Zope Page Templates (ZPT) mit ZMS nutzen

Workshop-Agenda

10:30 *Template Attribute Language* (TAL): Konzept und Syntax
11:30 Syntax-Vergleich DTML vs. TAL
12:15 ZMS und TAL
13:00 Break
14:00 Hands-on: Templating mit TAL
15:30 Strategie und Synergie
16:30 Ende

Konzept und Syntax

Übersicht

Zope Page Templates (ZPT) werden in *TAL (Template Attribute Language)* programmiert. Es handelt sich um eine Erweiterung des XML- bzw. XHTML-Namensraumes um ein Set von Element-Attributen, das auf jedes beliebige HTML-Element (*tag*) angewendet werden kann..
Dabei wird im Prinzip ein per Programmcode erzeugtes HTML-Fragment entweder

1. als Inhalt *in* das TAL-attributierte HTML-Element geschrieben (*content*),
2. das attributierte HTML-Element komplett ersetzt
   oder
3. neue Attributwerte für das ausgezeichnete HTML-Element erzeugt

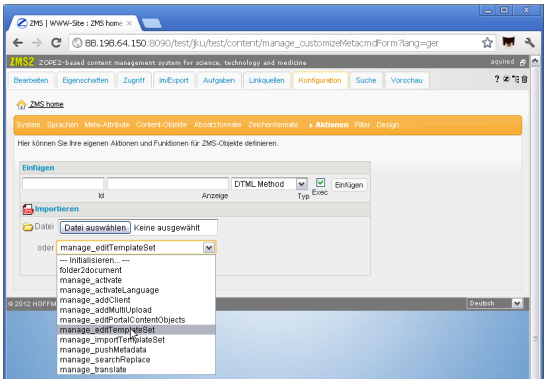
Beispiel:

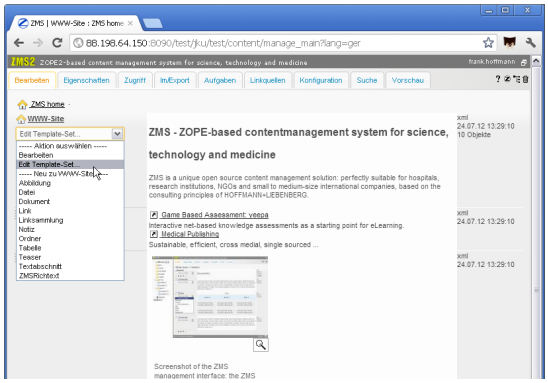
```
<title tal:content="context/title">Page Title</title>
```

Der TAL-Ausdruck liest den Objekt-Titel aus und ersetzt beim Rendering des Templates den bestehenden *default-content* "Page Title" durch diesen Objekt-Titel.
Der Vorteil des TAL-Ansatzes besteht darin, dass für das Webdesign bestehender HTML-Code als *Ganzes* verwendet kann (ohne dass eine logische Fragmentierung erforderlich ist).
Der HTML-Code wird mit den speziellen Attribute angereichert, damit der Zope-Publisher die Kontext-abhängigen funktionale Erweiterungen des HTML-Codes einsetzen kann („*Rendering*“). Im Browser kann das „ungerenderte“ Basis-Layout jederzeit kontrolliert und ein gestaltungstechnisches Feintuning durchgeführt werden.

