

# Inhaltsverzeichnis

<b>Allgemeines</b> .....	<b>1</b>
<u>Einführung</u> .....	1
<u>Trennung von Design und Inhalt</u> .....	2
<b>EGOTEC Templates</b> .....	<b>3</b>
<u>Die Verzeichnisstruktur</u> .....	3
<u>Das Haupttemplate</u> .....	4
<u>Beispiel</u> .....	4
<u>Platzhalter für Seitentyp-Templates</u> .....	4
<u>Beispiel (index.html)</u> .....	5
<u>Beispiel (HEAD-Bereich)</u> .....	6
<u>Der Seitentitel</u> .....	7
<u>Die Meta-Angaben</u> .....	7
<u>Favicons</u> .....	9
<u>Styles</u> .....	9
<u>Typenspezifisches Template einbinden</u> .....	9
<u>Style-Hinweis</u> .....	10
<u>Seitenspezifische Templates</u> .....	11
<u>Stylesheets</u> .....	12
<u>style.css</u> .....	12
<b>Suchtreffer hervorheben</b> .....	<b>13</b>
<u>span.css</u> .....	13
<u>table.css</u> .....	14
<u>Beispiel table.css</u> .....	14
<u>img.css</u> .....	14
<u>Beispiel img.css</u> .....	14
<u>CSS-Hinweise</u> .....	14
<u>Allgemeines</u> .....	15
<u>Wiederholende Elemente</u> .....	15
<u>Im eigentlichen Inhalt</u> .....	15
<u>Auf bestimmte Tags beschränken</u> .....	15
<b>Weitere Hinweise</b> .....	<b>16</b>
<u>Templates für verschiedene Medien</u> .....	17
<u>Templates für Druckansichten</u> .....	17
<u>Das Haupttemplate</u> .....	17
<u>Seitenspezifische Templates</u> .....	17
<u>Templates für die PDF-Generierung</u> .....	17
<b>Das Haupttemplate</b> .....	<b>18</b>
<b>Seitenspezifische Templates</b> .....	<b>19</b>
<b>Mögliche Endungen</b> .....	<b>20</b>
<u>html doc</u> .....	20
<u>html2ps</u> .....	20
<u>Templates für Adobe InDesign</u> .....	20
<u>Templates zur Bilddarstellung</u> .....	23
<u>Mehrsprachigkeit</u> .....	25
<u>Aufbau einer translation.ini</u> .....	25
<b>Smarty</b> .....	<b>26</b>
<u>Smarty Kommentare</u> .....	26
<u>Beispiel</u> .....	26

# Inhaltsverzeichnis

## Smarty

<u>Smarty Platzhalter</u> .....	27
<u>Beispiel für einen Platzhalter</u> .....	27
<u>url_dir</u> .....	27
<u>page</u> .....	27
<u>site</u> .....	28
<u>Einige Ausgabemöglichkeiten</u> .....	28
<u>Smarty Blöcke</u> .....	29
<u>{if}{elseif}{else}{/if}</u> .....	29
<u>Beispiel</u> .....	29
<u>{foreach}, {foreachelse}</u> .....	29
<u>Beispiel</u> .....	29
<u>{html_code}{/html_code}</u> .....	30
<u>{strip}{/strip}</u> .....	30
<u>Beispiel</u> .....	30
<u>Ausgabe</u> .....	31
<u>{t}{/t}</u> .....	31
<u>{literal}{/literal}</u> .....	31
<u>{cache}{/cache}</u> .....	32
<u>Beispiel</u> .....	32
<u>Smarty Funktionen</u> .....	33
<u>{admin_url}</u> .....	33
<u>Beispiel</u> .....	33
<u>{assign}</u> .....	33
<u>Beispiel</u> .....	33
<u>Ausgabe</u> .....	33
<u>{check_captcha}</u> .....	33
<u>Beispiel 1</u> .....	34
<u>Beispiel 2</u> .....	34
<u>{config_load}</u> .....	34
<u>Beispiel</u> .....	34
<u>{count_pages}</u> .....	34
<u>Beispiel</u> .....	34
<u>{debug}</u> .....	35
<u>Beispiel</u> .....	35
<u>{extra_push}</u> .....	35
<u>Beispiel</u> .....	35
<u>{file_exists}</u> .....	36
<u>Beispiel</u> .....	36
<u>{get_captcha}</u> .....	36
<u>Beispiel 1</u> .....	36
<u>Beispiel 2</u> .....	36
<u>Beispiel 3</u> .....	36
<u>{get_children}</u> .....	37
<u>Beispiele</u> .....	37
<u>{get_descendants}</u> .....	37
<u>Beispiele</u> .....	37
<u>{get_keywords}</u> .....	38
<u>Beispiele</u> .....	38
<u>{get_pages}</u> .....	38
<u>Beispiel</u> .....	38
<u>{get_parents}</u> .....	39
<u>Beispiel</u> .....	39
<u>{get_path}</u> .....	39
<u>Beispiele</u> .....	39

# Inhaltsverzeichnis

## Smarty

<u>{get_siblings}</u> .....	39
<u>Beispiel</u> .....	39
<u>{get_sibling}</u> .....	40
<u>Beispiele</u> .....	40
<u>{get_user}</u> .....	40
<u>Beispiel</u> .....	40
<u>{help}</u> .....	40
<u>Beispiel</u> .....	40
<u>{icon_by_extension}</u> .....	41
<u>Beispiel</u> .....	41
<u>{include_module_files}</u> .....	41
<u>Beispiel</u> .....	41
<u>{include}</u> .....	42
<u>Beispiel</u> .....	42
<u>Typenspezifische Templates einbinden</u> .....	42
<u>{input}</u> .....	42
<u>Beispiel</u> .....	42
<u>Die HTML-Datei</u> .....	43
<u>Bereich 1 Smarty+HTML</u> .....	43
<u>Bereich 2 Javascipt</u> .....	43
<u>{in_path}</u> .....	47
<u>Beispiel</u> .....	47
<u>{mailto}</u> .....	48
<u>{mail_send}</u> .....	48
<u>Beispiel</u> .....	48
<u>{mail_valid_email}</u> .....	48
<u>Beispiel</u> .....	48
<u>{navigation}</u> .....	48
<u>{nav_url}</u> .....	49
<u>Beispiel</u> .....	49
<u>{nav}</u> .....	49
<u>Beispiel</u> .....	49
<u>{new_child}</u> .....	49
<u>Beispiel</u> .....	49
<u>{online_hilfe}</u> .....	50
<u>Beispiel</u> .....	50
<u>{page_exists}</u> .....	50
<u>Beispiel</u> .....	50
<u>{page_url}</u> .....	51
<u>Beispiel</u> .....	51
<u>{print_url}</u> .....	51
<u>Beispiel</u> .....	51
<u>{sort_extra}</u> .....	51
<u>Beispiel</u> .....	51
<u>{style_inactive}</u> .....	52
<u>Beispiel</u> .....	52
<u>{sub_html}</u> .....	52
<u>Beispiel</u> .....	52
<u>Möglicher Anwendungsbereich</u> .....	53
<u>{tag_cloud}</u> .....	53
<u>{update_page}</u> .....	54
<u>Beispiel</u> .....	54
<u>Smarty Modifikatoren</u> .....	55
<u>Beispiel</u> .....	55

# Inhaltsverzeichnis

## **Smarty**

<u>Ausgabe</u> .....	55
<u>max_image_width</u> .....	55
<u>max_image_height</u> .....	59
<u>scale_image</u> .....	60
<u>truncate</u> .....	61
<u>Beispiel</u> .....	61
<u>Ausgabe</u> .....	61
<u>date_format</u> .....	61
<u>Beispiel</u> .....	61
<u>Ausgabe</u> .....	62
<u>Weitere Smarty Modifikatoren</u> .....	62
<u>Eigene Smarty-Plugins erstellen</u> .....	63

## **Vorlagen im Editor**.....64

## **Einen neuen Block erstellen**.....65

<u>Vorschauansicht</u> .....	65
------------------------------	----

## **Tipps und Tricks**.....67

<u>JavaScript</u> .....	67
<u>IE IMG Node kopieren</u> .....	67
<u>IE Bug Flickering Hintergrundbilder</u> .....	67
<u>Variablennamen</u> .....	67
<u>CSS</u> .....	68
<u>Listensymbole positionieren</u> .....	68
<u>YAML</u> .....	69

## **Vertikale und Horizontale Abstände bei Bildern**.....70

## **Beispiele & Vorlagen**.....71

<u>Aufklappbare Navigation</u> .....	71
<u>navigation.html</u> .....	71
<u>more_children.html</u> .....	74
<u>Einbinden der Navigation ins Template</u> .....	76
<u>Zweistufige Navigation</u> .....	77
<u>Popuppildergalerie</u> .....	78
<u>Bread-Crump-Navigation / Pfad</u> .....	79
<u>Rekursive Aufrufe</u> .....	80

# Allgemeines

Dieses Handbuch in erster Linie für **Webdesigner** ausgelegt, daher verzichten wir auf eine Einführung in die Programmierung von HTML-Seiten und setzen ein entsprechendes Basiswissen voraus. Bei Fragen zu (X)HTML, Javascript oder CSS empfehlen wir den Besuch auf z.B. [SelfHTML](#) bzw. [CSS4you](#) oder verweisen auf entsprechende Fachliteratur.

## Einführung

Wie integriere ich ein fertiges Design in das CMS?

### Das Haupttemplate

beinhaltet den kompletten Aufbau und das Design Ihrer Website. Dieses ist im ersten Schritt eine einfache HTML-Datei, die Stylesheets und Bilder verwendet. Es wird bei jedem Aufruf der Seite geladen und bildet quasi das Grundgerüst ihrer Website.

### Die Template-Bereiche

sind variable Bereiche im Haupttemplate wie z.B. linke Navigation, Inhaltsbereich, rechte News-Spalte usw.. Diese Bereiche sollen sich ändern, je nach dem, auf welcher Seite der Besucher sich gerade befindet.

### Die Platzhalter

dienen zum gezielten Platzieren von Informationen in Ihrem Auftritt. Über ein Skript können diese nach Belieben befüllt und auf der Website nach Wunsch positioniert werden.

### Die Seitentypen

bestimmen, wie eine Seite im Frontend aussehen hat. So kann z.B. eine Startseite eine Auflistung von News beinhalten, wobei eine Standardseite nur Titel, Kurzbeschreibung und den eigentlichen Inhalt ausgeben soll. Zwar bleibt das grobe Grundgerüst des Auftritts fest, die einzelnen Bereiche können jedoch unterschiedlich aussehen. Das jeweilige Erscheinungsbild wird in sogenannten "Templates" einmal pro Seitentyp (Vorlage) festgelegt.

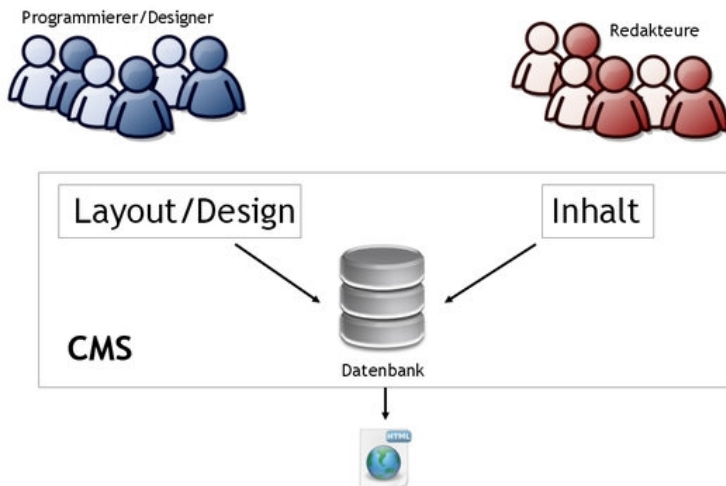
### Zusammengefasst:

- Sie erstellen einen fertigen Auftritt in purem (X)HTML
- definieren variable Bereiche in Ihrem Auftritt, die sich ändern dürfen (z.B. eine linke Navigation, der Inhaltsbereich usw... ) und
- setzen Platzhalter für diese Bereiche ein.

# Trennung von Design und Inhalt

Eine der Stärken vom EGOTEC® CMS ist die klare und schlichte Trennung von Design und Inhalt.

Für jeden Seitentypen kann der Designer ein individuelles Layout (Template) entwerfen; dieses kann anschließend für beliebig viele Seiten verwendet werden. Zum anderen können Redakteure die Inhalte der einzelnen Seiten mit EGOTEC® einpflegen, ohne sich über das Layout Gedanken machen zu müssen. Damit sind für den Redakteur auch keinerlei Programmierkenntnisse erforderlich.



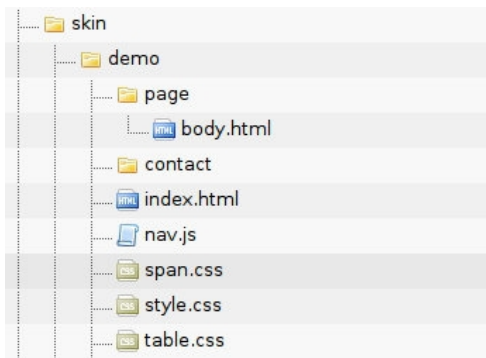
# EGOTEC Templates

## Templates

Templates bestimmen das Erscheinungsbild eines Seitentyps.

## Die Verzeichnisstruktur

Alle Dateien, die das Aussehen der Website betreffen, liegen immer zentral im **skin**-Verzeichnis. Verwendet der Mandant als Design "demo", finden Sie dazugehörige Templates im Verzeichnis skin/demo/.



Die Verzeichnisse "page" und "contact" stehen für entsprechende Seitentypen. Sie beinhalten das seitentypspezifische Template (body.html).

Wie Sie einen neuen Seitentyp erstellen, entnehmen Sie bitte dem Entwickler-Handbuch.

# Das Haupttemplate

Das Haupttemplate liegt in **skin/MANDANT/index.html** und ist das zentrale Grundgerüst der Website, sprich: Das globale Design.

Hier bestimmt der Designer, wie die Webseite aussieht und aufgebaut ist, welche Spalten verfügbar sind und wo welcher Inhalt angezeigt werden sollen.

Das Haupttemplate wird bei jedem Seitenaufruf geladen.

## Beispiel

Logo	Hauptnavigation
Navigation	Inhalt
	Fusszeile

Die blauen Bereiche würden dabei über die *index.html* definiert werden, da sie für alle Seiten mehr oder weniger identisch sind.

Der grüne Bereich wird je nach Seitentyp durch die dazugehörige *body.html* definiert.

Bestandteile der Webseite, die sich auf allen Seiten wiederholen, sollten ebenfalls in die *index.html* eingebunden werden, um so ein einheitliches Design zu gewährleisten. Für ein einheitliches Design sorgt auch eine CSS Style Sheet Datei, die man in der *index.html* einbindet. Wir bevorzugen die Einbindung von Style Sheet Definitionen als Datei, da die so erzeugten HTML-Seiten schlank bleiben.

## Platzhalter für Seitentyp-Templates

Damit ein seitentypspezifisches Template eingebunden werden kann, stellt das System automatisch die Variable `{typeTemplate}` zu Verfügung. Dieser Platzhalter muss im Haupttemplate an gewünschter Stelle positioniert werden.

