d%22%6d%61%69%6c%74%6f%3a%6d%65%40%64%6f%6d%
61%69%6e%2e%63%6f%6d%22%20%3e%6d%65%40%64%6f%6d%61%69%6e%2e%63%6f%6d%3c%2f%61%3e
%27%29%3b'))</script>
<a href="mailto:%6d%65@%64%6f%6d%61%69%6e.%63%6f%6d" >me@domain.com</a>
<a href="mailto:me@domain.com?subject=Hallo%21" >me@domain.com</a>
<a href="mailto:me@domain.com?cc=you@domain.com%2Cthey@domain.com" >me@domain.com</a>
<a href="mailto:me@domain.com" class="email">me@domain.com</a>
```

math (Mathematik)

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
equation	string	Ja	*n/a*	Der auszuführende Vergleich.
format	string	Nein	*n/a*	Format der Ausgabe. (sprintf)
var	numeric	Ja	*n/a*	Wert der Vergleichsvariable.
assign	string	Nein	*n/a*	Template-Variable welcher die Ausgabe zugewiesen werden soll.
[var ...]	numeric	Yes	*n/a*	Zusätzliche Werte.

'math' ermöglicht es dem Designer, mathematische Gleichungen durchzuführen. Alle numerischen Template-Variablen können dazu verwendet werden und die Ausgabe wird an die Stelle des Tags geschrieben. Die Variablen werden der Funktion als Parameter übergeben, dabei kann es sich um statische oder um Template-Variablen handeln. Erlaubte Operatoren umfassen: +, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans und tan. Konsultieren Sie die PHP-Dokumentation für zusätzliche Informationen zu dieser Funktion.

Falls Sie die spezielle 'assign' Variable übergeben, wird die Ausgabe der 'math'-Funktion der Template-Variablen mit dem selben Namen zugewiesen anstatt ausgegeben zu werden.

> **Technische Bemerkung:** Die 'math'-Funktion ist wegen ihres Gebrauchs der 'eval()'-Funktion äusserst Ressourcen intensiv. Mathematik direkt im PHP-Skript zu verwenden ist wesentlich performanter. Sie sollten daher - wann immer möglich - auf die Verwendung verzichten. Stellen Sie jedoch auf jeden Fall sicher, dass Sie keine 'math'-Tags in 'sections' oder anderen 'loop'-Konstrukten verwenden.

Beispiel 8.19. math (Mathematik)

```
{* $height=4, $width=5 *}

{math equation="x + y" x=$height y=$width}

AUSGABE:

9


{* $row_height = 10, $row_width = 20, #col_div# = 2, aus Template zugewiesen *}
```

```
{math equation="height * width / division"
      height=$row_height
      width=$row_width
      division=#col_div#}

AUSGABE:

100


{* Sie können auch Klammern verwenden *}

{math equation="(( x + y ) / z )" x=2 y=10 z=2}

AUSGABE:

6


{* Sie können als Ausgabeformat alle von sprintf unterstötzen Definitionen verwenden *}

{math equation="x + y" x=4.4444 y=5.0000 format="%.2f"}

AUSGABE:

9.44
```

popup (Popup-Inhalt definieren)

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
text	string	Ja	*n/a*	Text/HTML der im Popup ausgegeben werden soll.
trigger	string	Nein	*onMouseOver*	Definiert bei welchem Event das Popup aufgerufen werden soll. Erlaubte Werte sind: onMouseOver und onClick
sticky	boolean	Nein	*false*	Definiert ob das Popup geöffnet bleiben soll bis es manuell geschlossen wird.
caption	string	Nein	*n/a*	Definiert die Überschrift.
fgcolor	string	Nein	*n/a*	Hintergrundfarbe des Popups.
bgcolor	string	Nein	*n/a*	Rahmenfarbe des Popups.
textcolor	string	Nein	*n/a*	Farbe des Textes im Popup.
capcolor	string	Nein	*n/a*	Farbe der Popup-Überschrift.
closecolor	string	Nein	*n/a*	Die Farbe des 'close'-Textes.

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
textfont	string	Nein	*n/a*	Die Farbe des Textes.
captionfont	string	Nein	*n/a*	Die Schriftart für die Überschrift.
closefont	string	Nein	*n/a*	Die Schriftart für den 'close'-Text.
textsize	string	Nein	*n/a*	Die Schriftgrösse des Textes.
captionsize	string	Nein	*n/a*	Die Schriftgrösse der Überschrift.
closesize	string	Nein	*n/a*	Die Schriftgrösse des 'close'-Textes.
width	integer	Nein	*n/a*	Die Breite der Popup-Box.
height	integer	Nein	*n/a*	Die Höhe der Popup-Box.
left	boolean	Nein	*false*	Öffnet die Popup-Box links von Mauszeiger.
right	boolean	Nein	*false*	Öffnet die Popup-Box rechts von Mauszeiger.
center	boolean	Nein	*false*	Öffnet die Popup-Box in der Mitte des Mauszeigers.
above	boolean	Nein	*false*	Öffnet die Popup-Box oberhalb des Mauszeigers. Achtung: nur möglich wenn 'height' definiert ist.
below	boolean	Nein	*false*	Öffnet die Popup-Box unterhalb des Mauszeigers.
border	integer	Nein	*n/a*	Die Rahmenbreite der Popup-Box.
offsetx	integer	Nein	*n/a*	Horizontale Distanz zum Mauszeiger bei der das Popup geöffnet bleibt.
offsety	integer	Nein	*n/a*	Vertikale Distanz zum Mauszeiger bei der das Popup geöffnet bleibt.
fgbackground	url to image	Nein	*n/a*	Das Hintergundbild.
bgbackground	url to image	Nein	*n/a*	Definiert das Bild welches verwendet werden soll um den Rahmen zu zeichnen. Achtung: Sie müssen 'bgcolor' auf '' setzen, da die Farbe sonst angezeigt wird. Achtung: Wenn sie einen 'close'-Link

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				verwenden, wird Netscape (4.x) die Zellen mehrfach rendern, was zu einer falschen Anzeige führen kann.
closetext	string	Nein	*n/a*	Definiert den Text des 'close'-Links.
noclose	boolean	Nein	*n/a*	Zeigt den 'close'-Link nicht an.
status	string	Nein	*n/a*	Definiert den Text der in der Browser-Statuszeile ausgegeben wird.
autostatus	boolean	Nein	*n/a*	Gibt als Statusinformationen den Popup-Text aus. Achtung: Dies überschreibt die definierten Statuswerte.
autostatuscap	string	Nein	*n/a*	Zeigt in der Statusleiste den Wert der Popup-Überschrift an. Achtung: Dies überschreibt die definierten Statuswerte.
inarray	integer	Nein	*n/a*	Weist 'overLib' an, den Wert aus dem in 'overlib.js' definierten Array 'ol_text' zu lesen.
caparray	integer	Nein	*n/a*	Weist 'overLib' an, die Überschrift aus dem in 'overlib.js' definierten Array 'ol_caps' zu lesen.
capicon	url	Nein	*n/a*	Zeigt das übergebene Bild vor der Überschrift an.
snapx	integer	Nein	*n/a*	Aliniert das Popup an einem horizontalen Gitter.
snapy	integer	Nein	*n/a*	Aliniert das Popup an einem vertikalen Gitter.
fixx	integer	Nein	*n/a*	Fixiert das Popup an der definierten horizontalen Position. Achtung: überschreibt alle anderen horizontalen Positionen.
fixy	integer	Nein	*n/a*	Fixiert das Popup an der definierten

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				vertikalen Position. Achtung: überschreibt alle anderen vertikalen Positionen.
background	url	Nein	*n/a*	Definiert das Hintergrundbild welches anstelle des Tabellenhintergrundes verwendet werden soll.
padx	integer,integer	Nein	*n/a*	Erzeugt horizontale Leerzeichen, um den Text platzieren zu können. Achtung: Dies ist eine 2-Parameter Funktion.
pady	integer,integer	Nein	*n/a*	Erzeugt vertikale Leerzeichen, um den Text platzieren zu können. Achtung: Dies ist eine 2-Parameter Funktion.
fullhtml	boolean	Nein	*n/a*	Lässt Sie den HTML-Code betreffend einem Hintergrundbild komplett kontrollieren.
frame	string	Nein	*n/a*	Kontrolliert Popups in einem anderen Frame. Sehen sie die 'overLib'-Seite für zusätzliche Informationen zu dieser Funktion.
timeout	string	Nein	*n/a*	Führt die übergebene Javascript-Funktion aus, und verwendet deren Ausgabe als Text für das Popup.
delay	integer	Nein	*n/a*	Macht, dass sich das Popup wie ein Tooltip verhält, und nach den definierten Millisekunden verschwindet.
hauto	boolean	Nein	*n/a*	Lässt 'overLib' automatisch definieren an welcher Seite (links/rechts) des Mauszeigers das Popup ausgegeben werden soll.
vauto	boolean	Nein	*n/a*	Lässt 'overLib' automatisch definieren an welcher Seite

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				(oben/unten) des Mauszeigers das Popup ausgegeben werden soll.

'popup' wird verwendet um Javascript-Popup-Fenster zu erzeugen.

Beispiel 8.20. popup

```
{* 'popup_init' muss am Anfang jeder Seite aufgerufen werden die 'popup' verwendet *}
{popup_init src="/javascripts/overlib.js"}

{* create a link with a popup window when you move your mouse over *}
{* ein link mit einem Popup welches geöffnet wird wenn die Maus über dem Link ist. *}
<A href="mypage.html" {popup text="This link takes you to my page!"}>mypage</A>


{* Sie können in einem Popup text, html, links und weiteres verwenden *}
<A href="mypage.html" {popup sticky=true caption="mypage contents"
text="<UL><LI>links<LI>pages<LI>images</UL>" snapx=10 snapy=10}>mypage</A>

AUSGABE:


(Für Beispiele können Sie sich die Smarty Homepage anschauen.)
```

popup_init (Popup Initialisieren)

'popup' ist eine Integration von 'overLib', einer Javascript Library für 'popup'-Fenster. Dies kann verwendet werden um Zusatzinformationen als Context-Menu oder Tooltip auszugeben. 'popup_init' muss am Anfang jedes Templates aufgerufen werden, falls Sie planen darin die popup-Funktion zu verwenden. Der Author von 'overLib' ist Erik Bosrup, und die Homepage ist unter http://www.bosrup.com/web/overlib/ erreichbar.

Seit Smarty 2.1.2 wird 'overLib' NICHT mehr mitgeliefert. Laden Sie 'overLib' herunter und platzieren Sie es in Ihrer Document Root. Danach können Sie mit dem Attribut 'src' definieren an welcher Stelle die Datei liegt.

Beispiel 8.21. popup_init

```
{* popup_init must be called once at the top of the page *}
{popup_init src="/javascripts/overlib.js"}
```

textformat (Textformatierung)

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
style	string	Nein	*n/a*	aktueller Stil
indent	number	Nein	*0*	Anzahl Zeichen die für das einrücken von Zeilen verwendet werden.
indent_first	number	Nein	*0*	Anzahl Zeichen die für

Attribut Name	Typ	Erforderlich	Standardwert	Beschreibung
				das Einrücken der ersten Zeile verwendet werden.
indent_char	string	Nein	*(single space)*	Das Zeichen welches zum Einrücken verwendet werden soll.
wrap	number	Nein	*80*	Maximale Zeilenlänge bevor die Zeile umgebrochen wird.
wrap_char	string	Nein	*\n*	Das für Zeilenumbrüche zu verwendende Zeichen.
wrap_cut	boolean	Nein	*false*	Wenn auf 'true' gesetzt, wird die Zeile an der definierten Position abgeschnitten.
assign	string	Nein	*n/a*	Die Template-Variable welcher die Ausgabe zugewiesen werden soll.

'textformat' ist eine Funktion um Text zu formatieren. Die Funktion entfernt überflüssige Leerzeichen und formatiert Paragrafen indem sie die Zeilen einrückt und umbricht.

Sie können entweder den aktuellen Stil verwenden, oder ihn anhand der Parameter selber definieren. Im Moment ist 'email' der einzig verfügbare Stil.

Beispiel 8.22. textformat (Text Formatierung)

```
{textformat wrap=40}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

AUSGABE:

This is foo. This is foo. This is foo.
This is foo. This is foo. This is foo.

This is bar.

bar foo bar foo foo. bar foo bar foo
```

```
foo. bar foo bar foo foo. bar foo bar
foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo.


{textformat wrap=40 indent=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

AUSGABE:

    This is foo. This is foo. This is
    foo. This is foo. This is foo. This
    is foo.

    This is bar.

    bar foo bar foo foo. bar foo bar foo
    foo. bar foo bar foo foo. bar foo
    bar foo foo. bar foo bar foo foo.
    bar foo bar foo foo. bar foo bar
    foo foo.

{textformat wrap=40 indent=4 indent_first=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

AUSGABE:

        This is foo. This is foo. This
    is foo. This is foo. This is foo.
    This is foo.

        This is bar.

        bar foo bar foo foo. bar foo bar
    foo foo. bar foo bar foo foo. bar
```

```
    foo bar foo foo. bar foo bar foo
    foo. bar foo bar foo foo. bar foo
    bar foo foo.

{textformat style="email"}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

AUSGABE:

This is foo. This is foo. This is foo. This is foo. This is foo. This is
foo.

This is bar.

bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo
foo.
```

Kapitel 9. Konfigurationsdateien

Konfigurationsdateien sind ein praktischer Weg um Template-Variablen aus einer gemeinsamen Datei zu lesen. Ein Beispiel sind die Template-Farben. Wenn Sie die Farben einer Applikation anpassen wollen, müssen Sie normalerweise alle Templates durcharbeiten, und die entsprechenden Werte ändern. Mit einer Konfigurationsdatei können Sie alle Definitionen in einer einzigen Datei vornehmen, und somit auch einfach ändern.

Beispiel 9.1. Beispiel der Konfigurationsdatei-Syntax

```
# global variables
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[Customer]
pageTitle = "Customer Info"

[Login]
pageTitle = "Login"
focus = "username"
Intro = """Diese Zeile erstreckt sich über
                mehrere Zeilen, und muss deswegen
                        mit dreifachen Anführungszeichen
                                umschlossen werden."""