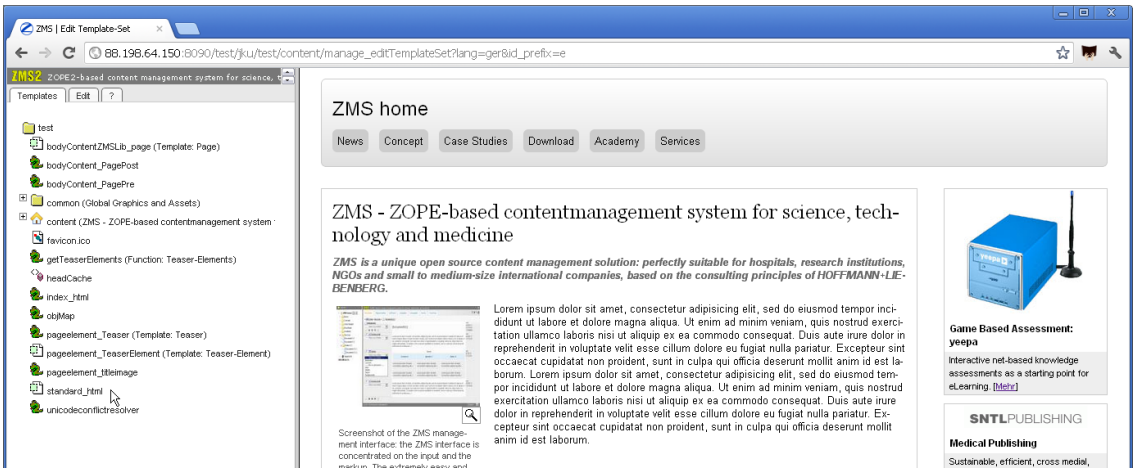
ZMS und TAL

Anstatt der schachtelnden DTML-Mastertemplates *standard_html_header* und *standard_html_footer* benötigt das TAL-Konzept nur ein einziges Master-Template: *standard_html.*
ZMS kann um eine TAL-Editor erweitert werden, der eine interaktive Bearbeitung des Templates „auf dem Content“ erlaubt. Zu diesem Zweck wird unter *Konfiguration ->Aktionen* die Standard-Aktion *editTemplateSet* importiert:





Die neue Aktion kann dann über die Aktionsliste auf den aktuellen Content-Node angewendet werden. Es erscheint eine framebasierte Vorschau: links die Templates, rechts das Seiten-Rendering. Nach Anklicken eines Templates erscheint dessen Code in einem HTML-Editor. Bei



ZMS erwartet für das TAL-Publishing das Mastertemplate mit ID *standard_html*; gleiches gilt nun auch für alle Standard-Templates der (konfigurierten) Contentobjekte. Das Master-Template ordnet Layout-Elemente und Navigations-Funktionalität um den Seitenkern an. Dieser wird über das Standard-Template *bodyContentZMSLib_page* gerendert und folgendermaßen aufgerufen:

```
<div id="pagecore" tal:replace="structure context/bodyContentZMSLib_page">
 Seiteninhalt
</div>
```

Seiteninhalt-Redering bodyContentZMSLib_page

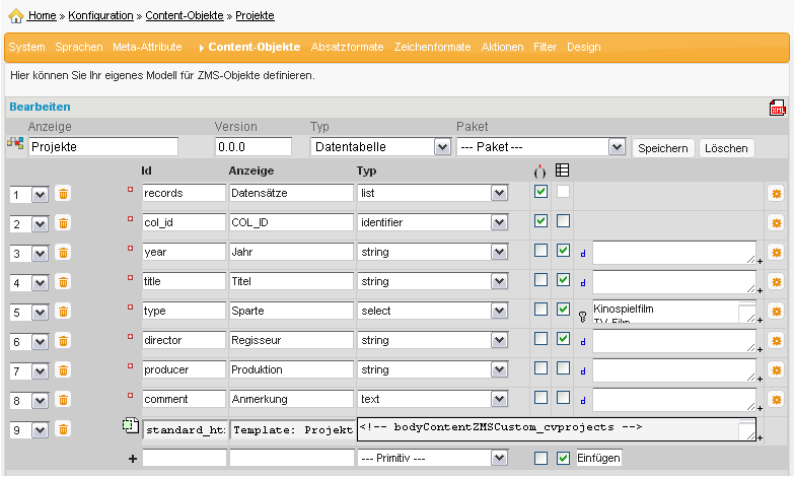
```
<!-- bodyContentZMSLib_page -->
<div id="page">
 <tal:block tal:define="global childNodes python:here.getObjChildren('e',request,here.PAGEELEMENTS)">
  <tal:block tal:condition="childNodes">
   <tal:block tal:repeat="childNode childNodes">
    <tal:block tal:content="structure python:unicode(childNode.getBodyContent(request),'utf-8')">
      The page-element body-content
    </tal:block>
   </tal:block>
  </tal:block>
 </tal:block>
</div>
<!-- /bodyContentZMSLib_page -->
```