QuelltextHTML Code:

1. `<div id="inhalt">`
2. `{if $typeTemplate} {* prüfen, ob eine Template vorhanden ist *}`
3. `{include file=$typeTemplate} {* wenn ja, dieses Template einbinden *}`
4. `{else}`
5. `{$page->field.content} {* ansonsten der Inhalt anzeigen *}`
6. `{/if}`
7. `</div>`

```
<div id="inhalt">
  {if $typeTemplate} {* prüfen, ob eine Template vorhanden ist *}
  {include file=$typeTemplate} {* wenn ja, dieses Template einbinden *}
{else}
  {$page->field.content} {* ansonsten der Inhalt anzeigen *}
{/if}
</div>
```

## Beispiel (index.html)

Eine einfache **index.html** könnte wie folgt aussehen:

QuelltextHTML Code:

```
1. <html>
2. <head>
3.   <link rel="shortcut icon" href="{ $url_dir }skin/<mandant>/favicon.ico" />
4.   <link type="text/css" rel="stylesheet" href="{ $url_dir }skin/{ $site->skin }/style.css" />
5. </head>
6. <body>
7.   <table>
8.     <tr>
9.       <td>
10.        
11.      </td>
12.      <td>
13.        {navigation id=$site->rootId var=hauptnavigation}
14.        {foreach from=$hauptnavigation item=seite}
15.          <a href="{page_url page=$seite}">{$seite->field.name}</a>&nbsp;&nbsp;&nbsp;
16.        {/foreach}
17.      </td>
18.    </tr>
19.    <tr>
20.      <td>
21.        {navigation page=$page show=true var=navigation}
22.        {foreach from=$navigation item=seite}
23.          <br/><a href="{page_url page=$seite}">{$seite->field.name}</a>
24.        {/foreach}
25.      </td>
26.      <td>
27.        <div id="inhalt">
28.          {if $typeTemplate}
29.            {include file=$typeTemplate}
30.          {else}
31.            <h2><a name="top">{$page->field.title}</a></h2>
32.            <p><i>{$page->field.short|nl2br}</i></p>
33.            <p>{$page->field.content}</p>
34.          {/if}
35.        </div>
36.      </td>
37.    </tr>
38.  </table>
```

```
39. </body>
40. </html>
```

```
<html>
<head>
  <link rel="shortcut icon" href="{ $url_dir }skin/<mandant>/favicon.ico" />
  <link type="text/css" rel="stylesheet" href="{ $url_dir }skin/{ $site->skin }/style.css" />
</head>
<body>
  <table>
    <tr>
      <td>
        
      </td>
      <td>
        {navigation id=$site->rootId var=hauptnavigation}
        {foreach from=$hauptnavigation item=seite}
          <a href="{page_url page=$seite}">{ $seite->field.name}</a>&nbsp;&nbsp;&nbsp;
        {/foreach}
      </td>
    </tr>
    <tr>
      <td>
        {navigation page=$page show=true var=navigation}
        {foreach from=$navigation item=seite}
          <br/><a href="{page_url page=$seite}">{ $seite->field.name}</a>
        {/foreach}
      </td>
      <td>
        <div id="inhalt">
          {if $typeTemplate}
            {include file=$typeTemplate}
          {else}
            <h2><a name="top">{ $page->field.title}</a></h2>
            <p><i>{ $page->field.short|nl2br}</i></p>
            <p>{ $page->field.content}</p>
          {/if}
        </div>
      </td>
    </tr>
  </table>
</body>
</html>
```

## Beispiel (HEAD-Bereich)

*Die Seite lässt sich im Editor einfach nicht speichern.*

Wie jede normale HTML-Datei verfügt auch die index.html über einen Kopf-Bereich in dem verschiedene Meta-Daten hinterlegt werden können.

EGOTEC bietet einige Platzhalter an, die speziell für den HEAD-Bereich gedacht sind. Ein vollständiger HEAD-Bereich könnte dann wie folgt aussehen :

## QuelltextHTML Code:

1. `<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="{ $site->language}" lang="{ $site->language}">`
2. `<head>`
3. `<meta http-equiv="content-type" content="text/html; charset=UTF-8" />`
4. `<title>{if`  
`$page->extra.meta_title}{ $page->extra.meta_title}{else}{ $page->field.title|escape:"html"}{/if}</title>`
5. `<meta http-equiv="content-language" content="{ $site->language}" />`
6. `<meta name="description" lang="{ $site->language}" content="{if`  
`$page->extra.meta_descr}{ $page->extra.meta_descr|escape:"html"}{else}{ $site->site.description|escape:"html"}{/if}`  
`</meta>`
7. `</head>`

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="{ $site->language}" lang="{ $site->language}">
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <title>{if $page->extra.meta_title}{ $page->extra.meta_title}{else}{ $page->field.title|escape:"html"}{/if}</title>
  <meta http-equiv="content-language" content="{ $site->language}" />
  <meta name="description" lang="{ $site->language}" content="{if
$page->extra.meta_descr}{ $page->extra.meta_descr|escape:"html"}{else}{ $site->site.description|escape:"html"}{/if}"
/>
</head>
```

## **Der Seitentitel**

### QuelltextHTML Code:

1. `<title>{if`  
`$page->extra.meta_title}{ $page->extra.meta_title}{else}{ $page->field.title|escape:"html"}{/if}</title>`

```
<title>{if $page->extra.meta_title}{ $page->extra.meta_title}{else}{ $page->field.title|escape:"html"}{/if}</title>
```

In diesem Beispiel wird das Titel-Feld der aktuellen Seite auch als Titel des Browser verwendet. Hier sind natürlich andere Kombinationen möglich.

Außerdem wird an dieser Stelle geprüft, ob auf dem Meta-Reiter (im Administrationsbereich der Seite) das Feld "Titel im Browser" befüllt wurde. Falls ja, wird dieser Titel verwendet.

## **Die Meta-Angaben**

### QuelltextHTML Code:

1. `<meta http-equiv="content-type" content="text/html; charset=UTF-8" />`

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
```

Der *content-type* sollte auf jeden Fall auf *text/html; charset=UTF-8* gesetzt werden, da EGOTEC komplett auf UTF-8 ausgelegt ist.

### QuelltextHTML Code:

1. `<meta http-equiv="content-language" content="{ $site->language}" />`

```
<meta http-equiv="content-language" content="{ $site->language}" />
```

Die Sprache der dargestellten Seite kann dynamisch über `{ $site->language }` gesetzt werden. Bei mehrsprachigen Auftritten wird damit immer die korrekte Sprache gesetzt.

#### QuelltextHTML Code:

```
1. <meta name="description" lang="{ $site->language}"
    content="{ $site->site.description|escape:"html"}{ $page->extra.meta_descr|escape:"html"}" />
```

```
<meta name="description" lang="{ $site->language}"
content="{ $site->site.description|escape:"html"}{ $page->extra.meta_descr|escape:"html"}" />
```

Die Beschreibung der Seite werden die Platzhalter `{ $site->site.description }` und `{ $page->extra.meta_descr }` verwendet.

Der Platzhalter `{ $site->site.description }` entspricht dabei der global hinterlegten Beschreibung (unter: Verwaltung->Konfiguration->*Mandant*->Allgemein).

`{ $page->extra.meta_descr }` hingegen zeigt die Beschreibung auf dem Meta-Reiter der jeweiligen Seite.

Je nach Bedarf können Sie an dieser Stelle gerne beide Platzhalter kombinieren (siehe Beispiel) oder über eine `{if}`-Kontrollstruktur nur einen verwenden

#### QuelltextHTML Code:

```
1. <meta name="keywords" lang="{ $site->language}" content="{if
    $page->extra.meta_keys}{ $page->extra.meta_keys|escape:"html"}{else}{ $site->site.keywords|escape:"html"}"
/>
```

```
<meta name="keywords" lang="{ $site->language}" content="{if
$page->extra.meta_keys}{ $page->extra.meta_keys|escape:"html"}{else}{ $site->site.keywords|escape:"html"}{/if}"
/>
```

Die Schlüsselwörter verhalten sich ähnlich wie die Beschreibung der Seite. Auch diese können aus den speziellen Schlüsselwörtern der Seite und/oder aus den globalen Einstellungen generiert werden.

#### QuelltextHTML Code:

```
1. <meta name="date" content="{ $page->field.c_date}" />
```

```
<meta name="date" content="{ $page->field.c_date}" />
```

Das Aktualisierungs-Datum der Seite kann leicht über einen Platzhalter (Datum der letzten Änderung) positioniert werden.

#### QuelltextHTML Code:

```
1. <meta name="robots" content="{ $site->site.robots}" />
```

```
<meta name="robots" content="{ $site->site.robots}" />
```

Die Anweisungen für Suchmaschinen-Robots können aus den Einstellungen des Mandanten übernommen werden. Die entsprechende Seiteneinstellung aus dem Meta-Reiter kann über den Platzhalter `{ $page->extra.meta_robots }` verwendet werden.

#### QuelltextHTML Code:

1. `<meta name="generator" content="{ $generator}" />`

`<meta name="generator" content="{ $generator}" />`

Über den Platzhalter `{ $generator}` wird die aktuelle Version von EGOTEC als Generator der Webseite angegeben.

## Favicons

QuelltextHTML Code:

1. `<link rel="shortcut icon" href="{ $url_dir}skin/{ $site->skin}/favicon.ico" />`

`<link rel="shortcut icon" href="{ $url_dir}skin/{ $site->skin}/favicon.ico" />`

Über diese Zeile wird ein individuelles FAVICON eingebunden. Falls Sie dazu Grafiken verwenden möchten, die sich innerhalb des EGOTEC-Pfades befinden, sollten Sie die URL unbedingt mit `{ $url_dir}` beginnen lassen.

## Styles

QuelltextHTML Code:

1. `<link rel="StyleSheet" href="{ $url_dir}skin/{ $site->skin}/style.css" />`

`<link rel="StyleSheet" href="{ $url_dir}skin/{ $site->skin}/style.css" />`

Der Pfad zu CSS-Dateien besteht aus `{ $url_dir}` dem URL-Pfad, und `{ $site->skin}`, dem aktuell verwendeten Design des Mandanten. Bitte beachten Sie weiterführende [Hinweise zu Stylesheets](#).

QuelltextSmarty Code:

1. `{include _module_files page=$page}`

`{include _module_files page=$page}`

Der Platzhalter für typenspezifische CSS bzw. Javascript Dateien wird von EGOTEC automatisch befüllt. Näheres zu diesem Platzhalter finden Sie auf der Seite [{include \\_module\\_files}](#)

## Typenspezifisches Template einbinden

Damit ein seitenspezifisches Template im Frontend angezeigt wird, muss es erst im **Haupttemplate eingebunden** werden:

QuelltextSmarty Code:

1. `{if $typeTemplate}`  
2. `{include file=$typeTemplate}`  
3. `{else}`  
4. `{ $page->field.content}`  
5. `{/if}`

`{if $typeTemplate}`  
  `{include file=$typeTemplate}`

```
{else}
    {$page->field.content}
{/if}
```

Über den Platzhalter **{\$typeTemplate}** wird geprüft, ob für die aktuelle Seite ein typenpezifisches Template existiert. Falls dies der Fall ist, wird dieses an der entsprechenden Stelle eingebunden. Andernfalls wird lediglich der Inhalt der Seite dargestellt.

## Style-Hinweis

Um eine identische Darstellung im Frontend und im WYSIWYG-Editor zu gewährleisten, sollten Sie den Inhaltsbereich mit einem `<div id="inhalt">` umschließen und die CSS-Styles entsprechend an dieser ID ausrichten.

QuelltextSmarty Code:

1. `<div id="inhalt">`
2. `{if $typeTemplate}`
3. `{include file=$typeTemplate}`
4. `{else}`
5.  `{$page->field.content}`
6. `{/if}`
7. `</div>`

```
<div id="inhalt">
    {if $typeTemplate}
        {include file=$typeTemplate}
    {else}
        {$page->field.content}
    {/if}
</div>
```

# Seitenspezifische Templates

Für jeden Seitentyp kann ein eigenes Aussehen bestimmt werden.

Das Template für einen Seitentyp befindet sich stets unter **skin / DESIGN / SEITENTYP / body.html**.

Es wird über einen entsprechenden Platzhalter in das Haupttemplate eingebunden und beinhaltet alle Ausgaben und besondere Funktionalitäten für den entsprechenden Seitentyp. Hier kann z.B. angegeben werden, dass alle Unterseiten in einer Liste mit jeweils Titel, Kurzbeschreibung und Verlinkung (News/Liste) ausgegeben werden sollen.

Die einfachste Variante einer solchen *body.html* könnte wie folgt aussehen:

QuelltextHTML Code:

1. <!-- Den Titel ausgeben -->
2. <h1>{\$page->field.title}</h1>
- 3.
4. <!-- Ausgabe der Kurzbeschreibung -->
5. <i>{\$page->field.short}</i>
- 6.
7. <!-- Ausgabe des Inhalts -->
8. {\$page->field.content}

```
<!-- Den Titel ausgeben -->
<h1>{$page->field.title}</h1>
```

```
<!-- Ausgabe der Kurzbeschreibung -->
<i>{$page->field.short}</i>
```

```
<!-- Ausgabe des Inhalts -->
{$page->field.content}
```

Hier wird lediglich der Titel, die Kurzbeschreibung und der Inhalt ausgegeben. Dabei können Sie Platzhalter an beliebiger Stelle im Template positionieren.

Nähere Informationen zum Erstellen von Seitentypen und deren Namensgebung finden Sie im Entwicklerhandbuch

Weil die "body.html"-Datei nur Teil des Haupttemplates ist und in diesem eingebunden wird, benötigen Sie weder einen <html>, noch <head> oder <body>-Tag.

# Stylesheets

Es wird sehr empfohlen alle verwendeten CSS-Anweisungen in eine (oder mehrerer) externe Datei auszulagern. Diese kann dann in dem head-Bereich des Haupttemplates der Website eingebunden werden. Das hat den entscheidenden Vorteil, dass Änderungen nur an dieser zentralen Stelle durchgeführt werden.

Mit dieser Zeile binden Sie ein externe CSS-Datei ein:

QuelltextHTML Code:

```
1. <link rel="StyleSheet" type="text/css" href="{ $base_url }skin/{ $site->skin }/style.css" />
```

```
<link rel="StyleSheet" type="text/css" href="{ $base_url }skin/{ $site->skin }/style.css" />
```

## Zusätzliche Styles pro Seitentyp

Wie Sie für einen bestimmten Seitentyp eine weiterführende CSS-Datei einbinden können, finden Sie auf der Onlinehilfe unter [{include\\_module\\_files}](#).

## style.css

Die *style.css* ist die Hauptdatei für Stylesheets. Hier werden alle CSS-Klassen und Definitionen gesammelt.

Gerne können Sie auch weitere CSS-Dateien über diese Datei verlinken:

QuelltextCSS Code:

```
1. /* CSS-Hauptdatei */
2.
3. @import url("erweitert.css");
4. @import url("special.css");
5. @import url("usw.css");
```

```
/* CSS-Hauptdatei */
```

```
@import url("erweitert.css");
@import url("special.css");
@import url("usw.css");
```

Alle diese Stylesheet-Angaben sind gewöhnlich durch den Designer/Templateentwickler fest vorgegeben. Ein Redakteur hat auf diese Einstellungen keinen Einfluss.