# hidden section
[.Database]
host=my.example.com
db=ADDRESSBOOK
user=php-user
pass=foobar
```

Die Werte in einer Konfigurationsdatei können in einfachen/doppelten Anführungszeichen notiert werden. Falls Sie einen Wert haben der sich über mehrere Zeilen ausbreitet muss dieser Wert in dreifachen Anführungszeichen (""") eingebettet werden. Die Kommentar-Syntax kann frei gewählt werden, solange sie nicht der normalen Syntax entsprechen. Wir empfehlen die Verwendung von # (Raute) am Anfang jeder Kommentar-Zeile.

Dieses Beispiel hat 2 'sections'. 'section'-Namen werden von []-Zeichen umschlossen und können alle Zeichen ausser [und] enthalten. Die vier Variablen welche am Anfang der Datei definiert werden sind globale Variablen. Diese Variablen werden immer geladen. Wenn eine definierte 'section' geladen wird, werden also die globalen Variablen ebenfalls eingelesen. Wenn eine Variable sowohl global als auch in einer 'section' vorkommt, wird die 'section'-Variable verwendet. Wenn zwei Variablen in der gleichen 'section' den selben Namen aufweisen wird die Letztere verwendet, es sei denn $config_overwrite ist deaktiviert ('false').

Konfigurationsdateien werden mit **config_load** geladen.

Sie können Variablen oder auch ganze 'sections' verstecken indem Sie dem Namen ein '.' voranstellen. Dies ist besonders wertvoll wenn Ihre Applikation sensitive Informationen aus der Konfigurationsdatei liest welche von der Template-Engine nicht verwendet werden. Falls eine Drittpartei eine Änderung an der Konfigurationsdatei vornimmt können Sie so sicherstellen, dass die sensitiven Daten nicht in deren Template geladen werden können.

Siehe auch: {config_load}, $config_overwrite, get_config_vars(), clear_config() und config_load()

Kapitel 10. Debugging Konsole

Smarty wird mit einer eingebauten Debugging Konsole ausgeliefert. Diese Konsole informiert über die im aufgerufenen Template eingebundenen Templates, die zugewiesenen Variablen und die Konfigurations-Variablen. Die Formatierung der Konsole wird über das Template debug.tpl gesteuert. Um debugging zu aktivieren, setzten Sie $debugging auf 'true' und (falls nötig) übergeben in $debug_tpl den Pfad zum Debugtemplate (normalerweise SMARTY_DIRdebug.tpl). Wenn Sie danach eine Seite laden, sollte ein Javascript-Fenster geöffnet werden in welchem Sie alle Informationen zur aufgerufenen Seite finden. Falls Sie die Variablen eines bestimmten Templates ausgeben wollen, können Sie dazu die Funktion {debug} verwenden. Um debugging auszuschalten, können Sie $debugging auf 'false' setzen. Sie können debugging auch temporär aktivieren, in dem Sie der aufgerufenen URL SMARTY_DEBUG mit übergeben, dies muss jedoch zuerst mit $debugging_ctrl aktiviert werden.

> **Technische Bemerkung:** Die Debugging Konsole funktioniert nicht für Daten die via fetch() geladen wurden, sondern nur für Daten die via display() ausgegeben werden. Die Konsole besteht aus ein paar Zeilen Javascript welche am Ende jeder Seite eingefügt werden. Wenn Sie Javascript nicht mögen, können Sie die Ausgabe in 'debug.tpl' selbst definieren. Debug-Ausgaben werden nicht gecached und Informationen zu 'debug.tpl' selbst werden nicht ausgegeben.
>
> **Anmerkung:** Die Ladezeiten werden in Sekunden, oder Bruchteilen davon, angegeben.

Teil III. Smarty für Programmierer

Inhaltsverzeichnis

Kapitel 11. Konstanten

Inhaltsverzeichnis

SMARTY_DIR

Definiert den **absoluten Systempfad** zu den Smarty Klassendateien. Falls der Wert nicht definiert ist, versucht Smarty ihn automatisch zu ermitteln. **Der Pfad muss mit einem '/'-Zeichen enden**.

Beispiel 11.1. SMARTY_DIR

```
<?php
// Pfad zum Smarty Verzeichnis setzen
define('SMARTY_DIR', '/usr/local/lib/php/Smarty/libs/');

// Pfad zum Smarty Verzeichnis setzen (unter Windows)
define('SMARTY_DIR', 'c:/usr/local/lib/php/Smarty/libs/');

// Smarty einbinden (der Dateiname beginnt mit großem 'S')
require_once(SMARTY_DIR . 'Smarty.class.php');
?>
```

Siehe auch $smarty.const und $php_handling constants

SMARTY_CORE_DIR

Dies ist der absolute Systempfad zu den Smarty Kerndateien. Wenn nicht vorher definiert, dann definiert Smarty diesen Wert mit *internals/* unterhalb des Verzeichniss SMARTY_DIR. Wenn angegeben, dann muss dieser Wert mit einem '/' enden.

Beispiel 11.2. SMARTY_CORE_DIR

```
<?php

// Laden von core.get_microtime.php
require_once(SMARTY_CORE_DIR . 'core.get_microtime.php');

?>
```

Siehe auch: $smarty.const

Kapitel 12. Smarty Klassenvariablen (Objekteigenschaften)

Inhaltsverzeichnis

$template_dir

Definiert das Standard-Template Verzeichnis. Wenn sie beim Einbinden von Templates keinen Ressourcen-Typ übergeben, werden sie in diesem Pfad gesucht. Normalerweise lautet er './templates'. Das heisst, Smarty erwartet das Template-Verzeichnis im selben Verzeichnis wie das ausgeführte PHP-Skript.

Technische Bemerkung: Dieses Verzeichnis sollte ausserhalb der DocumentRoot des Webservers liegen.

$compile_dir

Definiert das Verzeichnis, in das die kompilierten Templates geschrieben werden. Normalerweise lautet es './templates_c'. Das heisst, Smarty erwartet das Kompilier-Verzeichnis im selben Verzeichnis wie das ausgeführte PHP-Skript.

> **Technische Bemerkung:** Diese Einstellung kann als relativer oder als absoluter Pfad angegeben werden. 'include_path' wird nicht verwendet.
>
> **Technische Bemerkung:** Dieses Verzeichnis sollte ausserhalb der DocumentRoot des Webservers liegen.

$config_dir

Dieses Verzeichnis definiert den Ort, an dem die von den Templates verwendeten Konfigurationsdateien abgelegt sind. Normalerweise ist dies './configs'. Das bedeutet, Smarty erwartet das Konfigurations-Verzeichnis im selben Verzeichnis wie das ausgeführte PHP-Skript.

> **Technische Bemerkung:** Dieses Verzeichnis sollte ausserhalb der DocumentRoot des Webservers liegen.

$plugins_dir

Definiert das Verzeichnis in welchem Smarty die zu ladenden Plugins sucht. Normalerweise ist dies 'plugins' im SMARTY_DIR Pfad. Wenn Sie einen relativen Pfad angeben, wird Smarty zuerst versuchen das Plugin von SMARTY_DIR aus zu erreichen, danach relativ zum aktuellen Verzeichnis (mit 'cwd' - current working directory) und zum Schluss in jedem Eintrag des PHP-'include_path'.

> **Technische Bemerkung:** Für optimale Performance ist es sinnvoll, 'plugins_dir' absolut oder relativ zu SMARTY_DIR bzw. dem aktuellen Verzeichnis zu definieren. Von der Definition des Verzeichnisses im PHP-'include_path' wird abgeraten.

$debugging

Aktiviert die Debugging Konsole. Die Konsole besteht aus einem Javascript-Fenster, welches Informationen zum momentan geladenen Template und den zugewiesenen Variablen enthält.

$debug_tpl

Definiert den Namen des für die Debugging Konsole verwendeten Template. Normalerweise lautet er 'debug.tpl' und befindet sich im SMARTY_DIR Verzeichnis.

$debugging_ctrl

Definiert Alternativen zur Aktivierung der Debugging Konsole. NONE verbietet alternative Methoden. URL aktiviert ds Debugging, wenn das Schlüsselwort 'SMARTY_DEBUG' im QUERY_STRING gefunden wird. Wenn '$debugging' auf 'true' gesetzt ist, wird dieser Wert ignoriert.

$autoload_filters

Filter die Sie zu jedem Template laden möchten, können Sie mit Hilfe dieser Variable festlegen. Smarty wird sie danach automatisch laden. Die Variable enthält ein assoziatives Array, in dem der Schlüssel den Filter-Typ und der Wert den Filter-Namen definiert. Zum Beispiel:

```
<?php
$smarty->autoload_filters = array('pre' => array('trim', 'stamp'),
```

```
                                    'output' => array('convert'));
?>
```

$compile_check

Bei jedem Aufruf der PHP-Applikation überprüft Smarty, ob sich das zugrundeliegende Template seit dem letzten Aufruf geändert hat. Falls es eine Änderung feststellt, wird das Template neu kompiliert. Seit Smarty 1.4.0 wird das Template - falls es nicht existiert - kompiliert, unabhängig davon welcher Wert '$compile_check' hat. Normalerweise ist der Wert dieser Variable 'true'. Wenn eine Applikation produktiv eingesetzt wird (die Templates ändern sich nicht mehr), kann der 'compile_check'-Schritt entfallen. Setzen Sie dann '$compile_check' auf 'false', um die Performance zu steigern. Achtung: Wenn Sie '$compile_check' auf 'false' setzen und anschliessend ein Template ändern, wird diese Änderung *nicht* angezeigt. Wenn caching und 'compile_check' eingeschaltet sind, werden die gecachten Skripts neu kompiliert, sobald eine Änderung an einem der eingebundenen Templates festgestellt wird. Siehe auch $force_compile und clear_compiled_tpl.

$force_compile

Veranlasst Smarty dazu die Templates bei jedem Aufruf neu zu kompilieren. Diese Einstellung überschreibt '$compile_check'. Normalerweise ist dies ausgeschaltet, kann jedoch für die Fehlersuche nützlich sein. In einem Produktiven-Umfeld sollte auf die Verwendung verzichtet werden. Wenn caching eingeschaltet ist, werden die gecachten Dateien bei jedem Aufruf neu kompiliert.

$caching

Definiert ob Smarty die Template-Ausgabe cachen soll. Normalerweise ist dies ausgeschaltet (disabled, Wert: 0). Falls Ihre Templates redundante Inhalte erzeugen, ist es empfehlenswert caching einzuschalten. Die Performance wird signifikant verbessert. Sie können auch mehrere Caches für ein Template haben. Die Werte 1 und 2 aktivieren caching. Bei 1 verwendet Smarty die Variable '$cache_lifetime', um zu berechnen ob ein Template neu kompiliert werden soll. Der Wert 2 weist Smarty an, den Wert von 'cache_lifetime' zur Zeit der Erzeugung des Cache zu verwenden. Damit können Sie 'cache_lifetime' setzen, bevor Sie das Template einbinden und haben so eine feine Kontrolle darüber, wann ein bestimmter Cache abläuft. Konsultieren Sie dazu auch: is_cached.

Wenn '$compile_check' aktiviert ist, wird der Cache regeneriert sobald ein Template oder eine Konfigurations-Variable geändert wurde. Wenn '$force_compile' aktiviert ist, werden die gecachten Inhalte bei jedem Aufruf neu generiert.

$cache_dir

Definiert den Namen des Verzeichnisses in dem die Template-Caches angelegt werden. Normalerweise ist dies './cache', was Smarty veranlasst das Cache-Verzeichnis im aktuellen Verzeichnis zu suchen. Sie können auch einen eigenen Cache-Handler zur Kontrolle der Cache-Dateien definieren, der diese Einstellung ignoriert.

> **Technische Bemerkung:** Die Angabe muss entweder relativ oder absolut angegeben werden. 'include_path' wird nicht verwendet.
>
> **Technische Bemerkung:** Es wird empfohlen ein Verzeichnis ausserhalb der DocumentRoot zu verwenden.

$cache_lifetime

Definiert die Zeitspanne (in Sekunden) die ein Cache gültig bleibt. Ist die Zeit abgelaufen, wird der Cache neu generiert. '$caching' muss eingeschaltet (true) sein, damit '$cache_lifetime' Sinn macht. Der Wert -1 bewirkt, dass der Cache nie abläuft. Der Wert 0 bewirkt, dass der Inhalt immer neu generiert wird (nur sinnvoll für Tests, eine effizientere Methode wäre $caching auf 'false' zu setzen).

Wenn $force_compile gesetzt ist, wird der Cache immer neu generiert (was einem Ausschalten von caching gleichkommt).

Mit der clear_all_cache() Funktion können Sie alle Cache-Dateien auf einmal entfernen. Mit der clear_cache() Funktion können Sie einzelne Cache-Dateien (oder Gruppen) entfernen.

> **Technische Bemerkung:** Falls Sie bestimmten Templates eine eigene Cache-Lifetime geben wollen, können Sie dies tun indem Sie $caching auf 2 stellen und '$cache_lifetime' einen einmaligen Wert zuweisen, bevor Sie 'display()' oder 'fetch()' aufrufen.

$cache_handler_func

Sie können auch eine eigene Cache-Handler Funktion definieren. Siehe Abschnitt zur custom cache handler Funktion.

$cache_modified_check

Wenn auf 1 gesetzt, verwendet Smarty den If-Modified-Since Header des Clients. Falls sich der Timestamp der Cache-Datei seit dem letzten Besuch nicht geändert hat, wird der Header '304 Not Modified' anstatt des Inhalts ausgegeben. Dies funktioniert nur mit gecachten Inhalten die keine **insert** Tags enthalten.

$config_overwrite

Definiert ob Variablen in einer Konfiguration sich gegenseitig überschreiben können. Wenn false, werden die Variablen dem Array angehängt. Dies kann hilfreich sein, um Arrays in Konfigurationsdateien abzulegen. Standardwert: true

$config_booleanize

Wenn auf 'true' gesetzt, werden die Werte on/true/yes und off/false/no von Variablen aus Konfigurationsdateien automatisch auf true oder false gesetzt. Dies erlaubt eine einfachere Handhabung in Templates, da Sie somit {if #foobar#} ... {/if} benutzen können. Standardwert: true

$config_read_hidden

Definiert ob Abschnitte die mit einem '.' beginnen aus Konfigurationsdateien gelesen werden sollen. Normalerweise sollte man dies auf false belassen, da man so sensitive Daten in der Konfiguration laden kann, die nicht dem Template zugewiesen werden. Standardwert: false

$config_fix_newlines

Definiert ib MAC und DOS Zeilenumbrüche (\r und \r\n) in Konfigurationsdateien automatisch in \n umgewandelt werden sollen. Standardwert: true

$default_template_handler_func

Diese Funktion wird aufgerufen, wenn ein Template nicht aus der vorgegebenen Quelle geladen werden kann.

$php_handling

Definiert wie Smarty mit PHP-Code innerhalb von Templates umgehen soll. Es gibt 4 verschiedene Einstellungen. Normalerweise wird SMARTY_PHP_PASSTHRU verwendet. Achtung: '$php_handling' wirkt sich NICHT auf PHP-Code aus, der zwischen {php}{/php} Tags steht.

- SMARTY_PHP_PASSTHRU - Smarty gibt die Tags aus.

- SMARTY_PHP_QUOTE - Smarty maskiert die Tags als HTML-Entities.
- SMARTY_PHP_REMOVE - Smarty entfernt die Tags.
- SMARTY_PHP_ALLOW - Smarty führt den Code als PHP-Code aus.

ACHTUNG: Es wird davon abgeraten, PHP-Code in Templates einzubetten. Bitte verwenden Sie stattdessen custom functions oder Variablen-Modifikatoren.

$security

'$security' ein-/ausschalten. Normalerweise 'false' (ausgeschaltet). Die Sicherheitseinstellung ist wertvoll, wenn nicht vertrauenswürdigen Parteien Zugriff auf die Templates gegeben wird (zum Beispiel via FTP). Mit aktivierter '$security' kann verhindert werden, dass diese das System via Template-Engine kompromittieren. Die '$security' einzuschalten halt folgende Auswirkungen auf die Template-Language (ausser sie werden mit '$security_settings' überschrieben):

- Wenn '$php_handling' auf SMARTY_PHP_ALLOW geschaltet ist, wird der Wert auf SMARTY_PHP_PASSTHRU geändert.
- Ausser den in '$security_settings' definierten, sind keine Funktionen in IF-Statements aufrufbar.
- Templates können nur aus den im '$secure_dir'-Array definierten Verzeichnissen geladen werden.
- 'fetch()' kann nur verwendet werden um Dateien aus '$secure_dir' zu laden.
- {php}{/php}-Tags sind nicht erlaubt.
- Ausser den in '$security_settings' definierten, sind keine PHP-Funktionen direkt als Variablen-Modifikatoren aufrufbar.

$secure_dir

Definiert die als 'sicher' geltenden Verzeichnisse als Array. {include} und {fetch} verwenden diese Verzeichnisse, wenn '$security' eingeschaltet ist.

$security_settings

Wird verwendet um spezifische Sicherheits-Einstellungen zu ändern, wenn '$security' eingeschaltet ist.

- PHP_HANDLING - true/false. Wenn auf 'true' gesetzt wird '$php_handling' ignoriert.
- IF_FUNCS - Ist ein Array aller erlaubter Funktionen in IF-Statements.
- INCLUDE_ANY - true/false. Wenn 'true', kann jedes Template geladen werden, auch ausserhalb der '$secure_dir'-Liste.
- PHP_TAGS - true/false. Wenn 'true', sind keine {php}{/php}-Tags erlaubt.
- MODIFIER_FUNCS - Ist ein Array aller Funktionen die als Variablen-Modifikatoren verwendet werden dürfen.

$trusted_dir

'$trusted_dir' wird nur verwendet wenn die Sicherheit eingeschaltet ist. Der Wert ist ein Array aller Verzeichnisse, die als

vertrauenswürdig gelten. In diesen Verzeichnissen können PHP-Skripte, die man direkt aus einem Template mit {include_php} aufruft, abgelegt werden.

$left_delimiter

Das zu verwendende linke Trennzeichen der Template-Sprache. Normalerweise '{'.

$right_delimiter

Das zu verwendende rechte Trennzeichen der Template-Sprache. Normalerweise '}'.

$compiler_class

Definiert den Namen der Compiler-Klasse, die Smarty zum kompilieren der Templates verwenden soll. Normalerweise 'Smarty_Compiler'. Nur für fortgeschrittene Anwender.

$request_vars_order

Die Reihenfolge in welcher die Request-Variblen zugewiesen werden. Verhält sich wie 'variables_order' in der php.ini.

$request_use_auto_globals

Definiert ob Smarty php's $HTTP_*_VARS[] ($request_use_auto_globals=false welches der Standardwert ist) oder $_*[] ($request_use_auto_globals=true) verwenden soll. Dies betrifft Templates die {$smarty.request.*}, {$smarty.get.*}, etc... verwenden. Achtung: wenn $request_use_auto_globals auf TRUE gesetzt ist, hat variable.request.vars.order keine Auswirkungen, da php's Konfigurationswert `gpc_order` verwendet wird.

$error_reporting

Wenn dieser Wert nicht 0 ist, wird er benutzt um das error_reporting-Level von PHP zu bestimmen. Wenn debugging eingeschaltet ist, wird der Wert ignoriert und das Error-Level bleibt unverändert.

$compile_id

Persistenter 'compile-identifier'. Anstatt jedem Funktionsaufruf die selbe 'compile_id' zu übergeben, kann eine individuelle 'compile_id' gesetzt werden. Das ist z. B. sinnvoll, um in Kombination mit einem 'prefilter' verschiedene Sprach-Versionen eines Template kompilieren.

$use_sub_dirs

Wenn Sie Smarty in einer Umgebung einsetzen, die das Erstellen von Unterverzeichnissen nicht erlaubt, können Sie diesen Wert auf 'false' setzen. Unterverzeichnisse sind jedoch effizienter und sollten deshalb möglichst verwendet werden.

$default_modifiers

Definiert ein Array von Variablen-Modifikatoren, die auf jeder Variable anzuwenden sind. Wenn Sie zum Beispiel alle Variablen standardmässig HTML-Maskieren wollen, können Sie array('escape:"htmlall"'); verwenden. Um eine Variable von dieser Behandlung auszuschliessen, können Sie ihr den Parameter 'smarty' mit dem Modifikator 'nodefaults' übergeben. Als Beispiel: {$var|smarty:nodefaults}. Zum Beispiel: {$var|nodefaults}.

$default_resource_type

Definiert den Ressourcentyp der von Smarty implizitverwendet werden soll. Standartwert ist 'file', was dazu führt dass $smarty->display('index.tpl'); und $smarty->display('file:index.tpl'); identisch sind. Konsultieren Sie das Resource Kapitel für weitere Informationen.

Kapitel 13. Methoden der Klasse Smarty

Inhaltsverzeichnis

append (anhängen)

append (anhängen)

append (anhängen)

void **append** (mixed var)
void **append** (string varname, mixed var [, bool merge])

Wird verwendet, um an Template-Variablen weitere Daten anzuhängen. Sie können entweder ein Namen/Wert-Paar oder assoziative Arrays, die mehrere Namen/Wert-Paare enthalten, übergeben.

Beispiel 13.1. append (anhängen)