Navigation

```
<div id="menu" tal:content="structure python:zmsroot.getNavItems(
     here,request,{'add_self':False,'deep':False,'complete':False} )"
  ><ul>
    <li><a href="#" title="Item1"><span>Concept</span></a></li>
    <li><a href="#" title="Item2"><span>Cases</span></a></li>
    <li><a href="#" title="Item3"><span>Download</span></a></li>
    <li><a href="#" title="Item4"><span>Academy</span></a></li>
    <li><a href="#" title="Item5"><span>Services</span></a></li>
  </ul>
</div>
```

Listen-Iteration ZMS-Datentabelle

Screenshot: Definition der Datentabelle per ZMS-Konfiguration am Beispiel einer Werk-Liste



Screenshot: HTML-Ausgabe der Werk-Liste

Jahr	Titel	Kategorie	Regie	Produktion	Anmerkung
2001	Das Jahr der ersten Küsse	Kinospielfilm	Kai Wessel	mementoFilm	
2000	Das Experiment	Kinospielfilm	Oliver Hirschbiegel	Fanes Film	
2000	Nancy & Frank	Kinospielfilm	Wolf Gremm	Ziegler Film	NRW
2000	Mr. Boogie	Kinospielfilm	Vesna Jovanoska	ena Film	
2000	Dust	Kinospielfilm	Milcho Manchevski	ena Film	NRW
2000	Die Reise nach Kafiristan	Kinospielfilm	diverse	Dubini Filmproduktion	
1999	Verliebt in eine Unbekannte	TV-Film	Gabriel Barylli	Network Movie	
1999	Die Einsamkeit der Krokodile	Kinospielfilm	Jobst Christian Oetzmann	Olga Film GmbH	
1999	Lola Rennt	Kinospielfilm	Tom Tykwer	X Filme Creative Pool	
1999	Der Krieger und die Kaiserin	Kinospielfilm	Tom Tykwer	X Filme Creative Pool	
1999	Tatort - Trittbrettfahrer	TV-Film (Reihe)	Markus Fischer	Colonia Media Filmproduktion	
1998	Antrag vom Ex	TV-Film	Sven Unterwaldt	Visuelle Filmproduktion	
1998	Schimanski - Geschwister	TV-Film (Reihe)	Mark Schlichter	Colonia Media Filmproduktion	
1998	Schimanski - Sehnsucht	TV-Film (Reihe)	Hajo Gies	Colonia Media Filmproduktion	
1998	Tatort - Bittere Mandeln	TV-Film (Reihe)	Kaspar Heidelbach	Colonia Media Filmproduktion	
1997	Alptraum am Airport	TV-Film	Chris Bould	Eyeworks Film Gemini	

TAL-Template für die Iteration der Tabellen-Zellen

Zunächst wird der Datentabelleninhalt ausgelesen und in die globale *res* Variable geschrieben, Mit *tal:repeat="row res"* wird über die *res*-Liste iteriert und den Iterations-Items der Name *row* gegeben. Auf dieses *repeat*-Objekt kann per Pfad zugegriffen werden *repeat/row* und z.B. auf *even/odd* geprüft werden (also *repeat/row/odd*).
Per Python-Ausdruck kann der Objektname direkt angesprochen werden, z.B: *tal:content="python: row['year']"* und ein Wert aus dem Iterations-Item ausgelesen werden.

```
<div id="projectlist"
  tal:define="global res python:zmscontext.sort_list(zmscontext.projects.attr('records'),
  request.get('qorder','year'), qorderdir=request.get('qorderdir','desc'))"
  >
  <table border="1">
    <tal:block tal:repeat="row res">
      <tal:block tal:condition="repeat/row/odd">
        <tal:block tal:define="global tr_class python:'odd'"></tal:block>
      </tal:block>
      <tal:block tal:condition="repeat/row/even">
        <tal:block tal:define="global tr_class python:'even'"></tal:block>
      </tal:block>
      <tr tal:attributes="class tr_class">
```

```
        <td tal:content="python: row['year']">year</td>
        <td tal:content="python: row['title']">year</td>
        <td tal:content="python: row['type']">year</td>
        <td tal:content="python: row['director']">director</td>
        <td tal:content="python: row['producer']">producer</td>
        <td tal:content="python: row['comment']">comment</td>
      </tr>
    </tal:block>
</table>
</div>
```

Erzeugen eines Spalten-Headers mit Item-Links bestehend aus Sortierungs-Parametern:

- *qorder* für die Variable, nach der die Datensätze sortiert werden sollen
- *qorderdir* für Richtung der Sortierung (*asc* vs. *desc*)

Die URL des Links wird per *tal:attributes* erzeugt:

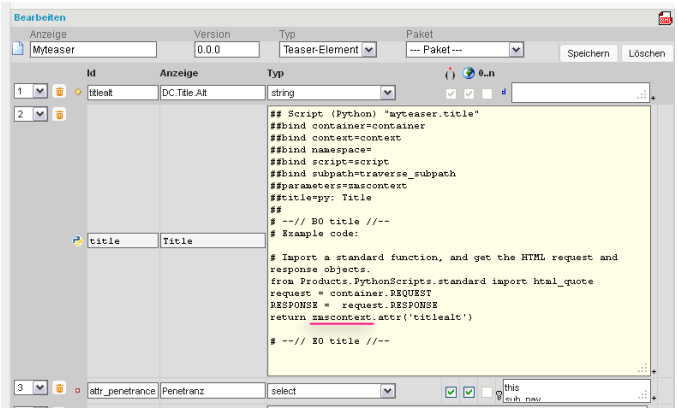
```
tal:attributes="href python:'?qorder=year&amp;qorderdir=asc'"
```

Im Detail sieht das TAL-Fragment dann so aus: *qorderdir* ist per default *desc*, ansonsten alteriert die Sortier-Richtung bei jedem Klick auf den Spalten-Header.

```
<tr>
  <th><a href="?qorder=year"
    tal:attributes="href python:'?qorder=year&amp;qorderdir=%s'
    %(['asc','desc'][request.get('qorderdir','desc')=='asc'])">Jahr</a></th>
  <th><a href="?qorder=title"
    tal:attributes="href python:'?qorder=title&amp;qorderdir=%s'
    %(['asc','desc'][request.get('qorderdir','desc')=='asc'])">Titel</a></th>
  <th><a href="?qorder=type" al:attributes="href python:'?qorder=type&amp;qorderdir=%s'
    %(['asc','desc'][request.get('qorderdir','desc')=='asc'])">Kategorie</a></th>
  <th><a href="?qorder=director" tal:attributes="href python:'?qorder=director&amp;qorderdir=%s'
    %(['asc','desc'][request.get('qorderdir','desc')=='asc'])">Regie</a></th>
  <th><a href="?qorder=producer" tal:attributes="href python:'?qorder=producer&amp;qorderdir=%s'
    %(['asc','desc'][request.get('qorderdir','desc')=='asc'])">Produktion</a></th>
  <th>Anmerkung</th>
</tr>
```

Python-Scripts als Attribut-Methoden

Um Python-Scripts als Attribut-Methoden verwenden zu können, stellt ZMS das *zmscontext*-Objekt bereit; diese ist eine ZMS-spezifische Variante des in Python-Scrips überlicherweise eingesetzten *context*-Objektes und liefert also das kontextuelle ZMS-Objekt.



Beispiel: Der Customteaser *myteaser* hat ein *titlealt*-Attribut, welches der üblicher Teaser-Container im ZMI nicht rendert, weil dieser ein titel-Attribut erwartet. Das Methoden-Attribut *title* gibt nun einfach das Bestandsattribut *titelalt* zurück mit

```
return zmscontext.attr('titlealt')
```

ZMS lagert die Python-Attributmethode wie üblich in der Objekt-Hierarchie ab:

```
/content/metaobj_manager/myteaser.title
```

Mobile-Templates

Zusätzlich zu *standard_html* kann ein alternatives Mastertemplate *mobile_html* eingesetzt werden: dieses Mastertemplate rendert automatisch die Dokumente unter einer alternativen URL; während *standard_html* den Knoten als *index_ger.html* rendert, erzeugt *mobile_html* das Dokument unter dem URL mobile_ger.html. Genau wie das *Standard*-Rendering kann das *Mobile*-Rendering auch mehrsprachige Dokumente ausgeben.