Diese CSS und darin verlinkte CSS-Dateien werden im **WYSIWYG-Editor** verwendet. Achten Sie also darauf, dass alle im Design verwendeten Styles in dieser style.css-Dateie stehen oder zumindest die entsprechende CSS-Datei hier verlinkt wird. Es ist darauf zu achten das im **WYSIWYG-Editor** das DIV **#inhalt** verwendet wird.

# Suchtreffer hervorheben

Verwenden Sie folgende CSS-Anweisung in der style.css um nach einer Suche die Suchtreffer hervorzuheben.

QuelltextCSS Code:

1. .highlight {
2. background-color:yellow;
3. font-weight:bold;
4. }

```
.highlight {  
background-color:yellow;  
font-weight:bold;  
}
```

Gerne können auch eigene Style-Angaben für die Formatierung verwendet werden.

## span.css

Über die *span.css* kann man einem Redakteur einige vorgefertigte Textformatierungen bereit stellen. Diese kann er im WYSIWYG-Editor über das Dropdown-Menü "CSS-Style" einem Textelement bzw. Absatz zuweisen.

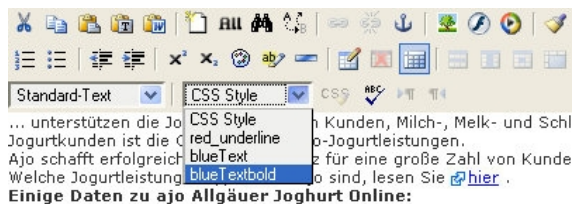
Folgende "**CMS-Installationspfad/skin/Mandant Name/span.css**"

QuelltextCSS Code:

1. .red\_underline {}
2. .blueText {}
3. .blueTextbold {}

```
.red_underline {}  
.blueText {}  
.blueTextbold {}
```

entspricht dann im Editor folgender Auswahlmöglichkeit:



Wie Styles einzelnen Textbausteinen zugewiesen werden können wird im Benutzerhandbuch auf folgender Seite beschrieben.

Dies in der span.css festgelegten Klassen müssen in der style.css definiert werden.

## table.css

Die *table.css* dient ebenfalls dazu dem Redakteur im WYSIWYG-Editor vorgefertigte Formatierungen bereit zu stellen. Allerdings sind die hier definierten Klassen für die Formatierung von Tabellen gedacht (vgl. "Einfügen und Bearbeiten einer Tabelle" im Benutzerhandbuch).

### Beispiel table.css

QuelltextCSS Code:

1. `.testTable {}`
2. `.egoTable {}`
3. `.farben_wechsel {}`
4. `.Ego_Buergerservice_Table {}`
5. `.Ego_Buergerservice_Table th {}`

```
.testTable {}  
.egoTable {}  
.farben_wechsel {}  
.Ego_Buergerservice_Table {}  
.Ego_Buergerservice_Table th {}
```

Dies in der *table.css* festgelegten Klassen müssen in der *style.css* definiert werden.

## img.css

Die *img.css* dient dazu dem Redakteur im WYSIWYG-Editor vorgefertigte Formatierungen bereit zu stellen. Allerdings sind die hier definierten Klassen für die Formatierung von Bildern gedacht (vgl. "Einfügen von Bildern" im Benutzerhandbuch).

### Beispiel img.css

QuelltextCSS Code:

1. `.roter_Rahmen {}`
2. `.gelber_Rahmen {}`

```
.roter_Rahmen {}  
.gelber_Rahmen {}
```

Dies in der *img.css* festgelegten Klassen müssen in der *style.css* definiert werden.

## CSS-Hinweise

Einige Richtlinien, auf die ein Designer bei der Erstellung der CSS-Anweisungen achten sollte:

## Allgemeines

- Verschachtelungen und abhängige CSS-Anweisungen sollten nur in den statischen Bereichen der Templates verwendet werden.
- <h1> bis <h6> sollten definiert und für den Gebrauch im eigentlichen Inhalt reserviert sein.
- In den statischen Bereichen des Templates sollten alle CSS-Anweisungen über CSS-Klassen oder die TAG-IDs festgelegt werden.

## Wiederholende Elemente

Wiederholende Elemente sind z.B. Menüpunkte und ähnliches.

Bereiche die sich vom HTML-Code ähnlich sind, und sich quasi nur durch den dargestellten Text unterscheiden.

- CSS-Anweisungen sollten für solche Elemente identisch sein (keine unterschiedlichen Breiten, Höhen, usw.)

## Im eigentlichen Inhalt

- CSS-Anweisungen die für den eigentlichen Inhalt verwendet werden, sollten ohne Abhängigkeiten und Verschachtelungen auskommen.
- Der eigentliche Inhaltsbereich sollte nicht noch weiter durch DIVs unterteilt werden.

## Auf bestimmte Tags beschränken

Sofern sich einige CSS-Klassen ausschließlich auf bestimmte Tag-Typen beziehen sollen, ist es ratsam, der Class-Definition diesen Tag mit einem Punkt voranzustellen.

So könnte man bei den Definitionen in der img.css beispielsweise auch *img.roter\_Rahmen* verwenden um sicher zu stellen, dass diese Anweisung nur bei Bildern interpretiert wird.

## Weitere Hinweise

Um eine bestmögliche Darstellung im WYSIWYG-Editor zu erzielen sollten Sie nach Möglichkeit darauf verzichten eigene CSS-Klassen zu verwenden.

Sinnvoller ist es die HTML-Tags selbst zu beschreiben. Über verschachtelte Definitionen können Sie die Klassen dabei auf den Inhaltsbereich und den Editor begrenzen.

Die folgende CSS-Definition würde z.B. auf alle <p>-Tags im Editor angewandt werden. Um im Frontend die gleiche Formatierung zu erreichen, muss der Inhaltsbereich in der Index.html dementsprechend von einem Bereich mit der ID inhalt eingeschlossen sein. (vgl. "Der BODY-Bereich" im Designhandbuch)

QuelltextCSS Code:

```
1. #inhalt p {
2.  font-family : Times New Roman;
3.  font-style : normal;
4.  font-size : 1.0em;
5. }
```

```
#inhalt p {
font-family : Times New Roman;
font-style : normal;
font-size : 1.0em;
}
```

## Templates für verschiedene Medien

Zusätzlich zu der Ausgabe als HTML-Dateien unterstützt EGOTEC auch Templates für weitere Medien.

Standardmäßig besteht z.B. die Möglichkeit für eine Druckansicht ein abgeändertes Design zu verwenden.

Im Zusammenhang mit dem PDF-Modul können auch Designs zur PDF-Generierung erstellt werden.

## Templates für Druckansichten

Für die Druckansicht über `{print_url}` verwendet EGOTEC als Basis ebenfalls HTML-Templates.

Diese können entweder global für einen Mandanten oder speziell für einen Seitentyp erstellt werden. In der Regel werden Elemente wie Navigation, rechte Spalten usw. in Drucktemplates entfernt um die Ansicht auf wichtigen Inhalt zu beschränken.

### Das Haupttemplate

Falls sich im entsprechenden Design-Verzeichnis eines Mandanten eine Datei *index.print.html* befindet, wird diese statt der *index.html* in der Druckansicht verwendet.

Eine solche *index.print.html* verzichtet dabei meistens auf die Navigationen, und konzentriert sich stärker auf den eigentlichen Inhalt der Seite.

### Seitenspezifische Templates

Falls notwendig können auch für *body.html*-Dateien durch spezielle Templates die Druckansicht ersetzen. Diese Dateien müssen haben die Bezeichnung *body.print* und befinden sich jeweils neben der *body.html* im gleichen Seitentyp-Verzeichnis.

## Templates für die PDF-Generierung

Für die PDF-Ansicht über `{pdf_url}` verwendet EGOTEC als Basis ebenfalls HTML-Templates. Diese werden aber für die PDF-Ansicht entsprechend vereinfacht.

# Das Haupttemplate

Falls im entsprechenden Design-Verzeichnis eines Mandanten eine Datei *namens index.pdf.html* oder *index.htm.html* befindet wird diese statt der *index.html* in der PDF-Ansicht verwendet.

Eine solche *index.pdf.html* verzichtet dabei meistens auf die Navigationen, und konzentriert sich stärker auf den eigentlichen Inhalt der Seite.

# Seitenspezifische Templates

Falls notwendig können auch für *body.html*-Dateien durch spezielle Templates für die PDF-Ansicht ersetzt werden.

# Mögliche Endungen

## htmldoc

body.pdf.html bzw. index.pdf.html

## html2ps

body.htm.html bzw. index.htm.html

Für die PDF-Generierung setzen wir das externe Tool *html2ps* ein. Diesem sind bzgl. der Darstellungsmöglichkeiten einige Grenzen gesetzt.

Näheres dazu entnehmen Sie bitte der Dokumentation von *html2ps* (<http://www.easysw.com/html2ps/>). Auch ist es Möglich *html2ps* einzusetzen.

# Templates für Adobe InDesign

Erzeugen eines InDesign Dokuments mittels EGOTEC CMS:

1. Man muss mittels Adobe InDesign eine Vorlage erzeugen und diese dann als Indesign Interchange Format exportieren.



2. Danach muss man in der erzeugten Datei unter Umständen ein paar `{literal}` und `{/literal}` Tags setzen.

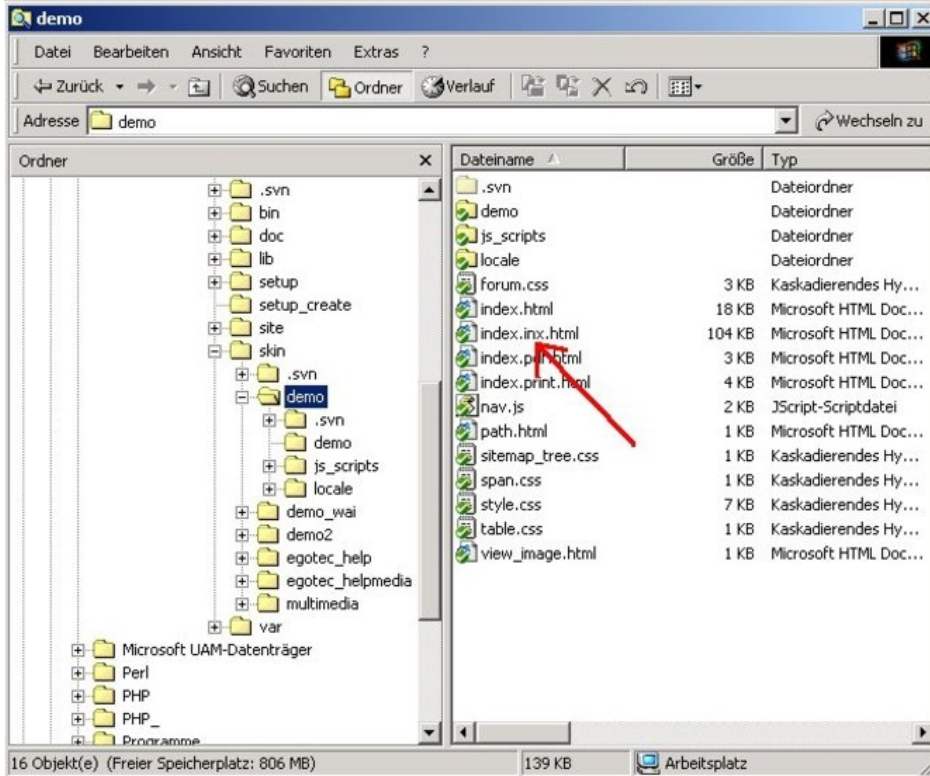
```
1 {literal}<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <?aid style="33" type="document" DOMVersion="4.0" readerVersion="3.
3 <docu pMep="o_dcMep1" InLi="x_0" zero="x_2_U_0_U_0" pacl="o_uab" pt
  " DCbi="e_RICs" DCii="e_RICs" CSrp="e_CPPP" CScp="e_CPPd" DCsa="b_f
4 <lang pnam="rk_[No Language]" lsqu="k_&apos;&apos;" ldqu="k
  Self="rc_u38"/>
5 <lang pnam="rk_English: UK" lsqu="k_'" ldqu="k_'" pril="k
```

```

606      <Jgda font="c_Times New Roman" ptfs="k" ptsz="D_9.212598
n="e_lltj" jgal="e_Jact" jcal="e_Jact" Self="rc_u10eJgda1"/>
607      <txsr prst="o_uao" crst="o_u56">
608          <pcnt>c_{/literal}{content}{literal}</pcnt>
609      </txsr>
610  </cflo>
611  <cflo pXLS="x_0" pWRD="x_0" pTXT="x_0" pSMP="x_2_x_0_x_0" Xans="r

```

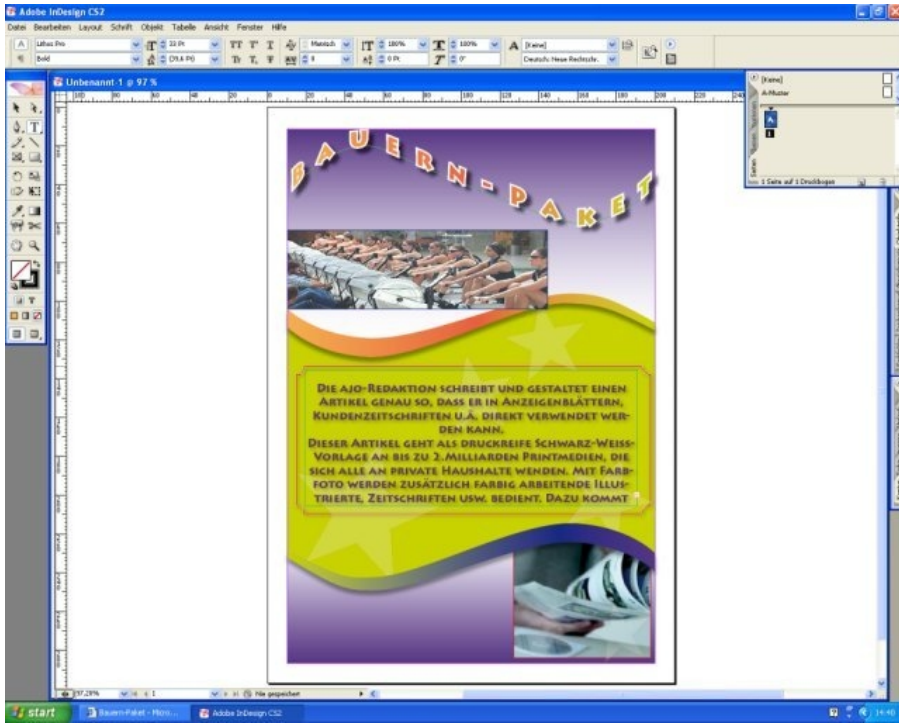
3. Danach speichert man die Datei im Skin Verzeichnis :



4. Dann kann man sich das ganze mit gefülltem Inhalt runterladen:



5. Und im Adobe InDesign öffnen:



# Templates zur Bilddarstellung

Falls verlinkte Bilder aus dem Multimediabereich gesondert dargestellt werden sollen, kann dies über die `view_image.html` gesteuert werden.