```
<?php
// Namen/Wert-Paare &uuml;bergeben
$smarty->append("Name", "Fred");
$smarty->append("Address", $address);

// assoziatives Array &uuml;bergeben
$smarty->append(array("city" => "Lincoln", "state" => "Nebraska"));
?>
```

append_by_ref (Referenz anhängen)

append_by_ref (Referenz anhängen)

append_by_ref (Referenz anhängen)

void **append_by_ref** (string varname, mixed var [, bool merge])

Wird verwendet, um an Template-Variablen Werte via Referenz (pass by reference) anstatt via Kopie anzuhängen. Konsultieren Sie das PHP-Manual zum Thema 'variable referencing' für weitere Erklärungen.

> **Technische Bemerkung:** 'append_by_ref()' ist effizienter als 'append()', da keine Kopie der Variable erzeugt, sondern auf die Variable im Speicher referenziert wird. Beachten Sie dabei, dass eine nachträgliche änderung Original-Variable auch die zugewiesene Variable ändert. PHP5 wird die Referenzierung automatisch übernehmen, diese Funktion dient als Workaround.
>
> **Technische Bemerkung:** Der *merge* Parameter berüksichtigt Array Keys. Das bedeutet, dass numerisch indizierte Arrays sich gegenseitig überschreiben können, oder die Keys nicht sequentiell ausgegeben werden. Dies, im Gegensatz zur PHP Funktion array_merge() [http://php.net/array_merge], die numerische Keys neu sortiert.

Beispiel 13.2. append_by_ref (via Referenz anhängen)

```
<?php
// Namen/Wert-Paare &uuml;bergeben
$smarty->append_by_ref("Name", $myname);
$smarty->append_by_ref("Address", $address);
?>
```

assign

assign

assign

void **assign** (mixed var)
void **assign** (string varname, mixed var)

Wird verwendet, um einem Template Werte zuzuweisen. Sie können entweder Namen/Wert-Paare oder ein assoziatives Array mit Namen/Wert-Paaren übergeben.

Beispiel 13.3. assign

```
<?php
// Namen/Wert-Paare &uuml;bergeben
$smarty->assign('Name', 'Fred');
$smarty->assign('Address', $address);

// assoziatives Array mit Namen/Wert-Paaren &uuml;bergeben
$smarty->assign(array("city" => "Lincoln", "state" => "Nebraska"));
?>
```

assign_by_ref (Referenz zuweisen)

assign_by_ref (Referenz zuweisen)

assign_by_ref (Referenz zuweisen)

void **assign_by_ref** (string varname, mixed var)

Weist einen Wert via Referenz zu, anstatt eine Kopie zu machen. Konsultieren Sie das PHP-Manual zum Thema 'variable referencing' für weitere Erklärungen.

> **Technical Note:** 'assign_by_ref()' ist effizienter als 'assign()', da keine Kopie der Variable erzeugt wird, sondern auf die Variable im Speicher referenziert wird. Beachten Sie dabei, dass eine nachträgliche änderung Original-Variable auch die zugewiesene Variable ändert. PHP5 wird die Referenzierung automatisch übernehmen, diese Funktion dient als Workaround.

Beispiel 13.4. assign_by_ref (via Referenz zuweisen)

```
<?php
// Namen/Wert-Paare &uuml;bergeben
$smarty->assign_by_ref('Name', $myname);
$smarty->assign_by_ref('Address', $address);
?>
```

clear_all_assign (alle Zuweisungen löschen)

clear_all_assign (alle Zuweisungen löschen)

clear_all_assign (alle Zuweisungen löschen)

void **clear_all_assign** (void)

Löscht die Werte aller zugewiesenen Variablen.

Beispiel 13.5. clear_all_assign (alle Zuweisungen löschen)

```
<?php
// l&ouml;sche alle zugewiesenen Variablen
$smarty->clear_all_assign();
?>
```

clear_all_cache (Cache vollständig leeren)

clear_all_cache (Cache vollständig leeren)

clear_all_cache (Cache vollständig leeren)

void **clear_all_cache** ([int expire_time])

Leert den gesamten Template-Cache. Als optionaler Parameter kann ein Mindestalter in Sekunden angegeben werden, das die einzelne Datei haben muss, bevor sie gelöscht wird.

Beispiel 13.6. clear_all_cache (Cache vollständig leeren)

```
<?php
// leere den gesamten cache
$smarty->clear_all_cache();
?>
```

clear_assign (lösche Zuweisung)

clear_assign (lösche Zuweisung)

clear_assign (lösche Zuweisung)

void **clear_assign** (mixed var)

Löscht den Wert einer oder mehrerer (übergabe als Array) zugewiesener Variablen.

Beispiel 13.7. clear_assign (lösche Zuweisung)

```
<?php
// l&ouml;sche eine einzelne Variable
$smarty->clear_assign("Name");

// l&ouml;sche mehrere Variablen
$smarty->clear_assign(array("Name", "Address", "Zip"));
?>
```

clear_cache (leere Cache)

clear_cache (leere Cache)

clear_cache (leere Cache)

void **clear_cache** (string template [, string cache_id [, string compile_id [, int expire_time]]])

Löscht den Cache eines bestimmten Templates. Falls Sie mehrere Caches für ein Template verwenden, können Sie als zweiten Parameter die 'cache_id' des zu leerenden Caches übergeben. Als dritten Parameter können sie die 'compile_id' angeben. Sie können Templates auch gruppieren und dann als Gruppe aus dem Cache löschen. Sehen sie dazu den Abschnitt über caching. Als vierten Parameter können Sie ein Mindestalter in Sekunden angeben, das ein Cache aufweisen muss, bevor er gelöscht wird.

Beispiel 13.8. clear_cache (Cache leeren)

```
<?php
// Cache eines Templates leeren
$smarty->clear_cache("index.tpl");

// leere den Cache einer bestimmten 'cache-id' eines mehrfach-gecachten Templates
$smarty->clear_cache("index.tpl", "CACHEID");
?>
```

clear_compiled_tpl (kompiliertes Template löschen)

clear_compiled_tpl (kompiliertes Template löschen)

clear_compiled_tpl (kompiliertes Template löschen)

void **clear_compiled_tpl** ([string tpl_file [, string compile_id [, int exp_time]]])

Löscht die kompilierte Version des angegebenen Templates. Falls kein Template-Name übergeben wird, werden alle kompilierten Templates gelöscht. Diese Funktion ist für fortgeschrittene Benutzer.

Beispiel 13.9. clear_compiled_tpl (kompiliertes Template löschen)

```
<?php
// ein bestimmtes kompiliertes Template l&ouml;schen
$smarty->clear_compiled_tpl("index.tpl");

// das gesamte Kompilier-Verzeichnis l&ouml;schen
$smarty->clear_compiled_tpl();
?>
```

clear_config

clear_config

clear_config

void **clear_config** ([string var])

Löscht alle zugewiesenen Konfigurations-Variablen. Wenn der Variablenname übergeben wird, wird nur diese Variable gelöscht.

Beispiel 13.10. clear_config

```
<?php
// alle config-variablen l&ouml;schen
$smarty->clear_config();

// eine l&ouml;schen
$smarty->clear_config('foobar');
?>
```

config_load

config_load

config_load

void **config_load** (string file [, string section])

Lädt die Konfigurationsdatei *file* und weist die Daten dem Template zu. Dies funktioniert identisch wie config_load.

> **Technische Bemerkung:** Seit Smarty 2.4.0 bleiben Variablen während fetch() und display() Aufrufen erhalten. Variablen, die mit config_load() geladen werden sind immer global deklariert. Konfigurationsdateien werden für eine schnellere Ausgabe ebenfalls kompiliert, und halten sich an die force_compile und compile_check Konfiguration.

Beispiel 13.11. config_load

```
<?php
// variablen laden und zuweisen
$smarty->config_load('my.conf');

// nur einen abschnitt laden
$smarty->config_load('my.conf', 'foobar');
?>
```

display (ausgeben)

display (ausgeben)

display (ausgeben)

void **display** (string template [, string cache_id [, string compile_id]])

Gibt ein Template aus. Sie müssen einen gültigen Template Ressourcen-Typ inklusive Pfad angeben. Als optionalen zweiten Parameter können Sie eine 'cache_id' übergeben. Konsultieren Sie den Abschnitt über caching für weitere Informationen.

Als optionalen dritten Parameter können Sie eine 'compile_id' übergeben. Dies ist wertvoll, falls Sie verschiedene Versionen eines Templates kompilieren wollen - zum Beispiel in verschiedenen Sprachen. 'compile_id' wird auch verwendet, wenn Sie mehr als ein '$template_dir' aber nur ein '$compile_dir' haben. Setzen Sie dazu für jedes Verzeichnis eine eigene 'compile_id', andernfalls werden Templates mit dem gleichen Namen überschrieben. Sie können die Variable $compile_id auch einmalig setzen, anstatt sie bei jedem Aufruf von 'display()' zu übergeben.

Beispiel 13.12. display (ausgeben)

```
<?php
include("Smarty.class.php");
$smarty = new Smarty;
$smarty->caching = true;

// Datenbank-Aufrufe nur durchf&uuml;hren, wenn kein Cache existiert
if(!$smarty->is_cached("index.tpl")) {

  // Beispieldaten
  $address = "245 N 50th";
  $db_data = array(
              "City" => "Lincoln",
              "State" => "Nebraska",
              "Zip" => "68502"
            );

  $smarty->assign("Name","Fred");
  $smarty->assign("Address",$address);
  $smarty->assign($db_data);

}

// ausgabe
$smarty->display("index.tpl");
?>
```

Verwenden Sie die Syntax von template resources um Dateien ausserhalb von '$template_dir' zu verwenden.

Beispiel 13.13. Beispiele von Template-Ressourcen für 'display()'

```
<?php
// absoluter Dateipfad
$smarty->display("/usr/local/include/templates/header.tpl");

// absoluter Dateipfad (alternativ)
$smarty->display("file:/usr/local/include/templates/header.tpl");

// absoluter Dateipfad unter Windows (MUSS mit 'file:'-Prefix versehen werden)
```

```
$smarty->display("file:C:/www/pub/templates/header.tpl");

// aus der Template-Ressource 'db' einbinden
$smarty->display("db:header.tpl");
?>
```

fetch

fetch

fetch

string **fetch** (string template [, string cache_id [, string compile_id]])

Gibt die Ausgabe des Template zurück, anstatt es direkt anzuzeigen. Übergeben Sie einen gültigen Template Ressource-Typ und -Pfad. Als optionaler zweiter Parameter kann eine 'cache_id' übergeben werden. Bitte konsultieren Sie den Abschnitt über caching für weitere Informationen.

Als optionalen dritten Parameter können Sie eine 'compile_id' übergeben. Dies ist wertvoll, falls Sie verschiedene Versionen eines Templates kompilieren wollen - zum Beispiel in verschiedenen Sprachen. 'compile_id' wird auch verwendet, wenn Sie mehr als ein '$template_dir' aber nur ein '$compile_dir' haben. Setzen Sie dann für jedes Verzeichnis eine eigene 'compile_id', andernfalls werden Templates mit dem gleichen Namen überschrieben. Sie können die Variable $compile_id auch einmalig setzen, anstatt sie bei jedem Aufruf von 'fetch()' zu übergeben.

Beispiel 13.14. fetch