TAL-Attribute in der Übersicht

1. *tal:attributes* – setzt HTML-Attribute.
2. *tal:block* – TAL-Blockelement
3. *tal:define* – definiert Variablen.
4. *tal:condition* – prüft Bedingungen.
5. *tal:content* – ersetzt des Inhalt eines Elements.
6. *tal:omit-tag* – entfernt das Element und belässt den Inhalt, falls der Ausdruck wahr ist; ansonsten werden *tag* und *content* belassen
7. *tal:on-error* – Fehler-Behandlung
8. *tal:repeat* – Wiederholt ein Element (List Processing)
9. *tal:replace* – ersetzt das komplette Element durch den erzeugten Inhalt.

Da diese zusätzlichen Element-Attribute aus einem für den Browser fremden Namespace entstammen (bzw. in der HTML-Spezifikation nicht vorkommen), werden sie vom Browser primär ignoriert. Im Sinne der Code-Vollständigkeit kann man den TAL-Namespace im Root-Element deklarieren:

```
xmlns:tal=http://xml.zope.org/namespaces/tal
```

Beispiel: ZMS-Inhalte per TAL auslesen

Der folgende Template-Code zeigt beispielhaft wie auf einem ZMS-Knoten typische Objekte mit dem Einsatz der ZMS-API über Python-Ausdrücke als HTML gerendert werden können, konkret

1. diverse Objekt-Attribute,
2. Node-Content und
3. Child-Nodes

Dieses Zope-Template kann auf jeder ZMS-Ebene aufgerufen werden, z.B.:
http://localhost:8080/myzms/content/e3/testtemplate
Das Bildschirmfoto weiter unten zeigt eine reduzierte Version dieses Beispiel-Codes (ohne Dokumentinhalt):

```
<html
 tal:define="zmsroot here/content;
             this here;
             charsetter python:request.RESPONSE.setHeader('Content-Type','text/html;; charset=utf-8')">
<head>
<title tal:content="python:zmsroot.attr('title')"></title>
</head>
<body>
<p tal:content="python:this.attr('titlealt')"
   style="color:red;font-weight:bold"></p>
<div tal:content="structure python:this.getBodyContent(request)"></div>
<hr />
<h4>filteredChildNodes()</h4>
<ol>
 <li tal:repeat="item python:this.filteredChildNodes(request, zmsroot.PAGES)"
     tal:attributes="title python:item.attr('title')">
  <span tal:replace="python:item.attr('titlealt')">Listitem</span>
 </li>
</ol>
</body>
</html>
```

Beachte: Das Element *tal:define* kann muliple Variable definieren, die per Semikolon separiert werden; folgt nun ein Semikolon *innerhalb* einer Variablen-Defintion bzw. soll dieses nicht als delimiter fungieren, so muss dieses doppelt geschrieben werden (…'text/html;; charset…)