Die `view_image.html` muss sich im Skin-Verzeichnis des Desktop-Mandanten befinden. Zusätzlich muss unter Verwaltung->Konfiguration->Multimedia-Mandant->Administration Reiter Verschiedenes die Checkbox bei "Links auf Bilder für Popups erweitern" aktiviert werden. Das können Sie sich [Hier](#) anschauen.

Eine solche `view_image.html`-Datei könnte z.B. wie folgt aussehen :

QuelltextHTML Code:

```
1. <span class="quellcode">{* Die maximale Breite für Bilder festlegen *}
2. {assign var=maxWidth value=500}
3.
4. <html>
5.   <head>
6.     <title>{$page->field.title}</title>
7.   </head>
8.   <body>
9.     {capture assign="inhalt"}
10.    <h1>{$page->field.name}</h1>
11.    <p>
12.      {if $page->extra.origImgWidth > $maxWidth}
13.        <a href="{page_url id=$page->field.id media=1}" target="_blank">
14.          
15.        </a>
16.      {else}
17.        
18.      {/if}
19.    </p>
20.    <p>{$page->field.short}</p>
21.    <p>{t}Originalgröße{/t} : {$page->extra.origImgWidth}x{$page->extra.origImgHeight}</p>
22.    <p>{t}Format{/t} : {$page->extra.image_type}</p>
23.    {if $page->extra.origImgWidth > $maxWidth}
24.      <p><a href="{page_url id=$page->field.id media=1}" target="_blank">{t}Original
        anzeigen{/t}</a></p>
25.    {/if}
26.    {/capture}
27.    {$inhalt|max_image_width:500}
28.  </body>
29. </html>
```

```
{* Die maximale Breite für Bilder festlegen *}
{assign var=maxWidth value=500}
```

```
<html>
  <head>
    <title>{$page->field.title}</title>
  </head>
  <body>
    {capture assign="inhalt"}
    <h1>{$page->field.name}</h1>
    <p>
```

```

{if $page->extra.origImgWidth > $maxWidth}
  <a href="{page_url id=$page->field.id media=1}" target="_blank">
    
  </a>
{else}
  
{/if}
</p>
<p>{$page->field.short}</p>
<p>{t}Originalgröße{/t} : {$page->extra.origImgWidth}x{$page->extra.origImgHeight}</p>
<p>{t}Format{/t} : {$page->extra.image_type}</p>
{if $page->extra.origImgWidth > $maxWidth}
  <p><a href="{page_url id=$page->field.id media=1}" target="_blank">{t}Original anzeigen{/t}</a></p>
{/if}
{/capture}
{$inhalt|max_image_width:500}
</body>
</html>

```

Im Gegensatz zu den seitenspezifischen Templates sollte die *view\_image.html* über einen HEAD- und BODY-Bereich verfügen.

# Mehrsprachigkeit

Sie können mit `{t}/{t}` recht einfach einzelne Textbausteine übersetzen.  
Dazu stehen Ihnen im Verzeichnis `skin/SkinName/locale` für jede verfügbare Sprache Übersetzungsdateien zur Verfügung.

Vorbereitungen

Legen Sie in `skin/mandant/`

einen Ordner mit dem Namen "locale" an

in diesem Ordner wird für jede gewünschte Sprache ein Ordner mit dem Länderkürzel als Namen angelegt, innerhalb der Sprache kommt eine `translation.ini`

Beispiel

- skin
  - ◆ demo
    - ◇ locale
      - de
        - translation.ini
      - en
        - translation.ini
      - fr
        - translation.ini

## Aufbau einer translation.ini

Diese Seite als PDF Dokument anzeigen. = show this page as PDF

Diese Seite ausdrucken. = print this page

Download = download

Drucken = print

Impressum = imprint

# Smarty

Zur Unterstützung der dynamischen Inhaltsgestaltung verwendet Egotec über verschiedene Erweiterungen. Diese Erweiterungen basieren auf der Smarty Template Engine. Im Rahmen dieses Handbuches werden die häufigsten dieser Erweiterungen aus Sicht des Designer beschrieben.

Auf den Folgenden Seiten bekommen Sie Informationen über den Einsatz von Smarty-Plugins

Eine Einführung und Informationen zur grundlegenden Syntax von Smarty finden Sie auf der [Smarty-Homepage](#).

## Smarty Kommentare

Ein Smarty-Kommentar beginnt mit `{*` und endet mit `*`. Alles was sich zwischen diesen Zeichen befindet, wird in der Ausgabe ignoriert, d.h. auch nicht in der Quelltextansicht der Seite angezeigt.

### Beispiel

QuelltextSmarty Code:

1. `{* Das ist ein Kommentar, welcher von Smarty nicht beachtet und in der Ausgabe entfernt wird *}`

`{* Das ist ein Kommentar, welcher von Smarty nicht beachtet und in der Ausgabe entfernt wird *}`

# Smarty Platzhalter

Die einfachste Erweiterung des HTML-Codes sind sogenannte Platzhalter.

Wie auch alle anderen Erweiterungen beginnen Platzhalter mit { und enden mit }. Sie werden je nach anzuzeigender Seite mit unterschiedlichen Inhalten gefüllt.

Zwischen den Platzhaltern steht der Variablenname, beginnend mit einem \$.

## Beispiel für einen Platzhalter

QuelltextSmarty Code:

```
1. {$mein_name}
```

```
{$mein_name}
```

Smarty bietet zusätzlich zu den Platzhaltern auch noch die Möglichkeit, konstante Werte aus Konfigurationsdateien einzulesen. Näheres dazu ist der Smarty-Dokumentation zu entnehmen. Mit Hilfe der *debug*-Funktion können Sie sich die möglichen Platzhalter zu einzelnen Templates jederzeit anzeigen lassen.

## url\_dir

```
{$url_dir}
```

Der Platzhalter {`$url_dir`} beinhaltet die URL zu Ihrer CMS-Installation.

Er wird in der Templates verwendet um Pfade zu Bildern, Skripten oder CSS-Dateien anzugeben:

QuelltextHTML Code:

1. `<!-- Das Logo der Website -->`
2. `skin }/img/logo.gif />
```

Im Platzhalter ist am Ende schon ein "/"-Zeichen mit enthalten. Setzen Sie daher nach dem Platzhalter KEIN "/"-Zeichen mehr (siehe Beispiel).

## page

```
{$page}
```

Bei jedem Seitenaufruf wird vom System die aktuelle Seite als Page-Objekt in Smarty bereit gestellt. Mit dem Platzhalter **{\$page}** können gespeicherte Werte und Eigenschaften direkt im Template ausgegeben werden.

| Platzhalter                           | Beschreibung              |
|---------------------------------------|---------------------------|
| <code>{\$page-&gt;field.name}</code>  | Name der aktuellen Seite  |
| <code>{\$page-&gt;field.title}</code> | Titel der aktuellen Seite |
| <code>{\$page-&gt;field.short}</code> | Kurzbeschreibung          |

{`$page->field.content`} Inhalt  
{`$page->field.a_date`} Erstellungsdatum  
{`$page->field.c_date`} Änderungsdatum  
{`$page->field.id`} ID der aktuellen Seite  
{`$page->field.type`} Interner Seitentyp  
Mit {`$page->extra.variablenname`} kann auch auf Werte im extra-Feld zugegriffen werden, sofern diese Werte vorhanden sind.

## site

### {`$site`}

Der {`$site`} Platzhalter steht bei jedem Seitenaufruf automatisch zu Verfügung. Er beinhaltet das Objekt des aktuellen Mandanten.

### Einige Ausgabemöglichkeiten

| Platzhalter                                   | Beschreibung                               |
|-----------------------------------------------|--------------------------------------------|
| { <code>\$site-&gt;name</code> }              | Name des aktuellen Mandanten               |
| { <code>\$site-&gt;skin</code> }              | Verwendetes Design des aktuellen Mandanten |
| { <code>\$site-&gt;language</code> }          | Die Sprache des Mandanten                  |
| { <code>\$site-&gt;site.default_skin</code> } | Das Standard-Design des Mandanten          |

Über das Array {`$site->site`} können allgemeine Einstellungen (aus dem Adminbereich) wie Standardsprache, verfügbare Sprachen oder E-Mail Adresse des Webadministrators ausgegeben werden. Alle verfügbaren Werte können über die {`debug`}-Funktion eingesehen werden.

# Smarty Blöcke

Mit Hilfe von Smarty Blöcken (und Kontrollstrukturen) können Sie die Darstellung der Seite steuern.

Im Prinzip sind Smarty-Blöcke nichts anderes als Smarty-Funktionen mit dem wichtigen Unterschied, dass sie mit einem schließenden Tag beendet werden müssen.

Die gebräuchlichsten werden im Folgenden beschrieben.

## **{if}{elseif}{else}{/if}**

Mit Hilfe von `{if}{elseif}{else}{/if}` -Anweisungen können Sie bestimmte Bereiche eines Templates von Bedingungen abhängig machen. So ist es z.B. möglich je nach Typ der Seite unterschiedliche Bilder oder einen alternativen Begrüßungstext einzublenden.

### Beispiel

QuelltextSmarty Code:

1. `{if $lang == "fr"}`
2.    Bienvenue sur notre site
3. `{elseif $lang == "de"}`
4.    Willkommen auf unserer Website
5. `{else}`
6.    Welcome to your website
7. `{/if}`

```
{if $lang == "fr"}
  Bienvenue sur notre site
{elseif $lang == "de"}
  Willkommen auf unserer Website
{else}
  Welcome to your website
{/if}
```

`{if}`-Statements in Smarty erlauben die selbe Flexibilität wie in PHP. Eine Liste der erlaubten Operatoren finden Sie im Smart- Onlinehandbuch hier.

## **{foreach}, {foreachelse}**

Mit Hilfe von `{foreach}{/foreach}` können Sie nacheinander alle Einträge eines Arrays durchlaufen.

### Beispiel

QuelltextSmarty Code:

1. `{foreach from=$meinArray item=eintrag}`
2.    `{$eintrag.feld1}<br />`
3.    `{$eintrag.feld2}<br />`
4. `{/foreach}`

```
{foreach from=$meinArray item=eintrag}
  {$eintrag.feld1}<br />
  {$eintrag.feld2}<br />
{/foreach}
```

Weitere [Informationen zu `{foreach}`](#) entnehmen Sie bitte den Smarty-Handbuch

## **{html\_code}{/html\_code}**

`{html_code}{/html_code}` sorgt dafür, dass HTML-Tags die in dem entsprechenden Bereich stehen, zusammen mit HTML-Tags ausgegeben werden.

(der komplette Bereich wird mit `<pre></pre>` umschlossen)

Dadurch können Sie z.B. Beispiel-HTML-Code recht einfach in einem Template anzeigen lassen.

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

## **{strip}{/strip}**

Der `{strip}`-Block entfernt alle Leerzeichen und Zeilenumbrüche, die innerhalb des `{strip}` und `{/strip}`-Tags liegen.

Oft wird die Ausgabe des erzeugten HTML im Browser durch letztere negativ beeinflusst. Daher kann diese Funktion an dieser Stelle Abhilfe schaffen.

## **Beispiel**

QuelltextSmarty Code:

```
1. {strip}
2. <table>
3. <tr>
4.   <td>
5.     <span>
6.       Hallo
7.     </span>
8.   </td>
9. </tr>
10.</table>
11. {/strip}
```

```
{strip}
<table>
<tr>
<td>
<span>
Hallo
</span>
</td>
</tr>
```

```
</table>
{/strip}
```

## Ausgabe

QuelltextHTML Code:

```
1. <table><tr><td><span>Hallo</span></td></tr></table>
```

```
<table><tr><td><span>Hallo</span></td></tr></table>
```

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

## {t}{/t}

Der {t}-Block vereinfacht Ihnen das Erstellen von mehrsprachigen Seiten. Texte, die innerhalb eines {t}{/t}-Blocks stehen, werden an Hand von Übersetzungsdateien automatisch in die gewählte Sprache übersetzt. Näheres zu den Übersetzungsdateien und wie diese gepflegt werden finden Sie [hier](#).

Ein typischer Aufruf des {t}{/t}-Blockes sieht wie folgt aus :

QuelltextSmarty Code:

```
1. {t}Dieser Text wird automatisch ersetzt.{/t}
```

```
{t}Dieser Text wird automatisch ersetzt.{/t}
```

Beachten Sie dabei, dass zwischen den {t}{/t}- Tags weder weitere Template-Platzhalter, noch HTML-Tags auftauchen sollten. Lediglich einfacher Text.

QuelltextSmarty Code:

```
1. {t}Die aktuelle Seite trägt den Namen{/t}{ $page->field.name}<br/>
2. {t}Und die Kurzbeschreibung wird{/t}<i>{/t}kursiv{/t}</i>{/t}dargestellt:{/t}<br/>
3. <i>{/t}{ $page->field.short|nl2br}</i>
```

```
{t}Die aktuelle Seite trägt den Namen{/t}{ $page->field.name}<br/>
```

```
{t}Und die Kurzbeschreibung wird{/t}<i>{/t}kursiv{/t}</i>{/t}dargestellt:{/t}<br/>
```

```
<i>{/t}{ $page->field.short|nl2br}</i>
```

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

## {literal}{/literal}

Mit Hilfe von `{literal}{/literal}` können Sie Bereiche des Templates von der Interpretation durch Smarty ausschließen. Smarty ignoriert damit alles, was zwischen `{literal}` und `$/literal}` steht. Dies ist vor allem für Javascript- oder andere Blöcke nützlich, die geschwungene Klammern verwenden (Ohne den `{literal}`-Block würde Smarty versuchen die geschweiften Klammern der Javascript-Funktion zu interpretieren, was zu einem Fehler führen würde).

QuelltextSmarty Code:

1. {literal}
2. <script language="javascript">
3. <!--
4. function meineFunktion(var
5. {
6.   alert(var);
7. }
8. //-->
9. </script>
10. {/literal}

```
{literal}
<script language="javascript">
<!--
function meineFunktion(var
{
  alert(var);
}
//-->
</script>
{/literal}
```

## {cache}{/cache}

### Beispiel

QuelltextSmarty Code:

1. {cache name="navigation1"}
- 2.
3. {\* Alle Unterseiten der Startseite auslesen \*}
4. {get\_children id=1 var="kinder"}
5. <ul>
6.   {foreach from=\$kinder item="kind"} {\* Alle Unterseiten durchlaufen \*}
7.     <li><a href="{page\_url page=\$kind}">{\$kind->field.name}</a></li> {\* Seitenname \*}
8.   {/foreach}
9. </ul>
- 10.
11. {/cache}

```
{cache name="navigation1"}
```

```
{* Alle Unterseiten der Startseite auslesen *}
{get_children id=1 var="kinder"}
<ul>
  {foreach from=$kinder item="kind"} {* Alle Unterseiten durchlaufen *}
    <li><a href="{page_url page=$kind}">{$kind->field.name}</a></li> {* Seitenname *}
  {/foreach}
</ul>