```
<?php
include("Smarty.class.php");
$smarty = new Smarty;

$smarty->caching = true;

// Datenbank-Aufrufe nur durchf&uuml;hren, wenn kein Cache existiert
if(!$smarty->is_cached("index.tpl")) {

  // Beispieldaten
  $address = "245 N 50th";
  $db_data = array(
              "City" => "Lincoln",
              "State" => "Nebraska",
              "Zip" => "68502"
            );

  $smarty->assign("Name","Fred");
  $smarty->assign("Address",$address);
  $smarty->assign($db_data);

}

// ausgabe abfangen
$output = $smarty->fetch("index.tpl");

// Etwas mit $output anstellen

echo $output;
?>
```

get_config_vars

get_config_vars

get_config_vars

array **get_config_vars** ([string varname])

Gibt den Wert der Konfigurationsvariable zurück. Wenn kein Parameter übergeben wird, wird ein Array aller geladenen Variablen zurück gegeben.

Beispiel 13.15. get_config_vars

```
<?php
// variable 'foo' zuweisen
$foo = $smarty->get_config_vars('foo');

// alle geladenen konfigurationsvariablen zuweisen
$config_vars = $smarty->get_config_vars();

// ausgabe
print_r($config_vars);
?>
```

get_registered_object

get_registered_object

get_registered_object

array **get_registered_object** (string object_name)

Gibt eine Referenz zum registrerten Objekt zurück. Dies ist vorallem sinnvoll, um von einer eigenen Funktion auf ein registriertes Objekt zuzugreiffen.

Beispiel 13.16. get_registered_object

```
<?php
function smarty_block_foo($params, &$smarty)
{
  if (isset($params['object'])) {
    // referenz zuweisen
    $obj_ref = &$smarty->get_registered_object($params['object']);
    // $obj_ref ist nun ein pointer zum registrierten objekt
  }
}
?>
```

get_template_vars (Template-Variablen extrahieren)

get_template_vars (Template-Variablen extrahieren)

get_template_vars (Template-Variablen extrahieren)

array **get_template_vars** ([string varname])

Gibt ein Array der zugewiesenen Template-Variablen zurück.

Beispiel 13.17. get_template_vars (Template-Variablen extrahieren)

```
<?php
// foo extrahieren
$foo = $smarty->get_template_vars('foo');

// alle zugewiesenen Template-Variablen extrahieren
$tpl_vars = $smarty->get_template_vars();

// Anschauen
print_r($tpl_vars);
?>
```

is_cached (gecachte Version existiert)

is_cached (gecachte Version existiert)

is_cached (gecachte Version existiert)

bool **is_cached** (string template [, string cache_id [, string compile_id]])

Gibt 'true' zurück, wenn ein gültiger Cache für das angegebene Template existiert. Dies funktioniert nur, wenn caching eingeschaltet ist.

Beispiel 13.18. is_cached

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl")) {
// Datenbank-Abfragen, Variablen zuweisen...
}

$smarty->display("index.tpl");
?>
```

Als optionalen zweiten Parameter können Sie die 'cache_id' übergeben, falls Sie mehrere Caches für ein Template verwenden.

Beispiel 13.19. 'is_cached' bei mehreren Template-Caches

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl", "FrontPage")) {
        // Datenbank Abfragen, Variablen zuweisen...
}

$smarty->display("index.tpl", "FrontPage");
?>
```

> **Technische Bemerkung:** Wenn `is_cached` true zurück gibt, wird die Ausgabe geladen. Alle weiteren Aufrufe von display() oder fetch() werden aus diesem Cache bedient. Dies verhindert eine Race Condition, die auftauchen könnte, wenn ein anderes Script das besagte Template aus dem Cache löscht. Das bedeutet natürlich auch, dass clear_cache() und andere Cache spezifische Einstellungen keine Auswirkungen haben, nachdem `is_cached` true zurückgegeben hat.

load_filter

load_filter

load_filter

void **load_filter** (string type, string name)

Mit dieser Funktion können Filter-Plugins geladen werden. Der erste Parameter definiert den Filter-Typ und kann einen der folgenden Werte haben: 'pre', 'post', oder 'output'. Als zweiter Parameter wird der Name des Filter-Plugins angegeben, zum Beispiel 'trim'.

Beispiel 13.20. Filter-Plugins laden

```
<?php
$smarty->load_filter('pre', 'trim');        // lade den 'pre'-Filter (Vor-Filter) namens 'trim'
$smarty->load_filter('pre', 'datefooter');  // lade einen zweiten Vor-Filter namens 'datefo
oter'
$smarty->load_filter('output', 'compress'); // lade den 'output'-Filter (Ausgabe-Filter) n
amens 'compress'
?>
```

register_block (Block-Funktion registrieren)

register_block (Block-Funktion registrieren)

register_block (Block-Funktion registrieren)

void **register_block** (string name, mixed impl, bool cacheable, mixed cache_attrs)

Wird verwendet, um Block-Funktion-Plugins dynamisch zu registrieren. Übergeben Sie dazu den Namen der Block-Funktion und den Namen der PHP-Callback-Funktion, die die entsprechende Funktionalität bereitstellt.

Der Parameter *impl* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

$cacheable und *$cache_attrs* können in den meisten Fällen weggelassen werden. Konsultieren Sie Die Ausgabe von cachebaren Plugins Kontrollieren für weitere Informationen.

Beispiel 13.21. register_block (Block-Funktion registrieren)

```
<?php
$smarty->register_block("translate", "do_translation");

function do_translation ($params, $content, &$smarty, &$repeat)
{
  if (isset($content)) {
    $lang = $params['lang'];
        // &uuml;bersetze den Inhalt von '$content'
    return $translation;
  }
}
?>
```

Wobei das Template wie folgt aussieht:

```
{* template *}
{translate lang="br"}
Hello, world!
{/translate}
```

register_compiler_function (Compiler-Funktion registrieren)

register_compiler_function (Compiler-Funktion registrieren)

register_compiler_function (Compiler-Funktion registrieren)

bool **register_compiler_function** (string name, mixed impl, bool cacheable)

Wird verwendet, um Compiler-Funktion-Plugins dynamisch zu registrieren. Übergeben Sie dazu den Namen der Compiler-Funktion und den Namen der PHP-Funktion, die die entsprechende Funktionalität bereitstellt.

Der Parameter *impl* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

$cacheable und *$cache_attrs* können in den meisten Fällen weggelassen werden. Konsultieren Sie Die Ausgabe von cachebaren Plugins Kontrollieren für weitere Informationen.

register_function

register_function

register_function

void **register_function** (string name, mixed impl, bool cacheable, mixed cache_attrs)

Wird verwendet, um Template-Funktion-Plugins dynamisch zu registrieren. Übergeben Sie dazu den Namen der Template-Funktion und den Namen der PHP-Funktion, die die entsprechende Funktionalität bereitstellt.

Der Parameter *impl* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

$cacheable und *$cache_attrs* können in den meisten Fällen weggelassen werden. Konsultieren Sie Die Ausgabe von cachebaren Plugins Kontrollieren für weitere Informationen.

Beispiel 13.22. register_function (Funktion registrieren)

```
<?php
$smarty->register_function("date_now", "print_current_date");

function print_current_date($params)
{
  if(empty($params['format'])) {
    $format = "%b %e, %Y";
  } else {
    $format = $params['format'];
    return strftime($format,time());
  }
}

// Von nun an k&ouml;nnen Sie {date_now} verwenden, um das aktuelle Datum auszugeben.
// Oder {date_now format="%Y/%m/%d"}, wenn Sie es formatieren wollen.</programlisting>
?>
```

register_modifier (Modifikator-Plugin registrieren)

register_modifier (Modifikator-Plugin registrieren)

register_modifier (Modifikator-Plugin registrieren)

void **register_modifier** (string name, mixed impl)

Wird verwendet, um Modifikator-Plugins dynamisch zu registrieren. Übergeben Sie dazu den Namen der Modifikator-Funktion und den Namen der PHP-Funktion, die die entsprechende Funktionalität bereitstellt.

Der Parameter *impl* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

Beispiel 13.23. register_modifier (Modifikator-Plugin registrieren)

```
<?php
// PHP's 'stripslashes()'-Funktion als Smarty Modifikator registrieren

$smarty->register_modifier("sslash", "stripslashes");

// Von nun an k&ouml;nnen Sie {$var|sslash} verwenden,
// um "\"-Zeichen (Backslash) aus Zeichenketten zu entfernen. ('\\' wird zu '\',...)
?>
```

register_object

register_object

register_object

void **register_object** (string object_name, object object, array allowed_methods_properties, boolean format, array block_methods)

Wird verwendet um ein Objekt zu registrieren. Konsultieren Sie den Abschnitt Objekte für weitere Informationen und Beispiele.

register_outputfilter (Ausgabefilter registrieren)

register_outputfilter (Ausgabefilter registrieren)

register_outputfilter (Ausgabefilter registrieren)

void **register_outputfilter** (mixed function)

Verwenden Sie diese Funktion um dynamisch Ausgabefilter zu registrieren, welche die Template Ausgabe verarbeiten bevor sie angezeigt wird. Konsultieren Sie den Abschnitt über Ausgabefilter für mehr Informationen.

Der Parameter *function* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

register_postfilter ('post'-Filter registrieren)

register_postfilter ('post'-Filter registrieren)

register_postfilter ('post'-Filter registrieren)

void **register_postfilter** (mixed function)

Wird verwendet, um 'post'-Filter dynamisch zu registrieren. 'post'-Filter werden auf das kompilierte Template angewendet. Konsultieren Sie dazu den Abschnitt template postfilters.

Der Parameter *function* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

register_prefilter ('pre'-Filter registrieren)

register_prefilter ('pre'-Filter registrieren)

register_prefilter ('pre'-Filter registrieren)

void **register_prefilter** (mixed function)

Wird verwendet, um 'pre'-Filter dynamisch zu registrieren. 'pre'-Filter werden vor der Kompilierung auf das Template angewendet. Konsultieren Sie dazu den Abschnitt 'pre'-Filter.

Der Parameter *function* kann als (a) einen Funktionnamen oder (b) einem Array der Form `array(&$object, $method)`, wobei `&$object` eine Referenz zu einem Objekt und `$method` der Name der Methode die aufgerufen werden soll ist, oder als Array der Form `array(&$class, $method)`, wobei `$class` der Name der Klasse und `$method` der Name der Methode ist die aufgerufen werden soll, übergeben werden.

register_resource (Ressource registrieren)

register_resource (Ressource registrieren)

register_resource (Ressource registrieren)

void **register_resource** (string name, array resource_funcs)

Wird verwendet, um ein Ressource-Plugin dynamisch zu registrieren. Übergeben Sie dazu den Ressourcen-Namen und das Array mit den Namen der PHP-Funktionen, die die Funktionalität implementieren. Konsultieren Sie den Abschnitt template resources für weitere Informationen zum Thema.

> **Technische Bemerkung:** Ein Ressourcename muss mindestens 2 Zeichen lang sein. Namen mit einem (1) Zeichen werden ignoriert und als Teil des Pfades verwenden, wie in $smarty->display('c:/path/to/index.tpl');.

Der Parameter *resource_funcs* muss aus 4 oder 5 Elementen bestehen. Wenn 4 Elemente übergeben werden, werden diese als Ersatz Callback-Funktionen fü "source", "timestamp", "secure" und "trusted" verwendet. Mit 5 Elementen muss der erste Parameter eine Referenz auf das Objekt oder die Klasse sein, welche die benötigten Methoden bereitstellt.

Beispiel 13.24. register_resource (Ressource registrieren)

```
<?php
$smarty->register_resource("db", array("db_get_template",
"db_get_timestamp",
"db_get_secure",
"db_get_trusted"));
?>
```

trigger_error (Fehler auslösen)

trigger_error (Fehler auslösen)

trigger_error (Fehler auslösen)

void **trigger_error** (string error_msg [, int level])

Wird verwendet, um eine Fehlermeldung via Smarty auszugeben. Der *level*-Parameter kann alle Werte der 'trigger_error()'-PHP-Funktion haben, zum Beispiel E_USER_NOTICE, E_USER_WARNING, usw. Voreingestellt ist E_USER_WARNING.

template_exists (Template existiert)

template_exists (Template existiert)

template_exists (Template existiert)

bool **template_exists** (string template)

Diese Funktion prüft, ob das angegebene Template existiert. Als Parameter können entweder ein Pfad im Dateisystem oder eine Ressource übergeben werden.

unregister_block (Block-Funktion deaktivieren)

unregister_block (Block-Funktion deaktivieren)

unregister_block (Block-Funktion deaktivieren)

void **unregister_block** (string name)

Wird verwendet, um registrierte Block-Funktionen auszuschalten. Übergeben Sie dazu den Namen der Block-Funktion.

unregister_compiler_function (Compiler-Funktion deaktivieren)

unregister_compiler_function (Compiler-Funktion deaktivieren)

unregister_compiler_function (Compiler-Funktion deaktivieren)

void **unregister_compiler_function** (string name)

Wird verwendet, um registrierte Compiler-Funktionen auszuschalten. Übergeben Sie dazu den Funktionsnamen der Compiler-Funktion.

unregister_function (Template-Funktion deaktivieren)

unregister_function (Template-Funktion deaktivieren)

unregister_function (Template-Funktion deaktivieren)

void **unregister_function** (string name)

Wird verwendet, um registrierte Template-Funktionen auszuschalten. Übergeben Sie dazu den Namen der Template-Funktion.

Beispiel 13.25. unregister_function

```
<?php
// Template-Designer sollen keinen Zugriff auf das Dateisystem haben
$smarty->unregister_function("fetch");
?>
```

unregister_modifier (Modifikator deaktivieren)

unregister_modifier (Modifikator deaktivieren)

unregister_modifier (Modifikator deaktivieren)

void **unregister_modifier** (string name)

Wird verwendet, um registrierte Variablen-Modifikatoren auszuschalten. Übergeben Sie dazu den Modifikator-Namen.

Beispiel 13.26. unregister_modifier

```
<?php
// Verhindern, dass Template-Designer 'strip_tags' anwenden

$smarty->unregister_modifier("strip_tags");
?>
```

unregister_object

unregister_object

unregister_object

void **unregister_object** (string object_name)

Pointer zu einem registrierten Objekt löschen

unregister_outputfilter (Ausgabefilter deaktivieren)

unregister_outputfilter (Ausgabefilter deaktivieren)

unregister_outputfilter (Ausgabefilter deaktivieren)

void **unregister_outputfilter** (string function_name)

Wird verwendet, um registrierte Ausgabefilter auszuschalten.

unregister_postfilter ('post'-Filter deaktivieren)

unregister_postfilter ('post'-Filter deaktivieren)

unregister_postfilter ('post'-Filter deaktivieren)

void **unregister_postfilter** (string function_name)

Wird verwendet, um registrierte 'post'-Filter auszuschalten.

unregister_prefilter ('pre'-Filter deaktiviern)

unregister_prefilter ('pre'-Filter deaktiviern)

unregister_prefilter ('pre'-Filter deaktiviern)

void **unregister_prefilter** (string function_name)

Wird verwendet, um registrierte 'pre'-Filter auszuschalten.

unregister_resource (Ressource deaktivieren)

unregister_resource (Ressource deaktivieren)

unregister_resource (Ressource deaktivieren)

void **unregister_resource** (string name)

Wird verwendet, um registrierte Ressourcen auszuschalten. Übergeben Sie dazu den Namen der Ressource.

Beispiel 13.27. unregister_resource (Ressource deaktivieren)

```
<?php
$smarty->unregister_resource("db");
?>
```

Kapitel 14. Caching

Inhaltsverzeichnis

Caching wird verwendet, um display() oder fetch() Aufrufe durch zwischenspeichern (cachen) der Ausgabe in einer Datei zu beschleunigen. Falls eine gecachte Version des Aufrufs existiert, wird diese ausgegeben, anstatt die Ausgabe neu zu generieren. Caching kann die Performance vor allem dann deutlich verbessern, wenn Templates längere Rechenzeit beanspruchen. Weil die Ausgabe von display() und fetch() gecached wird, kann ein Cache verschiedene Templates, Konfigurationsdateien usw. enthalten.

Da Templates dynamisch sind ist es wichtig darauf zu achten, welche Inhalte für für wie lange gecached werden sollen. Wenn sich zum Beispiel die erste Seite Ihrer Website nur sporadisch ändert, macht es Sinn die Seite für eine Stunde oder länger zu cachen. Wenn Sie aber eine Seite mit sich minütlich erneuernden Wetterinformationen haben, macht es möglicherweise keinen Sinn, die Seite überhaupt zu cachen.

Caching einrichten

Als erstes muss das Caching eingeschaltet werden. Dies erreicht man, indem $caching = 1 (oder 2) gesetzt wird.

Beispiel 14.1. Caching einschalten

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$smarty->display('index.tpl');
?>
```

Wenn Caching eingeschaltet ist, wird der Funktionsaufruf display('index.tpl') das Template normal rendern, zur selben Zeit jedoch auch eine Datei mit dem Inhalt in das $cache_dir schreiben (als gecachte Kopie). Beim nächsten Aufruf von display('index.tpl') wird die gecachte Kopie verwendet.