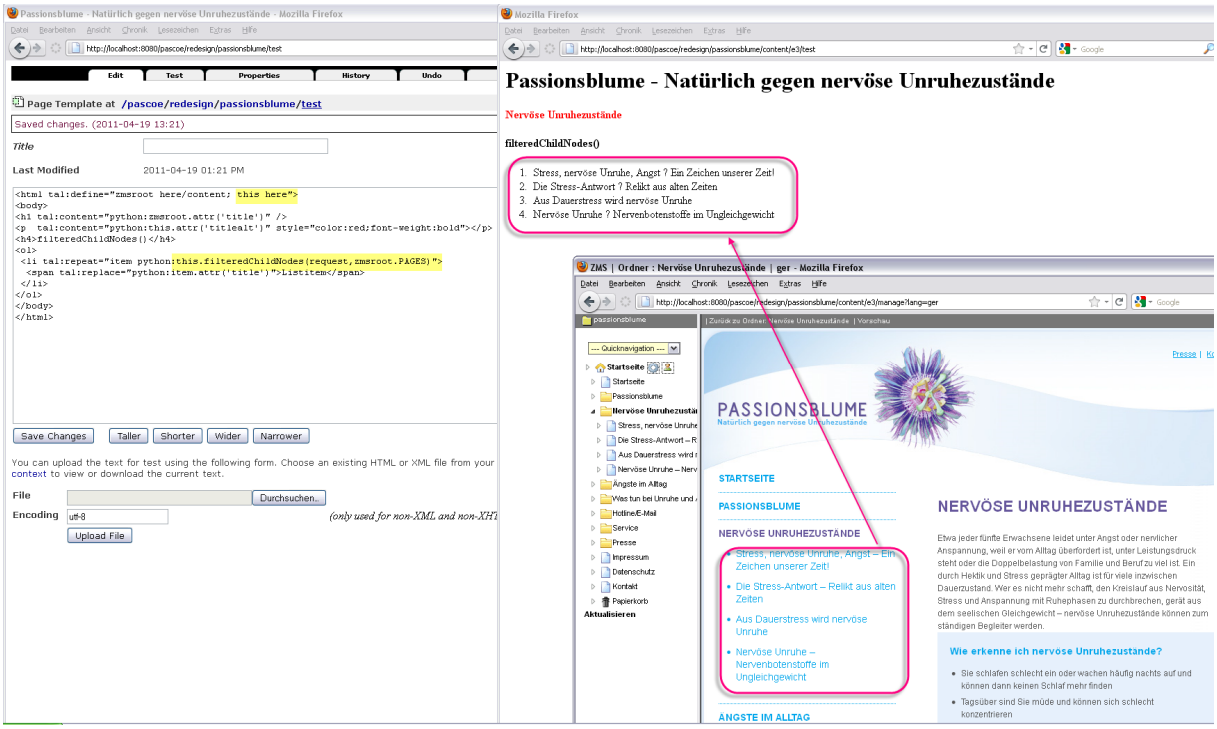


Bild: Liste der Unterobjekte per TAL-Code anzeigen

Auswertung der Ausdrücke

Die TAL-Ausdrücke eines HTML-Baumes werden von Zope bzw. TAL-Parser in der Reihenfolge "oben nach unten", also beginnend mit dem *root*-Element ausgewertet. Sofern mehrere TAL-.Ausdrücke innerhalb eines *tags* zum Einsatz kommen, kommt folgende Reihenfolge zum Einsatz:

1. *define*
2. *condition*
3. *repeat*
4. *content* oder *replace*
5. *attributes*
6. *omit-tag*

Der Inhalt des TAL-Ausdrucks kann folgende Typen annehmen:

1. *path*
2. *python*
3. *string*
4. *nocall*

Die Typ-Angabe vor dem Ausdruck (.z.B. path:a/b/c) ist nicht obligatorisch; Zope nimmt per *default* an, dass es sich um einen *Pfad*-Ausdruck handelt.

Die Deklaration als *nocall*-Ausdruck verhindert die Ausführung des Ausdrucks, wie es normalerweise der Fall bei *path*-Ausdrücken ist; hilfreich ist dies bei der Definition von Variablen:

Beispiele

path

```
<!-- A. Die URL des aktuellen Web-Requests. -->
<p tal:content="request/URL">URL</p>

<!-- B. Der Login-Name des aktuell angemeldeten Nutzers. -->
<p tal:content="user/getUserName">Username</p>

<!-- C. Liste aller IDs von Objekten im selben Ordner wie Template. -->
<li tal:reapeat="container/objectIds">Liste Item</li>

<!-- D. Context-Attribut ausgeben -->
<h1 tal:content="context/title"> Dokument-Titel </h1>
```

python

```
<span tal:content="python: 1 + 2">Resultat</span>

<h1 tal:content="python: context.title">Dokument-Titel</h1>
```

string

```
string:Hello, ${user/getUserName}
```

nocall

```
<span tal:define="page nocall:context/aPage"
      tal:content="string:${page/getId}: ${page/title}">
Id: Title</span>
```

Wichtige Built-in Names

Einige Namen sind a priori von Zope bereits besetzt und liefern folgende Objekte:

1. *context* – aktuelles Objekt, in dessen Kontext, auf den das Template angewendet wird
2. *zmscontext* – von ZMS bereitgesteller context
3. *container*- Ordner, in dem das Template platziert ist
4. *request* – Zope-REQUEST-Objekt.
5. *repeat* –Item einer sequence-Iteration
6. *default* – Vorgabe-Wert des TAL-Containers
7. *user* – eingeloggter User
8. *nothing* – Nicht-Wert entsprechend Python-*None* (bzw. void, Nil, NULL).
9. *modules* –Python Module, die zur Nutzung in den Templates freigegeben sind