{/cache}
```

Diese Funktion speichert den komplette Bereich, welcher durch die {cache}-Tags umschlossen wird. Bei variablen Inhalten ist daher auf den korrekten Einsatz dieser Funktion zu achten.

# Smarty Funktionen

Smarty-Funktionen sind wichtige Werkzeuge, die Ihnen die Arbeit mit Templates erleichtern. Der entscheidende Vorteil: Sowohl der Aufruf der Funktion als auch die weitere Verarbeitung der Funktionsergebnisse können direkt im Template durchgeführt werden.

Die wichtigsten Funktionen werden im folgenden beschrieben.

Die folgenden Funktionen sind nicht Standard von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu diesen Funktionen finden.

## {admin\_url}

### Beispiel

QuelltextSmarty Code:

1. {admin\_url id=10 action=edit}

```
{admin_url id=10 action=edit}
```

"url" oder "action" muss gesetzt sein, jedoch darf nur einer der beiden Parameter verwendet werden. Wenn Sie ein page Objekt zur Verfügung haben, verwenden Sie bitte page= als Parameter und nicht id.

## {assign}

### Beispiel

QuelltextSmarty Code:

1. {assign var="var" value="Hallo Welt"}

2. {\$var}

```
{assign var="var" value="Hallo Welt"}
```

```
{$var}
```

### Ausgabe

QuelltextHTML Code:

1. Hallo Welt

Hallo Welt

## {check\_captcha}

## Beispiel 1

QuelltextSmarty Code:

1. {check\_captcha var=ergebnis}

```
{check_captcha var=ergebnis}
```

## Beispiel 2

QuelltextSmarty Code:

1. {if \$smarty.request.submit}
2. {get\_captcha var=captcha\_result}
- 3.
4. {if \$captcha\_result}
5. Das CAPTCHA wurde korrekt gelöst.
6. {else}
7. Ihre Eingabe ist nicht richtig!
8. {/if}
9. {/if}

```
{if $smarty.request.submit}
  {get_captcha var=captcha_result}
```

```
  {if $captcha_result}
    Das CAPTCHA wurde korrekt gelöst.
  {else}
    Ihre Eingabe ist nicht richtig!
  {/if}
```

```
{/if}
```

Wenn das Formular abgeschickt wurde, wird überprüft ob das CAPTCHA gelöst wurde und dementsprechend eine Meldung ausgegeben.

## {config\_load}

### Beispiel

QuelltextSmarty Code:

1. {config\_load file="farben.conf"}

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

## {count\_pages}

### Beispiel

QuelltextSmarty Code:

1. {\* Die Kind-Elemente der aktuellen Seite auslesen \*}
2. {get\_children query.fields="id,order\_field" page=\$page query.where="type='news/entry'" var="alle\_news\_eintraege"}

- 3.
4. `{* Alle Ergebnisse zählen *}`
5. `{count_pages pages=$alle_news_eintraege var=scount}`
- 6.
7. `{* Anzahl der Ergebnisse ausgeben *}`
8. `{$scount}`

```
{* Die Kind-Elemente der aktuellen Seite auslesen *}
{get_children query.fields="id,order_field" page=$page query.where="type='news/entry'"
var="alle_news_eintraege"}
```

```
{* Alle Ergebnisse zählen *}
{count_pages pages=$alle_news_eintraege var=scount}
```

```
{* Anzahl der Ergebnisse ausgeben *}
{$scount}
```

## **{debug}**

### **Beispiel**

QuelltextSmarty Code:

1. `{debug}`

```
{debug}
```

Der Aufruf von `{debug}` erfolgt üblicherweise möglichst am Ende des Templates.

Vergessen Sie nicht, alle `{debug}` Aufrufe vor der Liveschaltung wieder zu entfernen. Andernfalls werden die Debug-Ausgaben auch weiterhin angezeigt.

## **{extra\_push}**

### **Beispiel**

QuelltextSmarty Code:

1. `{* in das extra-Feld "wochentag" den Wert "Dienstag" schreiben *}`
2. `{extra_push id=1 field="Wochentag" value="Dienstag" }`

```
{* in das extra-Feld "wochentag" den Wert "Dienstag" schreiben *}
```

```
{extra_push id=1 field="Wochentag" value="Dienstag" }
```

Existiert bereits ein entsprechender Eintrag im Extra-Bereich wird er **überschrieben**.

Es können nur Seiten geändert werden, die direkt zum aktuellen Mandanten gehören. Media-Dateien lassen sich z.B. mit `{extra_push}` nicht bearbeiten

## {file\_exists}

### Beispiel

QuelltextSmarty Code:

```
1. {file_exists file="/var/www/cms/skin/demo/index.html" var="file_exists_index"}
```

```
{file_exists file="/var/www/cms/skin/demo/index.html" var="file_exists_index"}
```

## {get\_captcha}

**Achtung:** Wenn Sie die Werte für name\_text und name\_hidden ändern, müssen Sie diese auch *{check\_captcha}* mitteilen.

Die Captcha-Funktion benötigt das GD-Lib Modul mit integrierter Free-Type Bibliothek (<http://de2.php.net/imagettfbbox>).

### Beispiel 1

QuelltextSmarty Code:

```
1. {* prüfen, ob der Eintrag korrekt war *}  
2. {get_captcha var=html_code}
```

```
{* prüfen, ob der Eintrag korrekt war *}  
{get_captcha var=html_code}
```

Über *{check\_captcha}* kann geprüft werden, ob die Eingabe korrekt war.

### Beispiel 2

Erzeugt ein CAPTCHA mit roter Schriftfarbe und grünen Linien und speichert den HTML-Code für das erzeugte Bild und für das Eingabefeld in der Variablen html\_code

QuelltextSmarty Code:

```
1. {get_captcha var=html_code text_color=#ff0000 lines_color=#00ff00}
```

```
{get_captcha var=html_code text_color=#ff0000 lines_color=#00ff00}
```

### Beispiel 3

Erzeugt ein CAPTCHA und speichert den HTML-Code für das erzeugte Bild in der Variablen code\_img und das für das Eingabefeld in code\_txt.

QuelltextSmarty Code:

```
1. {get_captcha var_text=code_txt var_image=code_img}
```

```
{get_captcha var_text=code_txt var_image=code_img}
```

Weitere Möglichkeiten zur Anpassung werden mit den CSS Klassen captcha\_user, captcha\_img und captcha\_reload angeboten.

## {get\_children}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiele

QuelltextSmarty Code:

1. {\* alle Kinder der aktuellen Seite auslesen \*}  
2. {get\_children page=\$page var="kinder"}
- 3.
4. {\* die ersten beiden Kinder der aktuellen Seite auslesen \*}  
5. {get\_children page=\$page var="kinder" query.limit="0,2"}
- 6.
7. {\* alle Kinder der aktuellen Seite, die vom Typ "news/entry" sind, auslesen \*}  
8. {get\_children page=\$page var="kinder" query.where="type='news/entry'"}

```
{* alle Kinder der aktuellen Seite auslesen *}  
{get_children page=$page var="kinder"}
```

```
{* die ersten beiden Kinder der aktuellen Seite auslesen *}  
{get_children page=$page var="kinder" query.limit="0,2"}
```

```
{* alle Kinder der aktuellen Seite, die vom Typ "news/entry" sind, auslesen *}  
{get_children page=$page var="kinder" query.where="type='news/entry'"}
```

## {get\_descendants}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiele

QuelltextSmarty Code:

1. {\* alle untergeordnete Seiten auslesen \*}  
2. {get\_descendants page=\$page var="nachfahren"}
- 3.
4. {\* alle Seiten des kompletten Auftritts zurückgeben, die vom Typ "news/entry" sind \*}  
5. {get\_descendants id=\$site->rootId query.where="type = 'news/entry'"}

```
{* alle untergeordnete Seiten auslesen *}  
{get_descendants page=$page var="nachfahren"}
```

```
{* alle Seiten des kompletten Auftritts zurückgeben, die vom Typ "news/entry" sind *}  
{get_descendants id=$site->rootId query.where="type = 'news/entry'"}
```

Diese Funktion übernimmt beim Auslesen der Seiten leider **keine variable Sortierung** aus dem Administrationsbereich  
(Reiter "Navigation => Sortierung: variabel").

## {get\_keywords}

### Beispiele

QuelltextSmarty Code:

1. {\* die Keywords der aktuellen Seite ausgeben \*}
2. {get\_keywords var="keywords" page=\$page}
- 3.
4. {\* Schlagworte ausgeben \*}
5. Aktuelle Schlagworte: {\$keywords}

```
{* die Keywords der aktuellen Seite ausgeben *}
{get_keywords var="keywords" page=$page}
```

```
{* Schlagworte ausgeben *}
```

```
Aktuelle Schlagworte: {$keywords}
```

Möchte man die Schlagworte einzeln ausgegeben, wird empfohlen, die Liste Skriptseitig über die explode-Funktion in ein Array abzulegen und an Smarty zu übergeben.

## {get\_pages}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiel

QuelltextSmarty Code:

1. {\* die 5 zuletzt geänderten Seiten vom Typ "news/entry" sortiert auslesen \*}
2. {get\_pages var="seiten" query.where="type='news/entry'" query.order="c\_date DESC" query.limit='0,5' }
- 3.
4. {\* Seitennamen (der Ergebnisse) in einer Liste darstellen \*}
5. <ul>
6.     {foreach from=\$seiten item="seite"}
7.         <li>{\$seite->field.name}</li>
8.     {/foreach}
9. </ul>

```
{* die 5 zuletzt geänderten Seiten vom Typ "news/entry" sortiert auslesen *}
```

```
{get_pages var="seiten" query.where="type='news/entry'" query.order="c_date DESC" query.limit='0,5' }
```

```
{* Seitennamen (der Ergebnisse) in einer Liste darstellen *}
```

```
<ul>
```

```
  {foreach from=$seiten item="seite"}
```

```
    <li>{$seite->field.name}</li>
```

```
  {/foreach}
```

```
</ul>
```

## {get\_parents}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiel

QuelltextSmarty Code:

1. {\* alle Eltern der aktuellen Seite auslesen \*}
2. {get\_parents page=\$page var="eltern"}

```
{* alle Eltern der aktuellen Seite auslesen *}
{get_parents page=$page var="eltern"}
```

## {get\_path}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiele

QuelltextSmarty Code:

1. {\* aktuellen Pfad (Seiten von der Startseite bis zur Elternseite) auslesen \*}
2. {get\_path page=\$page var="pfad"}
- 3.
4. {\* gleiche Abfrage wie oben, jedoch ist die aktuelle Seite auch im Ergebnis enthalten \*}
5. {get\_path page=\$page var="pfad" show\_self=1}
- 6.
7. {\* gleiche Abfrage wie oben, jedoch in umgekehrter Reihenfolge \*}
8. {get\_path page=\$page var="pfad" show\_self=1 bottomup=1}

```
{* aktuellen Pfad (Seiten von der Startseite bis zur Elternseite) auslesen *}
{get_path page=$page var="pfad"}
```

```
{* gleiche Abfrage wie oben, jedoch ist die aktuelle Seite auch im Ergebnis enthalten *}
{get_path page=$page var="pfad" show_self=1}
```

```
{* gleiche Abfrage wie oben, jedoch in umgekehrter Reihenfolge *}
{get_path page=$page var="pfad" show_self=1 bottomup=1}
```

## {get\_siblings}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiel

QuelltextSmarty Code:

1. {\* alle Seiten vom Typ "news/entry" auslesen, die sich auf gleicher Ebene mit \$page befinden \*}
2. {get\_siblings page=\$page var="geschwister" query.where="news/entry"}

```
{* alle Seiten vom Typ "news/entry" auslesen, die sich auf gleicher Ebene mit $page befinden *}
{get_siblings page=$page var="geschwister" query.where="news/entry"}
```

## {get\_sibling}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiele

QuelltextSmarty Code:

1. {\* den führenden Nachbar auslesen \*}
2. {get\_sibling var="pPage" page=\$page direction="prev"}
- 3.
4. {\* den folgenden Nachbar auslesen \*}
5. {get\_sibling var="nPage" page=\$page direction="next"}

```
{* den führenden Nachbar auslesen *}
{get_sibling var="pPage" page=$page direction="prev"}
```

```
{* den folgenden Nachbar auslesen *}
{get_sibling var="nPage" page=$page direction="next"}
```

## {get\_user}

### Beispiel

QuelltextSmarty Code:

1. {\* den Benutzer auslesen, der die aktuelle Seite zuletzt bearbeitet hat \*}
2. {get\_user user\_id=\$page->field.a\_user var="benutzer"}

```
{* den Benutzer auslesen, der die aktuelle Seite zuletzt bearbeitet hat *}
{get_user user_id=$page->field.a_user var="benutzer"}
```

## {help}

In der Regel wird dieses Plugin in der ersten Zeile eines Dialogs eingebunden.



### Beispiel

QuelltextSmarty Code:

1. {help id=99}

{help id=99}

Diese Plugin ist erst ab EGOTEC 5 verfügbar.

## {icon\_by\_extension}

### Beispiel

QuelltextSmarty Code:

1. {\* einen IMG-Tag zu einem Symbol für Word-Dokumente erzeugen \*}
2. {icon\_by\_extension extension="doc"}

```
{* einen IMG-Tag zu einem Symbol für Word-Dokumente erzeugen *}
{icon_by_extension extension="doc"}
```

## {include\_module\_files}

Es wird im Seitentyp-Verzeichnisse nach 2 Dateien durchsucht :

- skin/Mandant/seitentyp/**style.css**
- skin/Mandant/seitentyp/**script.js**

### Beispiel

QuelltextSmarty Code:

1. <html>
2. <head>
3. <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
4. <title>{\$page->field.title|escape:"html"}</title>
- 5.
6. {include\_module\_files page=\$page}
- 7.
8. </head>

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <title>{$page->field.title|escape:"html"}</title>
```

```
{include_module_files page=$page}
```

```
</head>
```

{include\_module\_files} sollte unbedingt im HEAD-Bereich der index.html eingebunden werden

## {include}

### Beispiel

QuelltextSmarty Code:

1. `{* Template Bereiche einbinden *}`
2. `{include file="tpl/header.html"}`
3. `{include file="tpl/content.html"}`

```
{* Template Bereiche einbinden *}
{include file="tpl/header.html"}
{include file="tpl/content.html"}
```

Der Pfad geht dabei immer vom jeweiligen SKIN-Verzeichnis aus.

### Typenspezifische Templates einbinden

EGOTEC stellt automatisch einen Platzhalter `{$typeTemplate}` zur Verfügung. Dieser enthält immer den Template-Pfad zur aktuellen Seite.