> **Technische Bemerkung:** Die im $cache_dir abgelegen Dateien haben einen ähnlichen Namen wie das Template, mit dem sie erzeugt wurden. Obwohl sie eine '.php'-Endung aufweisen, sind sie keine ausführbaren PHP-Skripte. Editieren Sie diese Dateien NICHT!

Jede gecachte Seite hat eine Lebensdauer, die von $cache_lifetime bestimmt wird. Normalerweise beträgt der Wert 3600 Sekunden (= 1 Stunde). Nach Ablauf dieser Lebensdauer wird der Cache neu generiert. Sie können die Lebensdauer pro Cache bestimmen indem Sie $caching auf 2 setzen. Konsultieren Sie den Abschnitt über $cache_lifetime für weitere Informationen.

Beispiel 14.2. '$cache_lifetime' pro Cache einstellen

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 2; // Lebensdauer ist pro Cache

// Standardwert für '$cache_lifetime' auf 5 Minuten setzen
$smarty->cache_lifetime = 300;
$smarty->display('index.tpl');

// '$cache_lifetime' für 'home.tpl' auf 1 Stunde setzen
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');

// ACHTUNG: die folgende Zuweisung an '$cache_lifetime' wird nicht funktionieren,
// wenn '$caching' auf 2 gestellt ist. Wenn die '$cache_lifetime' für 'home.tpl' bereits
// auf 1 Stunde gesetzt wurde, werden neue Werte ignoriert.
// 'home.tpl' wird nach dieser Zuweisung immer noch eine '$cache_lifetime' von 1 Stunde haben
$smarty->cache_lifetime = 30; // 30 seconds
$smarty->display('home.tpl');
?>
```

Wenn $compile_check eingeschaltet ist, werden alle in den Cache eingeflossenen Templates und Konfigurationsdateien hinsichtlich ihrer letzten Änderung überprüft. Falls eine der Dateien seit der Erzeugung des Cache geändert wurde, wird der Cache unverzüglich neu generiert. Dadurch ergibt sich ein geringer Mehraufwand. Für optimale Performance sollte $compile_check deshalb auf 'false' gesetzt werden.

Beispiel 14.3. '$compile_check' einschalten

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;
$smarty->compile_check = true;

$smarty->display('index.tpl');
?>
```

Wenn $force_compile eingeschaltet ist, werden die Cache-Dateien immer neu generiert und das Caching damit wirkungslos gemacht. $force_compile wird normalerweise nur für die Fehlersuche verwendet. Ein effizienterer Weg das Caching auszuschalten wäre, $caching auf 'false' (oder 0) zu setzen.

Mit der Funktion is_cached() kann überprüft werden, ob von einem Template eine gecachte Version vorliegt. In einem Template, das zum Beispiel Daten aus einer Datenbank bezieht, können Sie diese Funktion verwenden, um den Prozess zu überspringen.

Beispiel 14.4. is_cached() verwenden

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

if(!$smarty->is_cached('index.tpl')) {
    // kein Cache gefunden, also Variablen zuweisen
    $contents = get_database_contents();
    $smarty->assign($contents);
```

```
}

$smarty->display('index.tpl');
?>
```

Mit der {insert} Funktion können Sie Teile einer Seite dynamisch halten. Wenn zum Beispiel ein Banner in einer gecachten Seite nicht gecached werden soll, kann dessen Aufruf mit {insert} dynamisch gehalten werden. Konsultieren Sie den Abschnitt über insert für weitere Informationen und Beispiele.

Mit der Funktion clear_all_cache() können Sie den gesamten Template-Cache löschen. Mit clear_cache() einzelne Templates oder Cache-Gruppen.

Beispiel 14.5. Cache leeren

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// alle Cache-Dateien löschen
$smarty->clear_all_cache();

// nur Cache von 'index.tpl' löschen
$smarty->clear_cache('index.tpl');

$smarty->display('index.tpl');
?>
```

Multiple Caches für eine Seite

Sie können für Aufrufe von display() oder fetch() auch mehrere Caches erzeugen. Nehmen wir zum Beispiel an, der Aufruf von display('index.tpl') erzeuge für verschieden Fälle unterschiedliche Inhalte und Sie wollen jeden dieser Inhalte separat cachen. Um dies zu erreichen, können Sie eine 'cache_id' beim Funktionsaufruf übergeben.

Beispiel 14.6. 'display()' eine 'cache_id' übergeben

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

$smarty->display('index.tpl',$my_cache_id);
?>
```

Im oberen Beispiel übergeben wir die Variable *$my_cache_id* als 'cache_id' an display(). Für jede einmalige *cache_id* wird ein eigener Cache von 'index.tpl' erzeugt. In diesem Beispiel wurde 'article_id' per URL übergeben und als 'cache_id' verwendet.

> **Technische Bemerkung:** Seien Sie vorsichtig, wenn Sie Smarty (oder jeder anderen PHP-Applikation) Werte direkt vom Client (Webbrowser) übergeben. Obwohl das Beispiel oben praktisch aussehen mag, kann es

schwerwiegende Konsequenzen haben. Die 'cache_id' wird verwendet, um im Dateisystem ein Verzeichnis zu erstellen. Wenn ein Benutzer also überlange Werte übergibt oder ein Skript benutzt, das in hohem Tempo neue 'article_ids' übermittelt, kann dies auf dem Server zu Problemen führen. Stellen Sie daher sicher, dass Sie alle empfangenen Werte auf ihre Gültigkeit überprüfen und unerlaubte Sequenzen entfernen. Sie wissen möglicherweise, dass ihre 'article_id' nur 10 Zeichen lang sein kann, nur aus alphanumerischen Zeichen bestehen darf und in der Datenbank eingetragen sein muss. Überpüfen sie das!

Denken Sie daran, Aufrufen von is_cached() und clear_cache() als zweiten Parameter die 'cache_id' zu übergeben.

Beispiel 14.7. 'is_cached()' mit 'cache_id' aufrufen

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

if(!$smarty->is_cached('index.tpl',$my_cache_id)) {
    // kein Cache gefunden, also Variablen zuweisen
    $contents = get_database_contents();
    $smarty->assign($contents);
}

$smarty->display('index.tpl',$my_cache_id);
?>
```

Sie können mit clear_cache() den gesamten Cache einer bestimmten 'cache_id' auf einmal löschen, wenn Sie als Parameter die 'cache_id' übergeben.

Beispiel 14.8. Cache einer bestimmten 'cache_id' leeren

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// Cache mit 'sports' als 'cache_id' löschen
$smarty->clear_cache(null,"sports");

$smarty->display('index.tpl',"sports");
?>
```

Indem Sie allen dieselbe 'cache_id' übergeben, lassen sich Caches gruppieren.

Cache-Gruppen

Sie können auch eine feinere Gruppierung vornehmen, indem Sie 'cache_id'-Gruppen erzeugen. Dies erreichen Sie, indem Sie jede Cache-Untergruppe durch ein '|'-Zeichen (pipe) in der 'cache_id' abtrennen. Sie können so viele Untergruppen erstellen, wie Sie möchten.

Man kann Cache-Gruppen wie eine Verzeichnishierarchie betrachten. Zum Beispiel kann man sich die Cache-Gruppe "a|b|c" als eine Verzeichnisstruktur "/a/b/c" angesehen weden. clear_cache(null, 'a|b|c') würde die Dateien '/a/b/c/*' löschen, clear_cache(null, 'a|b') wäre das Löschen der Dateien '/a/b/*'. Wenn eine Compile-Id angegeben wurde, wie

clear_cache(null, 'a|b', 'foo'), dann wird die Compile-Id so behandelt, als sei sie an die Cache-Gruppe angehängt, also wie die Cache-Gruppe '/a/b/foo'. Wenn ein Templatename angegeben wurde, also wie bei clear_cache('foo.tpl', 'a|b|c'), dann wir Smarty auch nur '/a/b/c/foo.tpl' löschen. Es ist NICHT möglich, ein Template unterhalb mehrerer Cache-Gruppen (also '/a/b/*/foo.tpl') zu löschen. Das Gruppieren der Cache-Gruppen funktioniert nur von links nach rechts. Man muss die Templates, die man als eine Gruppe löschen möchte alle unterhalb einer einzigen Gruppenhierarchy anordnen, um sie als eine Gruppe löschen zu können.

Cache-Gruppen dürfen nicht mit der Hierarchie des Template-Verzeichnisses verwechselt werden. Die Cache-Gruppen wissen nicht, wie die Templatehierarchie strukturiert ist. Wenn man z. B. eine Templatestruktur wir "themes/blue/index.tpl" hat und man möchte alle Dateien für des "blue"-Theme löschen, dann muss man händisch eine Cache-Gruppe wie display("themes/blue/index.tpl", "themes|blue") und kann diese dann mit clear_cache(null,"themes|blue") löschen.

Beispiel 14.9. 'cache_id'-Gruppen

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// leere alle Caches welche 'sports|basketball' als erste zwei 'cache_id'-Gruppen enthalten
$smarty->clear_cache(null, 'sports|basketball');

// leere alle Caches welche 'sports' als erste 'cache_id'-Gruppe haben. Dies schliesst
// 'sports|basketball', oder 'sports|(anything)|(anything)|(anything)|...' ein
$smarty->clear_cache(null, 'sports');

$smarty->display('index.tpl', 'sports|basketball');
?>
```

Die Ausgabe von cachebaren Plugins Kontrollieren

Seit Smarty-2.6.0 kann bei der Registrierung angegeben werden ob ein Plugin cached werden soll. Der dritte Parameter für register_block, register_compiler_function und register_function heisst *$cacheable*, der Standardwert ist TRUE, was das Verhalten von Smarty vor Version 2.6.0 wiederspiegelt.

Wenn ein Plugin mit *$cacheable=false* registriert wird, wird er bei jedem Besuch der Seite aufgerufen, selbst wenn die Site aus dem Cache stammt. Die Pluginfunktion verhält sich ein wenig wie {insert}.

Im Gegensatz zu {insert} werden die Attribute standartmässig nicht gecached. Sie können das caching jedoch mit dem vierten Parameter *$cache_attrs* kontrollieren. *$cache_attrs* ist ein Array aller Attributnamen die gecached werden sollen.

Beispiel 14.10. Verhindern des Caching der Ausgabe eines Plugins

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->caching = true;

function remaining_seconds($params, &$smarty) {
    $remain = $params['endtime'] - time();
    if ($remain >=0)
        return $remain . " second(s)";
    else
        return "done";
}

$smarty->register_function('remaining', 'remaining_seconds', false, array('endtime'));
```

```
if (!$smarty->is_cached('index.tpl')) {
 // Objekt $obj aus Datenbank dem Template zuweisen
  $smarty->assign_by_ref('obj', $obj);
}

$smarty->display('index.tpl');
?>
```

Bei folgendem index.tpl:

```
Verbleibende Zeit: {remain endtime=$obj->endtime}
```

Der Wert von $obj->endtime ändert bei jeder Anzeige der Seite, selbst wenn die Seite gecached wurde. Das Objekt $obj wird nur geladen wenn die Seite nicht gecached wurde.

Beispiel 14.11. Verhindern dass Template Blöcke gecached werden

```
index.php:

<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->caching = true;

function smarty_block_dynamic($param, $content, &$smarty) {
    return $content;
}
$smarty->register_block('dynamic', 'smarty_block_dynamic', false);

$smarty->display('index.tpl');
?>
```

Bei folgendem index.tpl:

```
Seite wurde erzeugt: {"0"|date_format:"%D %H:%M:%S"}

{dynamic}

Jetzt ist es: {"0"|date_format:"%D %H:%M:%S"}

... weitere Ausgaben ...