QuelltextSmarty Code:

1. `{* Liegt ein Template für die aktuelle Seite vor? *}`
2. `{if $typeTemplate}`
3. `{include file=$typeTemplate} {* dann dieses verwenden *}`
4. `{else}`
5.  `{$page->field.content} {* ansonsten den einfachen Inhalt ausgeben *}`
6. `{/if}`

```
{* Liegt ein Template für die aktuelle Seite vor? *}
{if $typeTemplate}
    {include file=$typeTemplate} {* dann dieses verwenden *}
{else}
    {$page->field.content} {* ansonsten den einfachen Inhalt ausgeben *}
{/if}
```

## {input}

*Smarty Plugin für den Adminbereich. Einfache Erzeugung von Eingabefeldern.*

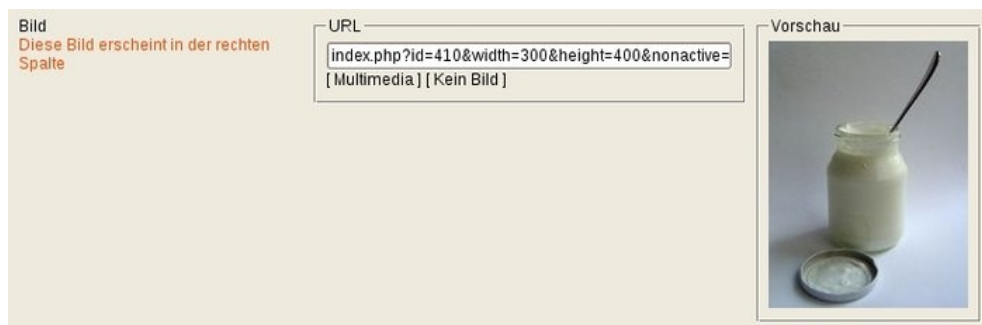
### Beispiel

QuelltextSmarty Code:

1. `{* Eingabefeld für ein Bild erzeugen *}`
2. `{input version=5 type="image" name="bild_rechts" title="Bild" short="Diese Bild erscheint in der rechten Spalte"}`

```
{* Eingabefeld für ein Bild erzeugen *}
{input version=5 type="image" name="bild_rechts" title="Bild" short="Diese Bild erscheint in der rechten Spalte"}
```

**Ausgabe:**



## Die HTML-Datei

wir erstellen zuerst eine HTML-Datei

jede HTML Datei besitzt 2 Bereiche

### Bereich 1 Smarty+HTML

QuelltextSmarty Code:

1. <form name="extra">
2. <div align="center">
3. <table class="table">
4. <tr>
5. <td class="cell">
6. {input version=5 type="button" name="input\_test" title="Dies ist ein button" onclick="javascript:test()}"
7. </td>
8. </tr>
9. </table>
10. </div>
11. </form>

```
<form name="extra">
<div align="center">
<table class="table">
  <tr>
    <td class="cell">
      {input version=5 type="button" name="input_test" title="Dies ist ein button" onclick="javascript:test()}"
    </td>
  </tr>
</table>
</div>
</form>
```

Achten Sie darauf, dass das Formular auf jeden Fall den Namen *extra* bekommt  
 <div> und <table> sind für die einheitliche Gestaltung optional

### Bereich 2 Javascipt

Ab Version 5 werden bei Seiten die extra-Felder automatisch gespeichert und geladen.

Es muss nur die unload\_extra() Funktion die do\_unload bzw. do\_unload\_extra Funktion aufgenommen werden.

## QuelltextJavaScript Code:

```
1. // Für die Erweiterung des Informationsreiters
2.
3. <script language="javascript">
4. {literal}
5. function do_load_extra(){
6.
7. function do_unload_extra()
8. {
9.     window.parent.unload_extra()
10. }
11. {/literal}
12. </script>
13.
14.
15. //Für einen neuen Reiter
16.
17. <script language="javascript">
18. {literal}
19. function do_load()
20. {
21. }
22.
23. function do_unload()
24. {
25.     window.parent.unload_extra();
26. }
27. {/literal}
28. </script>
29.
```

// Für die Erweiterung des Informationsreiters

```
<script language="javascript">
{literal}
function do_load_extra(){

function do_unload_extra()
{
    window.parent.unload_extra()
}
{/literal}
</script>
```

//Für einen neuen Reiter

```
<script language="javascript">
{literal}
function do_load()
{
}

function do_unload()
```

```

{
  window.parent.unload_extra();
}
{/literal}
</script>

```

Wenn Sie Input-Plugins an anderen Stellen verwenden, z.B. auf einem Konfigurationsreiter in den Globalen Einstellungen können die Felder nicht automatisch abgespeichert werden.

Die Inhalte der Input-Felder können in dieser Version einheitlich mit den Funktionen `set_input_value()` und `get_input_value()` geladen und gespeichert werden.

QuelltextJavaScript Code:

```

1. function do_load()
2. {
3.   set_input_value('menue_color', window.parent.get_conf('administration', 'menue_color'));
4. }
5.
6. function do_unload()
7. {
8.   window.parent.set_conf('administration', 'menue_color', get_input_value('menue_color'));
9. }

```

```

function do_load()
{
  set_input_value('menue_color', window.parent.get_conf('administration', 'menue_color'));
}

```

```

function do_unload()
{
  window.parent.set_conf('administration', 'menue_color', get_input_value('menue_color'));
}

```

### **do\_load**

In diesem Teil werden die Werte beim Speichern in die DB geschrieben

### **do\_unload**

Hier werden, wenn vorhanden, gespeicherte Werte an das Formular übergeben

Mit Dojo generierte Input-Felder können nicht vollständig über die Eigenschaften ihrer ursprünglichen HTML Objekte manipuliert werden. Hierfür ist es manchmal notwendig das dazugehörige Dojo Objekt zu ermitteln und dieses dann zu manipulieren. Die Funktion hierfür ist `get_input_object()`. Übergeben wird der Name des Input-Feldes.

QuelltextJavaScript Code:

```

1. var obj = get_input_object('menue_color');

```

```

var obj = get_input_object('menue_color');

```

Soll ein Attribut geändert werden, verwendet man die `attr()` Funktion. Nur so aktualisiert Dojo seine Objekte.

### QuelltextJavaScript Code:

1. `obj.attr('disabled', true); // ändert einen Wert`
- 2.
3. `obj.attr({disabled : true, label : 'Neuer Text'}); // ändert mehrere Werte auf einmal`

`obj.attr('disabled', true); // ändert einen Wert`

`obj.attr({disabled : true, label : 'Neuer Text'}); // ändert mehrere Werte auf einmal`

Auslesen von Eigenschaften ist hingegen wie gewohnt machbar. Eigenschaften oder Arrays wie `checked`, `selected`, `options`, usw stehen nun auch Verfügung.

### QuelltextJavaScript Code:

1. `obj.checked;`
2. `obj.selected;`
3. `obj.options;`

`obj.checked;`

`obj.selected;`

`obj.options;`

- {admin\_url}
- {assign}
- {check\_captcha}
- {config\_load}
- {count\_pages}
- {debug}
- {extra\_push}
- {file\_exists}
- {get\_captcha}
- {get\_children}
- {get\_descendants}
- {get\_keywords}
- {get\_pages}
- {get\_parents}
- {get\_path}
- {get\_siblings}
- {get\_sibling}
- {get\_user}
- {help}
- {icon\_by\_extension}
- {include\_module\_files}
- {include}
- {input}
  
- Button
- Checkbox
- Color
- Datetime
- Folder
- Group\_role
- Image
- Link

- [Radio](#)
- [Select](#)
- [Submit](#)
- [Text](#)
- [Textarea](#)
- [User](#)
- [Password](#)
- [{in\\_path}](#)
- [{mailto}](#)
- [{mail\\_send}](#)
- [{mail\\_valid\\_email}](#)
- [{navigation}](#)
- [{nav\\_url}](#)
- [{nav}](#)
- [{new\\_child}](#)
- [{online\\_hilfe}](#)
- [{page\\_exists}](#)
- [{page\\_url}](#)
- [{print\\_url}](#)
- [{sort\\_extra}](#)
- [{style\\_inactive}](#)
- [{sub\\_html}](#)
- [{tag\\_cloud}](#)
- [{update\\_page}](#)

## {in\_path}

Die Funktion liefert als Ergebnis entweder *true* (1) oder *false* (0) zurück.

### Beispiel

QuelltextSmarty Code:

1. `{* den Pfad ermitteln *}`
2. `{get_path var="mein_pfad" id=$page->field.id show_self=1}`
- 3.
4. `{* prüfen, ob die aktuelle Seite im Pfad liegt *}`
5. `{in_path id=$page->field.id path=$mein_pfad var="ist_drin"}`

```
{* den Pfad ermitteln *}
{get_path var="mein_pfad" id=$page->field.id show_self=1}
```

```
{* prüfen, ob die aktuelle Seite im Pfad liegt *}
{in_path id=$page->field.id path=$mein_pfad var="ist_drin"}
```

`{in_path}` eignet sich besonders für mehrstufige baumartige Navigationsstrukturen, welche je nach Position der Besuchers an entsprechender Stelle aufklappen. Dabei wird einmalig über `{get_path}` der aktuelle Pfad ermittelt und in jeder Navigationsebene geprüft, ob die Seiten im Pfad liegen. Falls ja, werden weiter Kinder ausgelesen.

## {mailto}

Weitere **Parameter** und **Beispiele** entnehmen Sie bitte der [Smarty Dokumentation](#) auf smarty.net

## {mail\_send}

### Beispiel

QuelltextSmarty Code:

```
1. {mail_send to="info@egotec.com" from="demo@egotec.com" subject="Demo Mail" message="Ein
    etwas längerer Text"}
```

```
{mail_send to="info@egotec.com" from="demo@egotec.com" subject="Demo Mail" message="Ein etwas
längerer Text"}
```

## {mail\_valid\_email}

Rückgabewert ist true oder false.

### Beispiel

QuelltextSmarty Code:

```
1. {mail_valid_email adress="demo@egotec.com" var="mail_check"}
2.
3. {if $mail_check}
4.   richtig
5. {else}
6.   falsch
7. {/if}
```

```
{mail_valid_email adress="demo@egotec.com" var="mail_check"}
```

```
{if $mail_check}
  richtig
{else}
  falsch
{/if}
```

## {navigation}

Diese Funktion wird nicht weiter unterstützt und wurde durch die Funktion [{nav}](#) ersetzt.

## {nav\_url}

### Beispiel

QuelltextSmarty Code:

```
1. {nav id=$site->rootId item="nav1" query.fields='id,name' param.has_children=1}
2.   <li>
3.     <a href="{nav_url}">{$nav1->field.name}</a>
4.   {/if}
5. {/nav}
```

```
{nav id=$site->rootId item="nav1" query.fields='id,name' param.has_children=1}
  <li>
    <a href="{nav_url}">{$nav1->field.name}</a>
  {/if}
{/nav}
```

{nav\_url} kann nur innerhalb eines {nav}{/nav} Blocks verwendet werden.

## {nav}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiel

QuelltextSmarty Code:

```
1. {** Navigationsliste der ersten Ebene erzeugen *}
2. <ul>
3.   {nav id=1 item="ebene1"}
4.     <li><a href="{nav_url}">{$ebene1->field.name}</a></li>
5.   {/nav}
6. </ul>
```

```
{** Navigationsliste der ersten Ebene erzeugen *}
<ul>
  {nav id=1 item="ebene1"}
    <li><a href="{nav_url}">{$ebene1->field.name}</a></li>
  {/nav}
</ul>
```

## {new\_child}

### Beispiel

QuelltextSmarty Code:

```
1. {new_child parent_id=$parent_id var="new_page"
2.   field="array(
```

```

3. 'name' => '$title',
4. 'title' => '$title',
5. 'content' => '$content',
6. 'type' => 'forum/entry',
7. 'inactive' => '0',
8. 'nav_hide' => '1'"
9. extra="array(
10. 'anhang' => '$inhalt',
11. 'anhang_mime_type' => '$mtype',
12. 'anhang_file_name' => '$fname',
13. 'entry_user' => '$entry_user',
14. 'entry_email' => '$entry_email')"
15. }

```

```

{new_child parent_id=$parent_id var="new_page"
field="array(
'name' => '$title',
'title' => '$title',
'content' => '$content',
'type' => 'forum/entry',
'inactive' => '0',
'nav_hide' => '1')"
extra="array(
'anhang' => '$inhalt',
'anhang_mime_type' => '$mtype',
'anhang_file_name' => '$fname',
'entry_user' => '$entry_user',
'entry_email' => '$entry_email')"
}

```

## {online\_hilfe}

### Beispiel

QuelltextSmarty Code:

1. {\* Link auf Benutzerhandbuch \*}
2. <a href="{online\_hilfe id=6}">Benutzerhandbuch</a>

```

{* Link auf Benutzerhandbuch *}
<a href="{online_hilfe id=6}">Benutzerhandbuch</a>

```

Diese Smarty-Plugin ist in EGOTEC 5 nicht mehr verfügbar. Es wird durch das Plugin [help](#) ersetzt.

## {page\_exists}

Diese Funktion berücksichtigt Freigabe-Daten, Berechtigungen und den Aktiv-Haken.

### Beispiel

QuelltextSmarty Code:

1. {\* prüfen, ob die aktuelle Seite auch im englischen Bereich existiert \*}

2. {page\_exists id=\$page->field.id lang="en" var="page\_in\_en"}

```
{* prüfen, ob die aktuelle Seite auch im englischen Bereich existiert *}
{page_exists id=$page->field.id lang="en" var="page_in_en"}
```

## {page\_url}

### Beispiel

QuelltextSmarty Code:

1. {\* einen Link auf das aktuelle Page Objekt erstellen \*}
2. <a href="{page\_url page=\$page}">Link</a>
- 3.
4. {\* einen Link auf die Seite mit der ID 30 erstellen \*}
5. <a href="{page\_url id=30}">Link2</a>

```
{* einen Link auf das aktuelle Page Objekt erstellen *}
<a href="{page_url page=$page}">Link</a>
```

```
{* einen Link auf die Seite mit der ID 30 erstellen *}
<a href="{page_url id=30}">Link2</a>
```

## {print\_url}

### Beispiel

QuelltextSmarty Code:

1. <a href="{print\_url}">Seite drucken</a>

```
<a href="{print_url}">Seite drucken</a>
```

Näheres zu Druck-Designs erfahren Sie im entsprechenden [Kapitel des Designerhandbuchs](#)

## {sort\_extra}

*Sortiert nach einem Wert im Extra Feld*

### Beispiel

Die letzten 10 eingetragenen Events sortiert nach deren Enddaten anzeigen.

QuelltextSmarty Code:

1. {\* die 10 neusten Seiten vom Typ "events/entry" holen \*}
2. {get\_pages var="pages" query.where="type='events/entry'" query.order="c\_date DESC" query.limit="0,10"}

- 3.
4. {\* nach dem Wert "events\_until\_date" sortieren \*}
5. {sort\_extra var="events" pages=\$pages field="event\_until\_date" order="desc"}
- 6.
7. {\* und die Seiten ausgeben \*}
8. {foreach from=\$events item="event"}
9. ...
10. {/foreach}

```
{* die 10 neusten Seiten vom Typ "events/entry" holen *}
{get_pages var="pages" query.where="type='events/entry'" query.order="c_date DESC" query.limit="0,10"}
```

```
{* nach dem Wert "events_until_date" sortieren *}
{sort_extra var="events" pages=$pages field="event_until_date" order="desc"}
```

```
{* und die Seiten ausgeben *}
{foreach from=$events item="event"}
```

```
...
{/foreach}
```

## {style\_inactive}

### Beispiel

QuelltextSmarty Code:

```
1. 
```

```

```

## {sub\_html}

*Ein String wird abgeschnitten und der Teilstring bleibt XHTML konform.*

Als **Rückgabewert** erhält man ein Array gespeichert in der unter **var** angegebenen Variable mit folgenden Schlüsseln.

- html => Teilstring (oder der unveränderte String, falls er kleiner als length ist)
- cut => true oder false (wurde der HTML Code abgeschnitten oder nicht)

### Beispiel

QuelltextSmarty Code:

1. {sub\_html var=sub\_entry html=\$entry->field.content length=200}
2. {\$sub\_entry.html}
3. {if \$sub\_entry.cut}
4. ... <a href="{page\_url page=\$entry}">{t}weiterlesen{/t}</a>

## 5. {/if}

```
{sub_html var=sub_entry html=$entry->field.content length=200}
{$sub_entry.html}
{if $sub_entry.cut}
    ... <a href="{page_url page=$entry}">{t}weiterlesen{/t}</a>
{/if}
```

## Möglicher Anwendungsbereich

Wenn Sie auf einer Portalseite die letzten Newseinträge auflisten wollen, diese aber nicht komplett dargestellt werden sollen. Stattdessen sollen diese ab einer gewissen Länge abgeschnitten werden und einen Link erhalten, der zum eigentlichen News-Eintrag weiterführt.

## {tag\_cloud}

Eine Tag Cloud stellt die am meisten verwendeten Schlagwörter dar. Hierbei gilt standardmäßig: je größer ein Schlagwort dargestellt wird, desto öfters wird es verwendet. Ein Klick auf ein Schlagwort öffnet eine Auflistung aller Seiten deren dieses Schlagwort zugewiesen ist.

Weitere Informationen zur Verwendung von Schlagwörtern finden Sie in unserer Hilfe zum [Schlagwortregister](#).

Die Tag Cloud verwendet eine Standard Formattierung und sieht zum Beispiel so aus:



Die hierfür verwendete CSS Datei kann pro Skin überschrieben werden. Hier wird empfohlen die originale CSS Datei zu kopieren und diese dann nach eigenen Vorstellungen anzupassen.

- Original CSS Datei: **lib/smarty/plugins/tag\_cloud/style.css**
- Skin spezifische CSS Datei: **skin/<skin>/tag\_cloud.css**

Die CSS Datei wird automatisch eingebunden sobald diese Funktion aufgerufen wird. In der Regel reicht es wenn die Funktion an der Stelle im HTML Code aufgerufen wird, an der die Tag Cloud angezeigt werden soll:

QuelltextHTML Code:

1. `<h1>{t}Tag Cloud{/t}</h1>`
2. `{tag_cloud}`

```
<h1>{t}Tag Cloud{/t}</h1>
{tag_cloud}
```

Der generierte HTML Code ist so aufgebaut, dass dieser leicht mit externen Javascript Plugins verwendet werden kann:

QuelltextHTML Code:

```
1. <div id="ego_tag_cloud">
2. <ul>
3. <li>
4. <a href="#" rel="9" title="Milch" class="tag9">Milch</a>
5. </li>
6. <li>
7. <a href="#" rel="4" title="Joghurt" class="tag4">Joghurt</a>
8. </li>
9. </ul>
10. </div>
```

```
<div id="ego_tag_cloud">
<ul>
<li>
<a href="#" rel="9" title="Milch" class="tag9">Milch</a>
</li>
<li>
<a href="#" rel="4" title="Joghurt" class="tag4">Joghurt</a>
</li>
</ul>
</div>
```

Um die für die Tag Cloud notwendigen Dateien einzubinden, ist im HEAD Bereich des Skins folgender Funktionsaufruf notwendig:

**{init\_tag\_cloud}**

Damit ein Klick auf ein Schlagwort die Liste der zugeordneten Seiten öffnen kann, muss es eine Seite mit dem Seitentyp **Schlagwörter** geben.

## **{update\_page}**

Diese Funktion **ergänzt** die aktuellen Felder um die neuen Werte. Bereits vorhandene Felder werden dabei überschrieben.

### **Beispiel**

QuelltextSmarty Code:

```
1. {update_page page=$page field="array('name' => 'Neuer Name', 'title' => 'Neuer Titel')"
   extra="array('anhang' => '$inhalt')}"
```

```
{update_page page=$page field="array('name' => 'Neuer Name', 'title' => 'Neuer Titel')" extra="array('anhang'
=> '$inhalt')}"
```

# Smarty Modifikatoren

Variablen-Modifikatoren dienen dazu, den Inhalt eines Platzhalter zu bearbeiten, bevor dieser ausgegeben wird. Sie können auf alle Variablen angewendet werden.

Dazu hängen Sie einfach ein | (Pipe-Zeichen) und den Namen des Modifikators an die entsprechende Variable an. Können einem Modifikator noch Parameter mitgegeben werden, werden sie dem Modifikatorname angehängt und mit : getrennt.

## Beispiel

QuelltextSmarty Code:

1. `{* Den Titel kürzen, wenn er mehr als 15 Zeichen hat *}`
2. `{$begrueessung|truncate:15:"..."}`

```
{* Den Titel kürzen, wenn er mehr als 15 Zeichen hat *}
{$begrueessung|truncate:15:"..."}
```

## Ausgabe

QuelltextHTML Code:

1. Hallo Herr Maie...

Hallo Herr Maie...

In dem Beispiel wird ein Text gekürzt, sobald 15 Zeichen überschritten werden. In dem ersten Parameter kann die Anzahl der Zeichen bestimmt werden. Der zweite Parameter gibt die Zeichenfolge an, die nach 15 Zeichen erscheinen soll (in diesem Fall "..."). Auf diese Weise lässt sich z.B. auch ein Datum auf verschiedene Arten formatieren.

Näheres zu den Standard-Modifikatoren von Smarty können Sie der Smarty-Dokumentation entnehmen

Generell kann jede PHP-Funktion die Zeichenketten verarbeitet als Modifikator genutzt werden. Näheres hierzu finden Sie in der Smarty-Dokumentation

## max\_image\_width

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

{max\_image\_width} kann nur auf Bilder angewendet werden, die aus dem MM-Bereich von EGOTEC kommen.

*max\_image\_width* durchsucht den jeweiligen Platzhalter nach Bildern aus dem Multimediabereich. Diese werden dann mit der übergebenen maximalen Breite verglichen, und notfalls entsprechend verkleinert.

Ein typischer Aufruf des *max\_image\_width*-Modifikators sieht wie folgt aus :

{ \$page->field.content|max\_image\_width:200 }

Mit Hilfe des *{capture}*-Befehls von Smarty können Sie den Modifikator auch auf Bilder die direkt über das Template eingebunden werden anwenden.

```
{capture assign="bild"}  
  
{/capture}  
{$bild|max_image_width:150}
```

## **max\_image\_height**

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

*{max\_image\_height}* kann nur auf Bilder angewendet werden, die aus dem MM-Bereich von EGOTEC kommen

*max\_image\_height* durchsucht den jeweiligen Platzhalter nach Bildern aus dem Multimediabereich. Diese werden dann mit der übergebenen maximalen Höhe verglichen, und notfalls entsprechend verkleinert.

Ein typischer Aufruf des *max\_image\_height*-Modifikators sieht wie folgt aus :

```
{$page->field.content|max_image_height:200}
```

Mit Hilfe des *{capture}*-Befehls von Smarty können Sie den Modifikator auch auf Bilder die direkt über das Template eingebunden werden anwenden.

```
{capture assign="bild"}  
  
{/capture}  
{$bild|max_image_height:150}
```

## scale\_image

Dies ist keine Standard-Funktion von Smarty, der Aufruf steht Ihnen nur bei EGOTEC zur Verfügung. In der Smarty-Dokumentation werden Sie keine weiterführenden Hinweise zu dieser Funktion finden.

*{scale\_image}* kann nur auf Bilder angewendet werden, die aus dem MM-Bereich von EGOTEC kommen

*scale\_image* bildet eine Kombination von *max\_image\_width* und *max\_image\_height*.

Beim Aufruf von *scale\_image* können Sie sowohl eine maximale Breite, als auch eine maximale Höhe angeben.

Ein typischer Aufruf des *scale\_image*-Modifikators sieht wie folgt aus :

```
{$page->field.content|scale_image:100:100}
```

Der erste Parameter entspricht dabei der maximalen Bildbreite, der zweite Parameter der maximalen Bildhöhe.

Es ist dabei zwingend erforderlich **beide** Parameter anzugeben. Möchten Sie nur Höhe oder Breite definieren, sollten Sie statt *{scale\_image}* besser *{max\_image\_width}* bzw *{max\_image\_height}* verwenden.

Mit Hilfe des `{capture}`-Befehls von Smarty können Sie den Modifikator auch auf Bilder die direkt über das Template eingebunden werden anwenden.

```
{capture assign="bild"}  
  
{/capture}  
{$bild|scale_image:100:100}
```

## truncate

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.

### Beispiel

QuelltextSmarty Code:

1. {\$artikelTitel}
2. {\$artikelTitel|truncate}
3. {\$artikelTitel|truncate:30}

```
{$artikelTitel}  
{$artikelTitel|truncate}  
{$artikelTitel|truncate:30}
```

### Ausgabe

QuelltextHTML Code:

1. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut l
2. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod...
3. Lorem ipsum dolor sit amet, co...

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut l  
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod...  
Lorem ipsum dolor sit amet, co...

Weiter Informationen und Parameter dazu finden Sie in der Dokumentation auf [Smarty.net](http://Smarty.net).

## date\_format

Formatiert ein Datum in das gewünschte Format.

### Beispiel

QuelltextSmarty Code:

1. `{$smarty.now|date_format}`
2. `{$smarty.now|date_format:"%d.%m.%y %H:%M:%S"}`
3. `{$smarty.now|date_format:"%A, %B %e, %Y"}`

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

## Ausgabe

QuelltextHTML Code:

1. Dez 1, 2009
2. 01.12.09 09:46:36
3. Dienstag, Dezember 1, 2009

Dez 1, 2009  
01.12.09 09:46:36  
Dienstag, Dezember 1, 2009

Weiter Informationen und Parameter dazu finden Sie in der Dokumentation auf [Smarty.net](http://smarty.net).

## Weitere Smarty Modifikatoren

Eine komplette Liste der verfügbaren Smarty Modifikatoren finden Sie unter <http://www.smarty.net/manual/de/language.modifiers.php>.

## Eigene Smarty-Plugins erstellen

Wenn Sie eigene Smarty-Plugins bereitstellen möchten müssen Sie diese entweder in Ihrem Site-Verzeichnis (z.B.:site/SITENAME/plugins/smarty/) oder im var-Verzeichnis (z.B.:var/lib/plugins/smarty/) hinterlegen.

Bei einem Systemupdate (Einspielen einer neuen Version) werden CMS-fremde Dateien in lib und bin gelöscht.

Sie können auch **globale Smarty-Plugins** definieren.

Diese werden in "site/\_global/plugins/smarty/" hinterlegt und können von allen Mandanten aus genutzt werden.

# Vorlagen im Editor

*Informationen zum WYSIWYG - Vorlagenplugin (ehemals Blöcke)*

Das Vorlagen Plugin für den WYSIWYG-Editor, ermöglicht es Ihnen, eine bestimmte HTML-Vorlage oder vordefinierten HTML-Code in den Editor einfügen. Solche Blöcke können entweder speziell für jeden Mandanten oder global festgelegt werden.

Sie finden den Einfügen-Button  in der Toolbar des Editors.

# Einen neuen Block erstellen

Um einen neuen Block im WYSIWYG-Editor einzurichten, werden 2 neue Dateien im Verzeichnis **skin/MANDANT/blocks/** erstellt:

Globale Blöcke können im Verzeichnis `/skin/_global/blocks/` angelegt werden

Die Bezeichnung `meinBlock` müssen Sie durch den gewünschten Namen ersetzen

- `meinBlock_desc.php` (die Kurzbeschreibung der Blocks)
- `meinBlock.html` (der HTML-Code, welcher eingefügt werden soll)

Die Datei mit `"_desc.php"` erhält die Beschreibung Ihres Blocks, z.B:

QuelltextHTML Code:

1. Mein erster Block
2. `<br/><br/>`
3. Ein Bild mit Text

Mein erster Block

`<br/><br/>`

Ein Bild mit Text

In die `"meinBlock.html"` fügen Sie den eigentlichen HTML-Code des Blocks ein:

QuelltextHTML Code:

1. `<table border="0" width="100%>`
2. `<tr>`
3. `<td>`
4. ``
5. `</td>`
6. `<td>`
7. `Dummy-Text Dummy-Text Dummy-Text Dummy-Text`
8. `</td>`
9. `</tr>`
10. `</table>`

```
<table border="0" width="100%>
<tr>
<td>

</td>
<td>
Dummy-Text Dummy-Text Dummy-Text Dummy-Text
</td>
</tr>
</table>
```

## Vorschauansicht

Alle Dateien, die zu einem Block gehören, müssen immer den gleichen Namen besitzen.  
Die Sortierung der Vorlagen erfolgt aufsteigend (A-Z).

# Tipps und Tricks

## JavaScript

### IE IMG Node kopieren

Wenn man ein IMG-Node kopieren will, muss man zuerst das "src" Attribut setzen.

Der Internet Explorer setzt sonst alle anderen Attribute auf eine Standardeinstellung zurück.

### IE Bug Flickering Hintergrundbilder

Wird im Internet Explorer mit Hintergrundbildern gearbeitet, so tritt manchmal ein unerwünschtes (und störendes) Flackern auf.

Mit JavaScript lässt sich diesem Effekt ein wenig entgegenwirken.

Folgender Befehl bewirkt, dass die Bilder im IE Cache gelagert werden und somit schneller abgerufen werden:

```
document.execCommand('BackgroundImageCache', false, true);
```

### Variablennamen

Achten Sie bitte bei der Vergabe von Variablennamen darauf, dass ein Reiter im Adminbereich über verschiedene Template-Quellen (globale Anpassung, Seitentyp Anpassungen) gefüllt werden kann.

Verwenden Sie daher z.B. für "var f = document.forms['extra'] " nicht immer die Variable "f".

# CSS

## Listensymbole positionieren

wie man Listensymbole, die mit der Eigenschaft `list-style-image` eingebunden wurden, positionieren kann.

Hier ein Beispiel zur Erklärung:

```
<html>
<head>
<style>
ol {
  list-style-image: url(pic.gif);
}

li {
  padding : 0px;
  margin-top : -4px;
  font-size : 12px;
  line-height :17px;
  font-family:arial,Helvetica,sans-serif;
}
/* Versteht nur IE (Weiche) */
li {margin-top:0px!}

</style>
</head>
<body>
<ol>
  <li>bla</li>
  <li>MMMMMMM</li>
  <li>bla</li>
  <li>BBBBBBBBBBBB</li>
  <li>MMMMMMM</li>
  <li>bla</li>
  <li>bla</li>
</ol>
</body>
</html>
```

Das Symbol (pic.gif) hat oberhalb einen 4 Pixel-großen transparenten Rand, damit es im IE runterrutscht.

# YAML

Sollten Sie für das Layout Ihrer Webseite auf Yaml zurück greifen, kann es in einigen Bereichen des CMS zu Einschränkungen kommen.