{/dynamic}
```

Um sicherzustellen dass ein Teil eines Templates nicht gecached werden soll, kann dieser Abschnitt in einen {dynamic}...{/dynamic} Block verpackt werden.

Kapitel 15. Advanced Features

Inhaltsverzeichnis

Objekte

Smarty erlaubt es, auf PHP-Objekte [http://php.net/object] durch das Template zuzugreifen. Dafür gibt es zwei Wege. Der erste ist, Objekte zu registrieren und wie auf eine eigene Funktion zuzugreifen. Der andere Weg ist, das Objekt dem Template mit assign() zuzuweisen und darauf wie auf andere Variablen zuzugreifen. Die erste Methode hat eine nettere Template Syntax und ist sicherer da der Zugriff auf ein registriertes Objekt mit Sicherheitseinstellungen kontrolliert werden kann. Der Nachteil ist, dass über registrierte Objekte nicht in einer Schlaufe gelaufen werden kann und, dass es nicht möglich ist, Arrays registrierten Objekten anzulegen. Welchen Weg Sie einschlagen wird von Ihren Bedürfnissen definiert, die erste Methode ist jedoch zu bevorzugen.

Wenn die Sicherheitsfunktionen eingeschaltet sind, können keine private Methoden (solche die einen '_'-Prefix tragen) aufgerufen werden. Wenn eine Methode und eine Eigeschaft mit dem gleichen Namen existieren wird die Methode verwendet.

Sie können den Zugriff auf Methoden und Eigenschaften einschränken indem Sie sie als Array als dritten Registrationsparameter übergeben.

Normalerweise werden Parameter welche einem Objekt via Template übergeben werden genau so übergeben wie dies bei normalen eigenen Funktionen der Fall ist. Das erste Objekt ist ein assoziatives Array und das zweite das Smarty Objekt selbst. Wenn Sie die Parameter einzeln erhalten möchten können Sie den vierten Parameter auf `false` setzen.

Der optionale fünfte Parameter hat nur einen Effekt wenn *format* = `true` ist und eine Liste von Methoden enthält die als Block verarbeitet werden sollen. Das bedeutet, dass solche Methoden ein schliessendes Tag im Template enthalten müssen (`{foobar->meth2}...{/foobar->meth2}`) und die Parameter zu den Funktionen die selbe Syntax haben wie block-function-plugins: sie erhalten also die 4 Parameter *$params*, *$content*, *&$smarty* und *&$repeat*, und verhalten sich auch sonst wie block-function-plugins.

Beispiel 15.1. registierte oder zugewiesene Objekte verwenden

```
<?php
// Das Objekt

class My_Object {
    function meth1($params, &$smarty_obj) {
        return "meine meth1";
    }
}

$myobj = new My_Object;
// Objekt registrieren (referenz)
$smarty->register_object("foobar",$myobj);
// Zugriff auf Methoden und Eigeschaften einschränken
```

```
$smarty->register_object("foobar",$myobj,array('meth1','meth2','prop1'));
// wenn wir das traditionelle Parameterformat verwenden wollen, übergeben wir false für den Parameter f
$smarty->register_object("foobar",$myobj,null,false);

// Objekte zuweisen (auch via Referenz möglich)
$smarty->assign_by_ref("myobj", $myobj);

$smarty->display('index.tpl');
?>
```

Und hier das dazugehörige index.tpl:

```
{* Zugriff auf ein registriertes objekt *}
{foobar->meth1 p1="foo" p2=$bar}

{* Ausgabe zuweisen *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
ausgabe war: {$output}

{* auf unser zugewiesenes Objekt zugreifen *}
{$myobj->meth1("foo",$bar)}
```

Siehe auch register_object() und assign()

Prefilter

Template Prefilter sind Filter, welche auf das Template vor dessen Kompilierung angewendet werden. Dies ist nützlich, um zum Beispiel Kommentare zu entfernen oder um den Inhalt des Templates zu analysieren. Prefilter können auf verschiedene Arten geladen werden. Man kann sie registrieren, aus dem Plugin-Verzeichnis mit load_filter() laden oder $autoload_filters verwenden. Smarty übergibt der Funktion als ersten Parameter den Template-Quellcode und erwartet als Rückgabewert den bearbeiteten Quellcode.

Beispiel 15.2. Template Prefilter verwenden

Dieser Prefiler entfernt alle Kommentare aus dem Template-Quelltext

```
<?php

// fügen Sie folgende Zeilen in Ihre Applikation ein
function remove_dw_comments($tpl_source, &$smarty)
{
    return preg_replace("/<!--#.*-->/U",'',$tpl_source);
}

// registrieren Sie den Prefilter
$smarty->register_prefilter("remove_dw_comments");
$smarty->display("index.tpl");
?>

{* Smarty Template 'index.tpl' *}

<!--# diese Zeile wird vom Prefilter entfernt-->
```

Sie auch register_prefilter(), Postfilter und load_filter()

Postfilter

Template Postfilter sind Filter, welche auf das Template nach dessen Kompilierung angewendet werden. Postfilter können auf verschiedene Arten geladen werden. Man kann sie registrieren, aus dem Plugin-Verzeichnis mit load_filter() laden oder $autoload_filters verwenden. Smarty übergibt der Funktion als ersten Parameter den Template-Quellcode und erwartet als Rückgabewert den bearbeiteten Quellcode.

Beispiel 15.3. Template Postfilter verwenden

```
<?php

// fügen Sie folgende Zeilen in Ihre Applikation ein
function add_header_comment($tpl_source, &$smarty)
{
    return "<?php echo \"<!-- Created by Smarty! -->\n\" ?>\n".$tpl_source;
}

// registrieren Sie den Postfilter
$smarty->register_postfilter("add_header_comment");
$smarty->display("index.tpl");
?>

{* kompiliertes Smarty Template 'index.tpl' *}
<!-- Created by Smarty! -->
{* Rest des Template Inhalts... *}
```

Sie auch register_postfilter(), Prefilter und load_filter()

Ausgabefilter

Wenn ein Template mit 'display()' oder 'fetch()' benutzt wird, kann die Ausgabe durch verschieden Ausgabefilter geschleust werden. Der Unterschied zu 'post'-Filtern ist, dass Ausgabefilter auf die durch 'fetch()' oder 'display()' erzeugte Ausgabe angewendet werden, 'post'-Filter aber auf das Kompilat vor seiner Speicherung im Dateisystem.

Ausgabefilter können auf verschiede Arten geladen werden. Man kann sie registrieren, aus dem Plugin-Verzeichnis mit load_filter() laden oder $autoload_filters verwenden. Smarty übergibt der Funktion als ersten Parameter die Template-Ausgabe und erwartet als Rückgabewert die bearbeitete Ausgabe.

Beispiel 15.4. Ausgabefilter verwenden

```
<?php

// fügen Sie folgende Zeilen in Ihre Applikation ein
function protect_email($tpl_output, &$smarty)
{
    $tpl_output = preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                               '$1%40$2', $tpl_output);
    return $tpl_output;
}


// Ausgabefilter registrieren
$smarty->register_outputfilter("protect_email");
$smarty->display("index.tpl");

// von nun an erhalten alle ausgegebenen e-mail Adressen einen
// einfach Schutz vor Spambots.
?>
```

Cache Handler Funktion

Als Alternative zum normalen dateibasierten Caching-Mechanismus können Sie eine eigene Cache-Handler Funktion zum lesen, schreiben und löschen von Cache-Dateien definieren.

Schreiben Sie eine Funktion in Ihrer Applikation, die Smarty als Cache-Handler verwenden soll und weisen Sie deren Name der Variable $cache_handler_func zu. Smarty wird von da an Ihre Funktion zur Bearbeitung des Caches verwenden. Als erster Parameter wird die 'action' mit einem der folgendende Werte übergeben: 'read', 'write' und 'clear'. Als zweiter Parameter wird das Smarty-Objekt übergeben, als dritter der gecachte Inhalt. Bei einem 'write' übergibt Smarty den gecachten Inhalt, bei 'read' übergibt Smarty die Variable als Referenz und erwartet, dass Ihre Funktion die Inhalte zuweist. Bei 'clear' können Sie eine dummy-Variable übergeben. Als vierter Parameter wird der Template-Name übergeben (verwendet bei 'write'/'read'), als fünfter Parameter die 'cache_id' (optional) und als sechster die 'compile_id' (auch optional).

Der letzte Parameter (*$exp_time*) wurde in Smarty-2.6.0 hinzugefügt.

Beispiel 15.5. Beispiel mit einer MySQL Datenbank als Datenquelle

```
<?php
/*

Beispiel Anwendung:

include('Smarty.class.php');
include('mysql_cache_handler.php');

$smarty = new Smarty;
$smarty->cache_handler_func = 'mysql_cache_handler';

$smarty->display('index.tpl');


die Datenbank hat folgendes Format:

create database SMARTY_CACHE;

create table CACHE_PAGES(
CacheID char(32) PRIMARY KEY,
CacheContents MEDIUMTEXT NOT NULL
);

*/

function mysql_cache_handler($action, &amp;$smarty_obj, &amp;$cache_content, $tpl_file=null, $cache_id=
{

    // Datenbank Host, Benutzer und Passwort festlegen
   $db_host = 'localhost';
   $db_user = 'myuser';
   $db_pass = 'mypass';
   $db_name = 'SMARTY_CACHE';
   $use_gzip = false;

   // enmalige 'cache_id' erzeugen
   $CacheID = md5($tpl_file.$cache_id.$compile_id);

   if(! $link = mysql_pconnect($db_host, $db_user, $db_pass)) {
      $smarty_obj->_trigger_error_msg("cache_handler: could not connect to database");
      return false;
   }
   mysql_select_db($db_name);

   switch ($action) {
   case 'read':

      // Cache aus der Datenbank lesen
      $results = mysql_query("select CacheContents from CACHE_PAGES where CacheID='$CacheID'");
```

```
      if(!$results) {
         $smarty_obj->_trigger_error_msg("cache_handler: query failed.");
      }
      $row = mysql_fetch_array($results,MYSQL_ASSOC);

      if($use_gzip &amp;&amp; function_exists("gzuncompress")) {
         $cache_contents = gzuncompress($row["CacheContents"]);
      } else {
         $cache_contents = $row["CacheContents"];
      }
      $return = $results;
      break;

   case 'write':

      // Cache in Datenbank speichern
      if($use_gzip &amp;&amp; function_exists("gzcompress")) {
         // compress the contents for storage efficiency
         $contents = gzcompress($cache_content);
      } else {
         $contents = $cache_content;
      }
      $results = mysql_query("replace into CACHE_PAGES values(
   '$CacheID',
   '".addslashes($contents)."')
   ");
      if(!$results) {
         $smarty_obj->_trigger_error_msg("cache_handler: query failed.");
      }
      $return = $results;
      break;
   case 'clear':

      // Cache Informationen löschen
      if(empty($cache_id) &amp;&amp; empty($compile_id) &amp;&amp; empty($tpl_file)) {

         // alle löschen
         $results = mysql_query("delete from CACHE_PAGES");
      } else {
         $results = mysql_query("delete from CACHE_PAGES where CacheID='$CacheID'");
      }
      if(!$results) {
         $smarty_obj->_trigger_error_msg("cache_handler: query failed.");
      }
      $return = $results;
      break;
   default:

      // Fehler, unbekannte 'action'
      $smarty_obj->_trigger_error_msg("cache_handler: unknown action \"$action\"");
      $return = false;
      break;
   }
   mysql_close($link);
   return $return;

}

?>
```

Ressourcen

Ein Template kann aus verschiedenen Quellen bezogen werden. Wenn Sie ein Template mit 'display()' ausgeben, die Ausgabe mit 'fetch()' in einer Variablen speichern oder innnerhalb eines Template ein weiteres Template einbinden, müssen Sie den Ressourcen-Typ, gefolgt von Pfad und Template-Namen angeben. Wenn kein Resourcetyp angegeben wird, wird $default_resource_type verwendet.

Templates aus dem '$template_dir'

Templates aus dem '$template_dir' benötigen normalerweise keinen Ressourcen-Typ, es wird jedoch empfohlen 'file:' zu verwenden. Übergeben Sie einfach den Pfad, in dem sich das Template relativ zu '$template_dir' befindet.

Beispiel 15.6. Templates aus '$template_dir' verwenden

```
// im PHP-Skript
      $smarty->display("index.tpl");
      $smarty->display("admin/menu.tpl");
      $smarty->display("file:admin/menu.tpl"); // entspricht der vorigen Zeile


      {* im Smarty Template *}
      {include file="index.tpl"}
      {include file="file:index.tpl"} {* entspricht der vorigen Zeile *}
```

Templates aus beliebigen Verzeichnissen

Templates ausserhalb von '$template_dir' benötigen den 'file:' Ressourcen-Typ, gefolgt von absolutem Pfadnamen und Templatenamen.

Beispiel 15.7. Templates aus beliebigen Verzeichnissen benutzen

```
// im PHP-Skript
      $smarty->display("file:/export/templates/index.tpl");
      $smarty->display("file:/path/to/my/templates/menu.tpl");


      {* im Smarty Template *}
      {include file="file:/usr/local/share/templates/navigation.tpl"}
```

Windows Dateipfade

Wenn Sie auf einer Windows-Maschine arbeiten, enthalten absoluten Dateipfade normalerweise den Laufwerksbuchstaben (C:). Stellen Sie sicher, dass alle Pfade den Ressourcen-Typ 'file:' haben, um Namespace-Konflikten vorzubeugen.

Beispiel 15.8. Templates aus Windows Dateipfaden verwenden

```
// im PHP-Skript
       $smarty->display("file:C:/export/templates/index.tpl");
       $smarty->display("file:F:/path/to/my/templates/menu.tpl");


       {* im Smarty Template *}
       {include file="file:D:/usr/local/share/templates/navigation.tpl"}
```

Templates aus anderen Quellen

Sie können Templates aus jeder für PHP verfügbaren Datenquelle beziehen: Datenbanken, Sockets, LDAP, usw. Dazu müssen sie nur ein Ressource-Plugin schreiben und registrieren.

Konsultieren Sie den Abschnitt über Ressource-Plugins für mehr Informationen über die Funktionalitäten, die ein derartiges Plugin bereitstellen muss.

> **Anmerkung:** Achtung: Sie können die interne `file` Ressource nicht überschreiben. Es steht Ihnen jedoch frei, ein Plugin zu schreiben, das die gewünschte Funktionalität implementiert und es als alternativen Ressource-Typ zu registrieren.

Beispiel 15.9. Eigene Quellen verwenden

```
// im PHP-Skript


      // definieren Sie folgende Funktion in Ihrer Applikation
      function db_get_template ($tpl_name, &tpl_source, &$smarty_obj)
      {
      // Datenbankabfrage um unser Template zu laden,
      // und '$tpl_source' zuzuweisen
      $sql = new SQL;
      $sql->query("select tpl_source
      from my_table
      where tpl_name='$tpl_name'");
      if ($sql->num_rows) {
      $tpl_source = $sql->record['tpl_source'];
      return true;
      } else {
      return false;
      }
      }

      function db_get_timestamp($tpl_name, &$tpl_timestamp, &$smarty_obj)
      {

      // Datenbankabfrage um '$tpl_timestamp' zuzuweisen
      $sql = new SQL;
      $sql->query("select tpl_timestamp
      from my_table
      where tpl_name='$tpl_name'");
      if ($sql->num_rows) {
      $tpl_timestamp = $sql->record['tpl_timestamp'];
      return true;
      } else {
      return false;
      }
      }

      function db_get_secure($tpl_name, &$smarty_obj)
      {

      // angenommen alle Templates sind sicher
      return true;
      }

      function db_get_trusted($tpl_name, &$smarty_obj)
      {

      // wird für Templates nicht verwendet
      }


      // Ressourcen-Typ 'db:' registrieren
      $smarty->register_resource("db", array("db_get_template",
      "db_get_timestamp",
      "db_get_secure",
      "db_get_trusted"));


      // Ressource im PHP-Skript verwenden
```

```
$smarty->display("db:index.tpl");


{* Ressource in einem Smarty Template verwenden *}
{include file="db:/extras/navigation.tpl"}
```

Standard Template-Handler

Sie können eine Funktion definieren, die aufgerufen wird, wenn ein Template nicht aus der angegeben Ressource geladen werden konnte. Dies ist z. B. nützlich, wenn Sie fehlende Templates on-the-fly generieren wollen.

Beispiel 15.10. Standard Template-Handler verwenden

```
<?php

// fügen Sie folgende Zeilen in Ihre Applikation ein

function make_template ($resource_type, $resource_name, &$template_source, &$template_timestamp,
{
if( $resource_type == 'file' ) {
if ( ! is_readable ( $resource_name )) {

// erzeuge Template-Datei, gib Inhalte zurück
$template_source = "This is a new template.";
$template_timestamp = time();
$smarty_obj->_write_file($resource_name, $template_source);
return true;
}
} else {