# Vertikale und Horizontale Abstände bei Bildern

Dieses Problem tritt nur beim Einsatz des WYSIWYG-Editors aus Version 4 auf.

In der zentralen base.css-Datei von YamI werden etliche *Browser Resets* gesetzt die für gleiche Ausgangsbedingungen für alle Browser sorgen sollen.  
Eine dieser Anweisungen sieht wie folgt aus :

QuelltextCSS Code:

```
1. * { margin:0; padding:0; }
```

```
* { margin:0; padding:0; }
```

Dadurch werden sämtliche Innen- und Außenabstände aller HTML-Elemente auf Null gesetzt.  
Leider interpretieren einige Browser diese Anweisung dann stärker als die in unserem Editor gesetzten Abstände bei Bildern.

Diese Funktion wird dann unter Umständen nicht mehr korrekt ausgewertet.

Sie können das Problem umgehen, indem Sie auf die direkte Eingabe von Abständen verzichten und statt dessen in der img.css entsprechende Klassen für Bilder definieren.

Mit einem *!important* ist es möglich, die YamI-Anweisungen zu überschreiben.

# Beispiele & Vorlagen

Hier finden Sie gebrauchsfertige Vorlagen und Problemlösungen zu verschiedenen Template-Umsetzungen auf die Sie wahrscheinlich während der Template-Entwicklung stoßen werden.

## Aufklappbare Navigation

Ein zentrales Skript, über welches komplexe Navigationsstrukturen über Smarty erstellt werden können.

Dieses Skript verwendet die {nav}-Funktion und kann nach Belieben für Projekte angepasst und ändern werden.

### navigation.html

QuelltextSmarty Code:

```
1. {**
2. * XHTML-konforme Navigation für Egotec-CMS
3. * -----
4. * Features:
5. * - Navigations-Tiefe einstellbar
6. * - Nur Navigation des aktuellen Pfads anzeigen
7. * - keine leeren UL-Tags (XHTML-komform)
8. * - Einstellbare Ausgangsseite ($page oder $rootId)
9. * - CSS-Klassen:
10. * - Pfad zur aktuellen Seite: "active" (a-Element)
11. * - Aktive Seite: "active current" (a-Element)
12. * - Aktuelle Tiefe: "levelx" (ul-Element), x steht dabei für die Ebene
13. *}
14.
15. {* #####
16.  ## Konfigurationen ##
17.  ##### *}
18.
19. {* Nur Seiten im aktuellen Pfad aufklappen *}
20. {assign var="path_only" value="1"}
21.
22. {* maximale Tiefe der Navigation *}
23. {assign var="max_level" value="5"}
24.
25. {* ID der Ausgangsseite (bitte gewünschte Modus auskommentieren) *}
26. {assign var="rootPage" value=$site->getRoot()} {* Navigation startet bei ID=1 *}
27. {**assign var="rootPage" value=$page**} {* Navigation zeigt immer nur aktuelle Unterseiten *}
28.
29. {** Template-Pfad zur Include-Datei **}
30. {capture assign="childTpl"}{$egotec_conf.egotec_dir}skin/{$site->skin}/tpl/more_children.html{/capture}
31.
32. {**
33. * Beispiele:
34. * 1) Vollständig aufgeklappte Navigation (Navigationstiefe: 5) = Sitemap:
35. * path_only=0, max_level=5,rootPage=$site->getRoot
```

```

36. *
37. * 2) Aufklappbare Navigation mit 3 Ebenen
38. * path_only=1, max_level=3, rootPage=$site->getRoot
39. *}
40.
41. {* #####
42.  ## Skript ##
43.  ##### *}
44.
45. {** Aktuellen Pfad ermitteln **}
46. {get_path page=$page var="aktPfad" show_self=1}
47. {foreach from=$aktPfad item="p" key="k"}
48.   <a href="{page_url page=$p}">{$p->field.name}</a>
49. {/foreach}
50.
51. {assign var="level" value="1"}
52. <ul class="level{$level}">
53.
54.   {* Auf gleicher Ebene bleiben, wenn keine Kinder vorhanden sind *}
55.   {if $rootPage == $page}
56.
57.     {assign var="max_level" value="1"} {* nur 1 Ebene anzeigen *}
58.     {assign var="path_only" value="1"}
59.
60.     {* Prüfen, ob aktuelle Seite Unterseiten hat *}
61.     {get_children page=$page var="kind" param.no_nav_hide=1 first_page=1}
62.     {if !$kind} {* wenn nicht... *}
63.
64.       {* den aktuellen Pfad nochmal durchlaufen *}
65.       {foreach from=$aktPfad item="p" key="k"}
66.         {if $p->field.id != $page->field.id}
67.           {assign var="page_parent" value=$p}
68.         {/if}
69.       {/foreach}
70.
71.       {* als letztes Kommt der Parent raum *}
72.       {assign var="rootPage" value=$page_parent}
73.     {/if}
74.   {/if}
75.
76.   {nav id=$rootPage->field.id item="level1" param.has_children=1}
77.
78.   {* Liegt die Seite im Pfad? *}
79.   {in_path path=$aktPfad id=$level1->field.id var="inpath"}
80.   {if !$path_only}{assign var="inpath" value="1"}{/if}
81.
82.   <li><a class="{if $inpath}active{/if} {if $page->field.id == $level1->field.id}current{/if}"
href="{nav_url}">{$level1->field.name}</a>
83.
84.     {if $level1->field.has_children && $level<$max_level && $inpath} {* Gibts Unterseiten? *}
85.
86.       {* Weiter mit Unterseiten *}
87.       {include file=$childTpl start_id=$level1->field.id level=$level}
88.
89.     {/if} {* ENDIF: Gibts Unterseiten? *}

```

```

90.
91. </li>
92. {/nav}
93.
94. </ul>

```

```

{**
* XHTML-konforme Navigation für Egotec-CMS
* -----
* Features:
* - Navigations-Tiefe einstellbar
* - Nur Navigation des aktuellen Pfads anzeigen
* - keine leeren UL-Tags (XHTML-konform)
* - Einstellbare Ausgangsseite ($page oder $rootId)
* - CSS-Klassen:
* - Pfad zur aktuellen Seite: "active" (a-Element)
* - Aktive Seite: "active current" (a-Element)
* - Aktuelle Tiefe: "levelx" (ul-Element), x steht dabei für die Ebene
*}

{* #####
  ## Konfigurationen ##
  ##### *}

{* Nur Seiten im aktuellen Pfad aufklappen *}
{assign var="path_only" value="1"}

{* maximale Tiefe der Navigation *}
{assign var="max_level" value="5"}

{* ID der Ausgangsseite (bitte gewünschte Modus auskommentieren) *}
{assign var="rootPage" value=$site->getRoot()}  {* Navigation startet bei ID=1 *}
{**assign var="rootPage" value=$page**}      {* Navigation zeigt immer nur aktuelle Unterseiten *}

{** Template-Pfad zur Include-Datei **}
{capture assign="childTpl"}{$egotec_conf.egotec_dir}skin/{$site->skin}/tpl/more_children.html{/capture}

{**
* Beispiele:
* 1) Vollständig aufgeklappte Navigation (Navigationstiefe: 5) = Sitemap:
* path_only=0, max_level=5,rootPage=$site->getRoot
*
* 2) Aufklappbare Navigation mit 3 Ebenen
* path_only=1, max_level=3, rootPage=$site->getRoot
*}

{* #####
  ## Skript ##
  ##### *}

{** Aktuellen Pfad ermitteln **}
{get_path page=$page var="aktPfad" show_self=1}
{foreach from=$aktPfad item="p" key="k"}
  <a href="{page_url page=$p}">{$p->field.name}</a>
{/foreach}

```

```

{assign var="level" value="1"}
<ul class="level{ $level }">

    { * Auf gleicher Ebene bleiben, wenn keine Kinder vorhanden sind * }
    {if $rootPage == $page}

        {assign var="max_level" value="1"} { * nur 1 Ebene anzeigen * }
        {assign var="path_only" value="1"}

        { * Prüfen, ob aktuelle Seite Unterseiten hat * }
        {get_children page=$page var="kind" param.no_nav_hide=1 first_page=1}
        {if !$kind} { * wenn nicht... * }

            { * den aktuellen Pfad nochmal durchlaufen * }
            {foreach from=$aktPfad item="p" key="k"}
                {if $p->field.id != $page->field.id}
                    {assign var="page_parent" value=$p}
                {/if}
            {/foreach}

            { * als letztes Kommt der Parent raum * }
            {assign var="rootPage" value=$page_parent}
        {/if}
    {/if}

    {nav id=$rootPage->field.id item="level1" param.has_children=1}

    { * Liegt die Seite im Pfad? * }
    {in_path path=$aktPfad id=$level1->field.id var="inpath"}
    {if !$path_only}{assign var="inpath" value="1"}{/if}

    <li><a class="{if $inpath}active{/if} {if $page->field.id == $level1->field.id}current{/if}"
href="{nav_url}">{$level1->field.name}</a>

        {if $level1->field.has_children && $level<$max_level && $inpath} { * Gibts Unterseiten? * }

            { * Weiter mit Unterseiten * }
            {include file=$childTpl start_id=$level1->field.id level=$level}

        {/if} { * ENDIF: Gibts Unterseiten? * }

    </li>
{/nav}

```

</ul>

## more\_children.html

QuelltextSmarty Code:

1. {\*\*
2. \* Navigation für Unterseiten (wird rekursiv aufgerufen)
3. \* }
- 4.
5. { \* Aktuelle Ebene ermitteln \* }

```

6. {assign var="level" value=$level+1}
7.
8. <ul class="level{$level}">
9.
10. {nav id=$start_id item="child" param.has_children=1}
11.
12.   {* Im Pfad? *}
13.   {in_path path=$aktPfad id=$child->field.id var="inpath"}
14.   {if !$path_only}{assign var="inpath" value="1"}{/if}
15.
16.   <li><a class="{if $inpath}active{/if} {if $page->field.id == $child->field.id}current{/if}"
href="{nav_url}">{$child->field.name}</a>
17.
18.     {* Gibts Unterseiten ? *}
19.     {if $child->field.has_children && $level<$max_level && $inpath} {* Nur maximal eingestellte
Ebenen anzeigen *}
20.       {include file=$childTpl start_id=$child->field.id level=$level}
21.     {/if}   {* ENDIF: Gibts Unterseiten? *}
22.
23.   </li>
24.
25. {/nav}
26.
27. </ul>

```

```

{**
* Navigation für Unterseiten (wird rekursiv aufgerufen)
*}

```

```

{* Aktuelle Ebene ermitteln *}
{assign var="level" value=$level+1}

```

```

<ul class="level{$level}">

```

```

{nav id=$start_id item="child" param.has_children=1}

```

```

  {* Im Pfad? *}
  {in_path path=$aktPfad id=$child->field.id var="inpath"}
  {if !$path_only}{assign var="inpath" value="1"}{/if}

```

```

  <li><a class="{if $inpath}active{/if} {if $page->field.id == $child->field.id}current{/if}"
href="{nav_url}">{$child->field.name}</a>

```

```

    {* Gibts Unterseiten ? *}
    {if $child->field.has_children && $level<$max_level && $inpath} {* Nur maximal eingestellte Ebenen
anzeigen *}

```

```

      {include file=$childTpl start_id=$child->field.id level=$level}
    {/if}   {* ENDIF: Gibts Unterseiten? *}

```

```

  </li>

```

```

{/nav}

```

```

</ul>

```

## Einbinden der Navigation ins Template

- Erstellen Sie entsprechenden HTML-Dateien mit dem Code (siehe oben)
- Binden Sie die "navigation.html" an gewünschter Stelle im Ihrem Template ein
- Nehmen Sie gewünschte Einstellungen im Skriptbereich "konfiguration" (Datei: navigation.html) vor.

## Zweistufige Navigation

Ein recht häufiges Anwendungsbeispiel dürfte eine Navigation über mehrere Ebenen sein.

Ein entsprechender Code-Abschnitt könnte wie folgt aussehen:

```
<ul id="navmenu">
  {nav id=$site->rootId item=ebene1}
  <li><a href="{nav_url}">{$ebene1->field.name|escape:"html"}</a>
    <ul>
      {nav page=$ebene1 item=ebene2 param.has_children=1}
      <li>
        <a href="{nav_url}">{$ebene2->field.name|escape:"html"}</a>
      </li>
    </ul>
  </li>
</ul>
{/nav}
</ul>
```

Schauen Sie sich dazu bitte auch diese Seite an [{nav}](#) [{nav\\_url}](#).

# Popupbildergalerie

1. im Multimediamentanten den Haken "Links auf Bilder für Popups" setzen (Verschiedenes)
2. in der skin das Template view\_image.html anlegen

Hier sind folgende Variablen gegeben

- ◆ page
- ◆ original\_site
- ◆ original\_page
- ◆ original\_skin
- ◆ url\_dir

## Bread-Crump-Navigation / Pfad

Bestandteil fast jeder Webseite ist eine sogenannte Brotkrumen-Navigation oder auch Pfad. Dieser Pfad zeigt an, wo sich die aktuelle Seite innerhalb der Struktur des kompletten Auftritts befindet. Ein einfaches Beispiel für eine solche Brotkrumen-Navigation könnte wie folgt aussehen :

```
{get_path page=$page show_self=1 var=pfad}
{if $pfad}
  {foreach from=$pfad item=seite name=pfad}
    <a href="{page_url page=$seite}">{$seite->field.name}</a>
    {if NOT $smarty.foreach.pfad.last}
      &gt;&gt;
    {/if}
  {/foreach}
{/if}
```

## Rekursive Aufrufe

Über rekursive Aufrufe des `{include}`-Befehls können sie auch relativ einfach beliebig Tiefe Navigationen erstellen.

Dies eignet sich besonders, um z.B. eine einfache Sitemap zu generieren.

Dazu legen Sie z.B. direkt im Design-Verzeichnis eine Datei `unterseiten.html` an, die wie folgt aufgebaut ist :

```
{if $aktuelle_id}
  {navigation id=$aktuelle_id var=unterseiten}
{else}
  {navigation page=$aktuelle_seite var=unterseiten}
{/if}

{if $unterseiten}
  <ul>
    {foreach from=$unterseiten item=seite}
      <li>
        <a href="{page_url page=$seite}">{$seite->field.name}</a>
        {include file=unterseiten.html aktuelle_seite=$seite}
      </li>
    {/foreach}
  </ul>
{/if}
```

Jetzt muss man nur noch an passender Stelle (z.B. in der `index.html`) dieses Template zum ersten Mal einbinden :

```
{include file=unterseiten.html aktuelle_id=$site->rootId}
```

Um eine solche Rekursion zu realisieren greifen wir auf die Möglichkeit zurück dem `{include}`-Befehl zusätzliche Parameter übergeben zu können.

Näheres hierzu können Sie dem entsprechenden Kapitel des Smarty-Dokumentation entnehmen

Erstellt mit  
EGOTEC®  
Internet:  
[www.egotec.com](http://www.egotec.com)  
© EGOTEC  
GmbH

EGOTEC GmbH  
Alte Neckarelzer Straße 24  
D-74821 Mosbach

Tel: +49 (0)6261 /  
6743-0  
Fax: +49 (0)6261  
/ 6743-29  
E-Mail:  
info@egotec.com