// keine Datei
return false;
}
}


// Standard Handler definieren
$smarty->default_template_handler_func = 'make_template';
?>
```

Kapitel 16. Smarty durch Plugins erweitern

Inhaltsverzeichnis

In Version 2.0 wurde die Plugin-Architektur eingeführt, welche für fast alle anpassbaren Funktionalitäten verwendet wird. Unter anderem:

- Funktionen
- Modifikatoren
- Block-Funktionen
- Compiler-Funktionen
- 'pre'-Filter
- 'post'-Filter
- Ausgabefilter
- Ressourcen
- Inserts

Für die Abwärtskompatibilität wurden das register_* API zur Funktions-Registrierung beibehalten. Haben Sie früher nicht die API-Funktionen benutzt, sondern die Klassen-Variablen `$custom_funcs`, `$custom_mods` und andere direkt geändert, müssen Sie Ihre Skripte so anpassen, dass diese das API verwenden. Oder sie implementieren die Funktionalitäten alternativ mit Plugins.

Wie Plugins funktionieren

Plugins werden immer erst bei Bedarf geladen. Nur die im Template verwendeten Funktionen, Ressourcen, Variablen-Modifikatoren, etc. werden geladen. Des weiteren wird jedes Plugin nur einmal geladen, selbst wenn mehrere Smarty-Instanzen im selben Request erzeugt werden.

'pre'/'post'-Filter machen die Ausnahme. Da sie in den Templates nicht direkt erwähnt werden, müssen sie zu Beginn der Ausführung explizit via API geladen oder registriert werden. Die Reihenfolge der Anwendung mehrerer Filter desselben Typs entspricht der Reihenfolge in der sie geladen/registriert wurden.

Die plugins directory Variable kann eine Zeichenkette, oder ein Array mit Verzeichnisnamen sein. Um einen Plugin zu installieren können Sie ihn einfach in einem der Verzeichnisse ablegen.

Namenskonvention

Plugin-Dateien müssen einer klaren Namenskonvention gehorchen, um von Smarty erkannt zu werden.

Die Plugin-Dateien müssen wie folgt benannt werden:

```
type.name.php
```

Wobei `Typ` einen der folgenden Werte haben kann:

- function
- modifier
- block
- compiler
- prefilter
- postfilter
- outputfilter
- resource
- insert

und `Name` ein erlaubter Identifikator (bestehend aus Buchstaben, Zahlen und Unterstrichen) ist.

Ein paar Beispiele: `function.html_select_date.php`, `resource.db.php`, `modifier.spacify.php`.

Die Plugin-Funktion innerhalb das Plugin-Datei muss wie folgt benannt werden:

smarty_*type*, _*name*()

`type` und `name` haben die selbe Bedeutung wie bei den Plugin-Dateien.

Smarty gibt Fehlermeldungen aus, falls ein aufgerufenes Plugin nicht existiert, oder eine Datei mit falscher Namensgebung im Verzeichnis gefunden wurde.

Plugins schreiben

Plugins können von Smarty automatisch geladen oder zur Laufzeit dynamisch mit den register_* API-Funktionen registriert werden. Um registrierte Plugins wieder zu entfernen, können die unregister_* API-Funktionen verwendet werden.

Bei Plugins, die zur Laufzeit geladen werden, müssen keine Namenskonventionen beachtet werden.

Wenn ein Plugin auf die Funktionalität eines anderen Plugins angewiesen ist (wie dies bei manchen Smarty Standard-Plugins der Fall ist), sollte folgender Weg gewählt werden, um das benötigte Plugin zu laden:

```
<?php
require_once $smarty->_get_plugin_filepath('function', 'html_options');
?>
```

Das Smarty Objekt wird jedem Plugin immer als letzter Parameter übergeben (ausser bei Variablen-Modifikatoren und bei Blücken wird *&$repeat* nach dem Smarty Objekt übergeben um Rückwärtskompatibel zu bleiben).

Template-Funktionen

```
void smarty_function_name()($params, &$smarty);
array $params;
object &$smarty;
```

Alle einer Funktion übergebenen Parameter werden in der Variable *$params* als assoziatives Array abgelegt. Sie können auf

diese Werte entweder direkt mit `$params['start']` zugreifen oder sie mit `extract($params)` in die Symbol-Tabelle importieren.

Die Ausgabe der Funktion wird verwendet, um das Funktions-Tag im Template (**fetch()** Funktion, zum Beispiel) zu ersetzen. Alternativ kann sie auch etwas tun, ohne eine Ausgabe zurückzuliefern (**assign()** Funktion, zum Beispiel).

Falls die Funktion dem Template Variablen zuweisen oder auf eine andere Smarty-Funktionalität zugreifen möchte, kann dazu das übergebene *$smarty* Objekt verwendet werden.

Sehen Sie dazu: register_function(), unregister_function().

Beispiel 16.1. Funktionsplugin mit Ausgabe

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.eightball.php
 * Type:     function
 * Name:     eightball
 * Purpose:  outputs a random magic answer
 * -------------------------------------------------------------
 */
function smarty_function_eightball($params, &$smarty)
{
    $answers = array('Yes',
                     'No',
                     'No way',
                     'Outlook not so good',
                     'Ask again soon',
                     'Maybe in your reality');

    $result = array_rand($answers);
    return $answers[$result];
}
?>
```

Es kann im Template wie folgt angewendet werden:

```
Question: Will we ever have time travel?
 Answer: {eightball}.
```

Beispiel 16.2. Funktionsplugin ohne Ausgabe

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.assign.php
 * Type:     function
 * Name:     assign
 * Purpose:  assign a value to a template variable
 * -------------------------------------------------------------
 */
function smarty_function_assign($params, &$smarty)
{
    if (empty($params['var'])) {
        $smarty->trigger_error("assign: missing 'var' parameter");
        return;
    }
```

```
    if (!in_array('value', array_keys($params))) {
        $smarty->trigger_error("assign: missing 'value' parameter");
        return;
    }

    $smarty->assign($params['var'], $params['value']);
}
?>
```

Variablen-Modifikatoren

Variablen-Modifikatoren sind kleine Funktionen, die auf eine Variable angewendet werden, bevor sie ausgegeben oder weiterverwendet wird. Variablen-Modifikatoren können aneinadergereiht werden.

```
mixed smarty_modifier_name()($value, $param1);
mixed $value;
[mixed $param1, ...];
```

Der erste an das Modifikator-Plugin übergebene Parameter ist der Wert mit welchem er arbeiten soll. Die restlichen Parameter sind optional und hängen von den durchzuführenden Operationen ab.

Der Modifikator muss das Resultat seiner Verarbeitung zurückgeben.

Sehen Sie dazu: register_modifier(), unregister_modifier().

Beispiel 16.3. Einfaches Modifikator-Plugin

Dieses Plugin dient als Alias einer PHP-Funktion und erwartet keine zusätzlichen Parameter.

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.capitalize.php
 * Type:     modifier
 * Name:     capitalize
 * Purpose:  capitalize words in the string
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
?>
```

Beispiel 16.4. Komplexes Modifikator-Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.truncate.php
 * Type:     modifier
 * Name:     truncate
 * Purpose:  Truncate a string to a certain length if necessary,
 *           optionally splitting in the middle of a word, and
 *           appending the $etc string.
```

```
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
                                  $break_words = false)
{
    if ($length == 0)
        return '';

    if (strlen($string) > $length) {
        $length -= strlen($etc);
        $fragment = substr($string, 0, $length+1);
        if ($break_words)
            $fragment = substr($fragment, 0, -1);
        else
            $fragment = preg_replace('/\s+(\S+)?$/', '', $fragment);
        return $fragment.$etc;
    } else
        return $string;
}
?>
```

Block-Funktionen

```
void smarty_function_name()($params, $content, &$smarty, &$repeat);
array $params;
mixed $content;
object &$smarty;
boolean &$repeat;
```

Block-Funktionen sind Funktionen, die in der Form {func} .. {/func} notiert werden. Mit anderen Worten umschliessen sie einen Template-Abschnitt und arbeiten danach auf dessen Inhalt. Eine Block-Funktion {func} .. {/func} kann nicht mir einer gleichnamigen Template-Funktion {func} überschrieben werden.

Ihre Funktions-Implementation wird von Smarty zweimal aufgerufen: einmal für das öffnende und einmal für das schliessende Tag. (konsultieren Sie den Abschnitt zu `&$repeat` um zu erfahren wie Sie dies ändern können.)

Nur das Öffnungs-Tag kann Attribute enthalten. Alle so übergebenen Attribute werden als assoziatives Array *$params* der Template-Funktion übergeben. Sie können auf die Werte entweder direkt mit `$params['start']` zugreifen oder sie mit `extract($params)` in die Symbol-Tabelle importieren. Die Attribute aus dem Öffnungs-Tag stehen auch beim Aufruf für das schliessende Tag zur Verfügung.

Der Inhalt der *$content* Variable hängt davon ab, ob die Funktion für das öffnende Tag oder für das schliessende Tag aufgerufen wird. Für das öffnende Tag ist der Wert `null`, für das schliessende Tag ist es der Inhalt des Template-Abschnitts. Achtung: Der Template-Abschnitt den Sie erhalten, wurde bereits von Smarty bearbeitet. Sie erhalten also die Template-Ausgabe, nicht den Template-Quelltext.

Der Parameter *&$repeat* wird als Referenz übergeben und kontrolliert wie oft ein Block dargestellt werden soll. Standardwert von *$repeat* ist beim ersten Aufruf (für das öffnende Tag) `true`, danach immer `false`. Jedes Mal wenn eine Funktion für *&$repeat* TRUE zurück gibt, wird der Inhalt zwischen {func} .. {/func} erneut mit dem veränderten Inhalt als *$content* Parameter aufgerufen.

Wenn Sie verschachtelte Block-Funktionen haben, können Sie die Eltern-Block-Funktion mit der `$smarty->_tag_stack` Variable herausfinden. Lassen Sie sich ihren Inhalt mit 'var_dump()' ausgeben. Die Struktur sollte selbsterklärend sein.

Sehen Sie dazu: register_block(), unregister_block().

Beispiel 16.5. Block-Funktionen

```
<?php
/*
```

```
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     block.translate.php
 * Type:     block
 * Name:     translate
 * Purpose:  translate a block of text
 * -------------------------------------------------------------
 */
function smarty_block_translate($params, $content, &$smarty, &$repeat)
{
    if (isset($content)) {
        $lang = $params['lang'];
                // den $content irgendwie intelligent &uuml;bersetzen
        return $translation;
    }
}
?>
```

Compiler-Funktionen

Compiler-Funktionen werden während der Kompilierung des Template aufgerufen. Das ist nützlich, um PHP-Code oder zeitkritische statische Inhalte in ein Template einzufügen. Sind eine Compiler-Funktion und eine eigene Funktion unter dem selben Namen registriert, wird die Compiler-Funktion ausgeführt.

```
mixed smarty_compiler_name()($tag_arg, &$smarty);
string $tag_arg;
object &$smarty;
```

Die Compiler-Funktion erhält zwei Parameter: die Tag-Argument Zeichenkette - also alles ab dem Funktionsnamen bis zum schliessenden Trennzeichen - und das Smarty Objekt. Gibt den PHP-Code zurück, der in das Template eingefügt werden soll.

Sehen Sie dazu: register_compiler_function(), unregister_compiler_function().

Beispiel 16.6. Einfache Compiler-Funktionen

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     compiler.tplheader.php
 * Type:     compiler
 * Name:     tplheader
 * Purpose:  Output header containing the source file name and
 *           the time it was compiled.
 * -------------------------------------------------------------
 */
function smarty_compiler_tplheader($tag_arg, &$smarty)
{
    return "\necho '" . $smarty->_current_file . " compiled at " . date('Y-m-d H:M'). "';";
}
?>
```

Diese Funktion kann aus dem Template wie folgt aufgerufen werden:

```
{* diese Funktion wird nur zum Kompilier-Zeitpunkt ausgeführt *}
      {tplheader}
```

Der resultierende PHP-Code würde ungefähr so aussehen:

```
<?php
```

```
echo 'index.tpl compiled at 2002-02-20 20:02';
?>
```

'pre'/'post'-Filter

'pre'-Filter und 'post'-Filter folgen demselben Konzept. Der einzige Unterschied ist der Zeitpunkt der Ausführung.

```
string smarty_prefilter_name()($source, &$smarty);
string $source;
object &$smarty;
```

'pre'-Filter werden verwendet, um die Quellen eines Templates direkt vor der Kompilierung zu verarbeiten. Als erster Parameter wird die Template-Quelle, die möglicherweise bereits durch eine weiteren 'pre'-Filter bearbeitet wurden, übergeben. Das Plugin muss den resultierenden Wert zurückgeben. Achtung: Diese Werte werden nicht gespeichert und nur zum Kompilier-Zeitpunkt verwendet.

```
string smarty_postfilter_name()($compiled, &$smarty);
string $compiled;
object &$smarty;
```

'post'-Filter werden auf die kompilierte Ausgabe direkt vor dem Speichern angewendet. Als erster Parameter wird der kompilierte Template-Code übergeben, der möglicherweise zuvor von anderen 'post'-Filtern bearbeitet wurde. Das Plugin muss den veränderten Template-Code zurückgeben.

Beispiel 16.7. 'pre'-Filter Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     prefilter.pre01.php
 * Type:     prefilter
 * Name:     pre01
 * Purpose:  Convert html tags to be lowercase.
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
     return preg_replace('!<(\w+)[^>]+>!e', 'strtolower("$1")', $source);
 }
?>
```

Beispiel 16.8. 'post'-Filter Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     postfilter.post01.php
 * Type:     postfilter
 * Name:     post01
 * Purpose:  Output code that lists all current template vars.
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
     $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" . $compiled;
     return $compiled;
```

```
 }
?>
```

Ausgabefilter

Ausgabefilter werden auf das Template direkt vor der Ausgabe angewendet, nachdem es geladen und ausgeführt wurde.

```
string smarty_outputfilter_name()($template_output, &$smarty);
string $template_output;
object &$smarty;
```

Als erster Parameter wird die Template-Ausgabe übergeben, welche verarbeitet werden soll und als zweiter Parameter das Smarty-Objekt. Das Plugin muss danach die verarbeitete Template-Ausgabe zurückgeben.

Beispiel 16.9. Ausgabefilter Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     outputfilter.protect_email.php
 * Type:     outputfilter
 * Name:     protect_email
 * Purpose:  Converts @ sign in email addresses to %40 as
 *           a simple protection against spambots
 * -------------------------------------------------------------
 */
 function smarty_outputfilter_protect_email($output, &$smarty)
 {
     return preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                         '$1%40$2', $output);
 }
?>
```

Ressourcen

Ressourcen-Plugins stellen einen generischen Weg dar, um Smarty mit Template-Quellen oder PHP-Skripten zu versorgen. Einige Beispiele von Ressourcen: Datenbanken, LDAP, shared Memory, Sockets, usw.

Für jeden Ressource-Typ müssen 4 Funktionen registriert werden. Jede dieser Funktionen erhält die verlangte Ressource als ersten Parameter und das Smarty Objekt als letzten. Die restlichen Parameter hängen von der Funktion ab.

```
bool smarty_resource_name_source()($rsrc_name, &$source, &$smarty);
string $rsrc_name;
string &$source;
object &$smarty;
bool smarty_resource_name_timestamp()($rsrc_name, &$timestamp, &$smarty);
string $rsrc_name;
int &$timestamp;
object &$smarty;
bool smarty_resource_name_secure()($rsrc_name, &$smarty);
string $rsrc_name;
object &$smarty;
bool smarty_resource_name_trusted()($rsrc_name, &$smarty);
string $rsrc_name;
object &$smarty;
```

Die erste Funktion wird verwendet, um die Ressource zu laden. Der zweite Parameter ist eine Variable, die via Referenz übergeben wird und in der das Resultat gespeichert werden soll. Die Funktion gibt `true` zurück, wenn der Ladevorgang

erfolgreich war - andernfalls `false`.

Die zweite Funktion fragt das letzte Änderungsdatum der angeforderten Ressource (als Unix-Timestamp) ab. Der zweite Parameter ist die Variable, welche via Referenz übergeben wird und in der das Resultat gespeichert werden soll. Gibt `true` zurück, wenn das Änderungsdatum ermittelt werden konnte und `false` wenn nicht.

Die dritte Funktion gibt `true` oder `false` zurück, je nachdem ob die angeforderte Ressource als sicher bezeichnet wird oder nicht. Diese Funktion wird nur für Template-Ressourcen verwendet, sollte aber in jedem Fall definiert werden.

Die vierte Funktion gibt `true` oder `false` zurück, je nachdem ob die angeforderte Ressource als vertrauenswürdig angesehen wird oder nicht. Diese Funktion wird nur verwendet, wenn PHP-Skripte via **include_php** oder **insert** eingebunden werden sollen und ein 'src' Attribut übergeben wurde. Die Funktion sollte aber in jedem Fall definiert werden.

Sehen Sie dazu: register_resource(), unregister_resource().

Beispiel 16.10. Ressourcen Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     resource.db.php
 * Type:     resource
 * Name:     db
 * Purpose:  Fetches templates from a database
 * -------------------------------------------------------------
 */
function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
    // do database call here to fetch your template,
    // populating $tpl_source
    $sql = new SQL;
    $sql->query("select tpl_source
                   from my_table
                  where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_source = $sql->record['tpl_source'];
        return true;
    } else {
        return false;
    }
}

function smarty_resource_db_timestamp($tpl_name, &$tpl_timestamp, &$smarty)
{
    // do database call here to populate $tpl_timestamp.
    $sql = new SQL;
    $sql->query("select tpl_timestamp
                   from my_table
                  where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_timestamp = $sql->record['tpl_timestamp'];
        return true;
    } else {
        return false;
    }
}

function smarty_resource_db_secure($tpl_name, &$smarty)
{
    // assume all templates are secure
    return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
    // not used for templates
```

```
}
?>
```

Inserts

Insert-Plugins werden verwendet, um Funktionen zu implementieren, die via **insert** aufgerufen werden.

```
string smarty_insert_name()($params, &$smarty);
array $params;
object &$smarty;
```

Als erster Parameter wird der Funktion ein assoziatives Array aller Attribute übergeben, die im Insert-Tag notiert wurden. Sie können auf diese Werte entweder direkt mit `$params['start']` zugreifen oder sie mit `extract($params)` importieren.

Als Rückgabewert muss das Resultat der Ausführung geliefert werden, das danach den Platz des **insert**-Tags im Template einnimmt.

Beispiel 16.11. Insert-Plugin

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     insert.time.php
 * Type:     time
 * Name:     time
 * Purpose:  Inserts current date/time according to format
 * -------------------------------------------------------------
 */
function smarty_insert_time($params, &$smarty)
{
    if (empty($params['format'])) {
        $smarty->trigger_error("insert time: missing 'format' parameter");
        return;
    }

    $datetime = strftime($params['format']);
    return $datetime;
}
?>
```

Teil IV. Anhänge

Inhaltsverzeichnis

Kapitel 17. Problemlösung

Inhaltsverzeichnis

Smarty/PHP Fehler

Smarty kann verschiedene Fehler-Typen, wie fehlende Tag-Attribute oder syntaktisch falsche Variablen-Namen abfangen. Wenn dies geschieht, wird Ihnen eine Fehlermeldung ausgegeben. Beispiel:

Beispiel 17.1. Smarty Fehler

```
Warning: Smarty: [in index.tpl line 4]: syntax error: unknown tag - '%blah'
        in /path/to/smarty/Smarty.class.php on line 1041

Fatal error: Smarty: [in index.tpl line 28]: syntax error: missing section name
        in /path/to/smarty/Smarty.class.php on line 1041</programlisting>
```

In der ersten Zeile zeigt Smarty den Template-Namen, die Zeilennummer und den Fehler an. Darauf folgt die betroffene Zeile in der Smarty Klasse welche den Fehler erzeugt hat.

Es gibt gewisse Fehlerkonditionen, die Smarty nicht abfangen kann (bsp: fehlende End-Tags). Diese Fehler resultieren jedoch normalerweise in einem PHP-'compile-time' Fehler.

Beispiel 17.2. PHP Syntaxfehler

```
Parse error: parse error in /path/to/smarty/templates_c/index.tpl.php on line 75</programlisting>
```

Wenn ein PHP Syntaxfehler auftritt, wird Ihnen die Zeilennummer des betroffenen PHP Skriptes ausgegeben, nicht die des Templates. Normalerweise können Sie jedoch das Template anschauen um den Fehler zu lokalisieren. Schauen sie insbesondere auf Folgendes: fehlende End-Tags in einer {if}{/if} Anweisung oder in einer {section}{/section} und die Logik eines {if} Blocks. Falls Sie den Fehler so nicht finden, können Sie auch das kompilierte Skript öffnen und zu der betreffenden Zeilennummer springen um herauszufinden welcher Teil des Templates den Fehler enthält.

Kapitel 18. Tips & Tricks

Inhaltsverzeichnis

Handhabung unangewiesener Variablen

Manchmal möchten Sie vielleicht, dass anstatt einer Leerstelle ein Standardwert ausgegeben wird - zum Beispiel um im Tabellenhintergrund " " auszugeben, damit er korrekt angezeigt wird. Damit dafür keine {if} Anweisung verwendet werden muss, gibt es in Smarty eine Abkürzung: die Verwendung des *default* Variablen-Modifikators.

Beispiel 18.1. " " ausgeben wenn eine Variable nicht zugewiesen ist

```
{* kompliziert *}

{if $titel eq ""}
&amp;nbsp;
{else}
{$titel}
{/if}


{* einfach *}

{$titel|default:"&amp;nbsp;"}
```

Siehe auch default (Standardwert) und Handhabung von Standardwerten.

Handhabung von Standardwerten

Wenn eine Variable in einem Template häufig zum Einsatz kommt, kann es ein bisschen störend wirken, den *default*-Modifikator jedes mal anzuwenden. Sie können dies umgehen, indem Sie der Variable mit der {assign} Funktion einen Standardwert zuweisen.

Beispiel 18.2. Zuweisen des Standardwertes einer Variable

```
{* schreiben sie dieses statement an den Anfang des Templates *}
{assign var="titel" value=$titel|default:"kein Titel"}

{* falls 'titel' bei der Anweisung leer war, enthält es nun den Wert
   'kein Titel' wenn Sie es ausgeben *}
```

```
{$titel}
```

Siehe auch default (Standardwert) und Handhabung nicht zugewiesener Variablen.

Variablen an eingebundene Templates weitergeben

Wenn die Mehrzahl Ihrer Templates den gleichen Header und Footer verwenden, lagert man diese meist in eigene Templates aus und bindet diese mit{include} ein. Was geschieht aber wenn der Header einen seitenspezifischen Titel haben soll? Smarty bietet die Möglichkeit, dem eingebundenen Template, Variablen zu übergeben.

Beispiel 18.3. Die Titel-Variable dem Header-Template zuweisen

`mainpage.tpl`

```
{include file="header.tpl" title="Hauptseite"}
{* template body hier *}
{include file="footer.tpl"}
```

`archives.tpl`

```
{config_load file="archiv.conf"}
{include file="header.tpl" title=#archivSeiteTitel#}
{* template body hier *}
{include file="footer.tpl"}
```

`header.tpl`

```
<html>
<head>
<title>{$title|default:"Nachrichten"}</title>
</head>
<body>
```

`footer.tpl`

```
</BODY>
</HTML>
```

Sobald die erste Seite geparsed wird, wird der Titel 'Erste Seite' dem `header.tpl` übergeben und fortan als Titel verwendet. Wenn die Archivseite ausgegeben wird, wird der Titel 'Archive' ausgegeben. Wie Sie sehen können, wird der Wert dafür aus der Datei `'archiv_page.conf'` geladen und nicht von einem übergebenen Wert. Der Standardwert 'Nachrichten' wird verwendet, wenn die '$titel' leer ist. Erneut wird dafür der default-Modifikator angewandt.

Zeitangaben

Um dem Template Designer höchstmögliche Kontrolle über die Ausgabe von Zeitangaben/Daten zu ermöglichen, ist es empfehlenswert Daten immer als Timestamp zu übergeben. Der Designer kann danach die Funktion date_format für die Formatierung verwenden.

Bemerkung: Seit Smarty 1.4.0 ist es möglich jede Timestamp zu übergeben, welche mit strtotime() ausgewertet werden kann. Dazu gehören Unix-Timestamps und MySQL-Timestamps.

Beispiel 18.4. Die Verwendung von date_format

```
{$startDate|date_format}
```

AUSGABE:

```
Jan 4, 2001
```

```
{$startDatum|date_format:"%Y/%m/%d"}
```

AUSGABE:

```
2001/01/04
```

```
{if $datum1 < $datum2}
...
{/if}
```

Falls {html_select_date} in einem Template verwendet wird, hat der Programmierer die Möglichkeit den Wert wieder in ein Timestamp-Format zu ändern. Dies kann zum Beispiel wie folgt gemacht werden:

Beispiel 18.5. Formular Datum-Elemente nach Timestamp konvertieren

```
<?php

// hierbei wird davon ausgegangen, dass Ihre Formular Elemente wie folgt benannt sind
// startDate_Day, startDate_Month, startDate_Year

$startDate = makeTimeStamp($startDate_Year,$startDate_Month,$startDate_Day);

function makeTimeStamp($year="",$month="",$day="")
{
    if(empty($year)) {
        $year = strftime("%Y");
    }
    if(empty($month)) {
        $month = strftime("%m");
    }
    if(empty($day)) {
        $day = strftime("%d");
    }
    return mktime(0, 0, 0, $month, $day, $year);
}
```

Siehe auch {html_select_date}, {html_select_time}, date_format und $smarty.now,

WAP/WML

WAP/WML Templates verlangen, dass ein Content-Type Header im Template angegeben wird. Der einfachste Weg um dies zu tun, wäre, eine Funktion zu schreiben, welche den Header ausgibt. Falls sie den Caching Mechanismus verwenden, sollten Sie auf das 'insert'-Tag zurückgreifen ('insert'-Tags werden nicht gecached), um ein optimales Ergebnis zu erzielen. Achten Sie darauf, dass vor der Ausgabe des Headers keine Daten an den Client gesendet werden, da die gesendeten Header-Daten ansonsten von Client verworfen werden.

Beispiel 18.6. Die verwendung von 'insert' um einen WML Content-Type header zu senden

```
<?php

// stellen Sie sicher, dass Apache mit .wml Dateien umgehen kann!
// schreiben Sie folgende Funktion in Ihrer Applikation, oder in Smarty.addons.php
function insert_header($params)
{
    // folgende Funktion erwartet ein $inhalt argument
    if (empty($params['inhalt'])) {
        return;
    }
    header($params['inhalt']);
    return;
}

?>
```

Ihr Template *muss* danach wie folgt beginnen:

```
{insert name=header inhalt="Content-Type: text/vnd.wap.wml"}

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- neues wml deck -->
<wml>
 <!-- erste karte -->
 <card>
  <do type="accept">
   <go href="#zwei"/>
  </do>
  <p>
   Welcome to WAP with Smarty!
   Willkommen bei WAP mit Smarty!
   OK klicken um weiterzugehen...
  </p>
 </card>
 <!-- zweite karte -->
 <card id="zwei">
  <p>
   Einfach, oder?
  </p>
 </card>
</wml>
```

Template/Script Komponenten

Normalerweise werden Variablen dem Template wie folgt zugewiesen: In Ihrer PHP-Applikation werden die Variablen zusammengestellt (zum Beispiel mit Datenbankabfragen). Danach kreieren Sie eine Instanz von Smarty, weisen die Variablen mit assign() zu und geben das Template mit display() aus. Wenn wir also zum Beispiel einen Börsenticker in unserem Template haben, stellen wir die Kursinformationen in unserer Anwendung zusammen, weisen Sie dem Template zu und geben es aus. Wäre es jedoch nicht nett diesen Börsenticker einfach in ein Template einer anderen Applikation einbinden zu können ohne deren Programmcode zu ändern?

Sie können PHP-Code mit {php}{/php} in Ihre Templates einbetten. So können Sie Templates erstellen, welche die Datenstrukturen zur Anweisung der eigenen Variablen enthalten. Durch die Bindung von Template und Logik entsteht so eine eigenständig lauffähige Komponente.

Beispiel 18.7. Template/Script Komponenten

`function.load_ticker.php` - Diese Datei gehört ins $plugins directory

```
<?php

// setup our function for fetching stock data
function fetch_ticker($symbol)
{
    // put logic here that fetches $ticker_info
    // from some ticker resource
    return $ticker_info;
}

function smarty_function_load_ticker($params, $smarty)
{
    // call the function
    $ticker_info = fetch_ticker($params['symbol']);

    // assign template variable
    $smarty->assign($params['assign'], $ticker_info);
}
?>
```

`index.tpl`

```
{load_ticker symbol="YHOO" assign="ticker"}

Stock Name: {$ticker.name} Stock Price: {$ticker.price}
```

Verschleierung von E-mail Adressen

Haben Sie sich auch schon gewundert, wie Ihre E-mail Adresse auf so viele Spam-Mailinglisten kommt? Ein Weg, wie Spammer E-mail Adressen sammeln, ist über Webseiten. Um dieses Problem zu bekämpfen, können sie den 'mailto'-Plugin verwenden. Er ändert die Zeichenfolge mit Javascript so, dass sie im HTML Quellcode nicht lesbar ist, jedoch von jedem Browser wieder zusammengesetzt werden kann. Den {mailto}-Plugin gibt es im Smarty-Repository auf http://smarty.php.net. Laden sie den Plugin herunter und speichern Sie ihn im 'plugins' Verzeichnis.

Beispiel 18.8. Beispiel von verschleierung von E-mail Adressen

```
{* in index.tpl *}

Anfragen bitte an
{mailto address=$EmailAddress encode="javascript" subject="Hallo"}
senden
```

> **Technische Details:** Die Codierung mit Javascript ist nicht sehr sicher, da ein möglicher Spammer die Decodierung in sein Sammelprogramm einbauen könnte. Es wird jedoch damit gerechnet, dass, da Aufwand und Ertrag sich nicht decken, dies nicht oft der Fall ist.

Siehe auch escape und {mailto}.

Kapitel 19. Weiterführende Informationen

Smarty's Homepage erreicht man unter http://smarty.php.net/. Sie können der Smarty Mailingliste beitreten in dem sie ein E-mail an smarty-general-subscribe@lists.php.net. Das Archiv der Liste ist hier http:/ / marc.theaimsgroup.com/ ?l=smarty-general&r=1&w=2 einsehbar.

Kapitel 20. BUGS

Bitte konsultieren Sie die Datei `BUGS` welche mit Smarty ausgeliefert wird, oder die Webseite.