Syntax-Tricks: Zope-Objekte bekommen

HTML-Code ausgeben

Beim Rendern von *tal:replace* und *tal:content* werden Sonderzeichen (z.B. >)automatisch in HTML-Entities umgewandelt (z.B. >) und HTML-Code damit aufgelöst. Um das zu hindern, wird dem TAL-Ausdruck das Keyword *structure* vorangestellt.

```
<div tal:content="structure python:childNode.getBodyContent(request)" />
```

Problem-Punkt ID-Name

Anstatt

```
python:context.penguin.gif
```

wird *getattr*-Konstrukt verwendet

```
python:getattr(context, 'penguin.gif')
```

oder ein Pfad-Ausdruck

```
path:context/images/penguin.gif
```

Aufruf einer DTML-Methode

DTML-Methoden brauchen beim Aufruf per TAL mindestens die beiden Parameter:

1. context
2. request
3. fakultativ weitere Variablen

```
<span
  tal:replace="structure python:here.my_dtml(here,request,param='hello')"
  >This will be replaced by calling a DTML method.
</span>
```

Unicode-Konflikt lösen

Unter bestimmten Firefox-Versionen kann im http-Request die ACCEPT_CHARSET-Variable fehlen; damit der Zope-Publisher aus diesem Grunde keine Fehler erzeugt, kann man diese zu Beginn des standard_html-Templates prophylaktisch setzen:

```
<html
  xmlns=http://www.w3.org/1999/xhtml
  xml:lang="de" lang="de"
  xmlns:tal=http://xml.zope.org/namespaces/tal
  tal:define="zmsroot here/content;
              this here;
              charsetter python:here.unicodeconflictresolver()">
```

```
# Py-Script unicodeconflictresolver
request = container.REQUEST
response = request.response
response.setHeader('Content-Type','text/html; charset=utf-8')
if not request.environ.get('HTTP_ACCEPT_CHARSET',None):
  request.environ['HTTP_ACCEPT_CHARSET'] = 'ISO-8859-1,utf-
8;q=0.7,*;q=0.7'
return None
```

Javascript / jquery: HTML-Erzeugung

Zope-TAL-Renderer ist extrem-empfindlich, wenn man HTML-Fragment in JS/jquery erzeugt. Man muss die spitzen Klammern dann "isolieren":

```
$('#meine_id').html('L&ouml;sung: '+myvar+' <'+'a href="..')
```

Datum via ZMS-API

Das Code-Beispiel prüft auf Vorhandensein des Attributes *date1* und gibt per *tal:content* den GER-formatierten Datumswert als Element-Content aus. Per tal:attributes wird das class-Attribute gesetzt – entweder "*date*" oder "*date expired*"; die Prüfung vergleicht die ZopeTime mit dem ENG-formatierten Attributwert von *date1*.

```
<span class="date"
  tal:condition="python:here.attr('date1')"
  tal:content="python:context.getLangFmtDate(here.attr('date1'),'ger','DATE_FMT')"
  tal:attributes="class python:['date','date expired'][context.ZopeTime( context.getLangFmtDate(
    here.attr('date1'),'eng','%Y/%m/%d')) < context.ZopeTime()]"
>01.01.2001</span>
```

DTML vs. TAL

Mehr unter http://old.zope.org/Members/peterbe/DTML2ZPT/

with-Konstrukt

DTML

```
<dtml-with "subfolder.index_html">
  <a href="<dtml-var absolute_url>" title="<dtml-var title>"
  ><dtml-var title></a>
</dtml-with>
```

ZPT

```
<a href="dummylink.html" title="Dynamic Title"
  tal:define="doc nocall:context/subfolder/index_html; title doc/title"
  tal:attributes="href doc/absolute_url; title title"
  tal:content="title">Dynamic Title</a>
```

if-else-Kondition

DTML

```
<H4>
<dtml-if "_.len(result)==0">
  <dtml-var document_title>
<dtml-else>
  Untitled
</dtml-if>
</H4>
```

ZPT

```
<H4>
  <span tal:replace="template/title"
        tal:condition="python:len(result)==0"/>
  <span tal:replace="string:Untitled"
        tal:condition="python:not(len(result)==0)"/>
</H4>
```

oder

```
<h4 tal:content="python:test(len(result)==0, default,
path('template/title'))">Untitled</h4>
```

Verwendung von Python-Modulen

url_quote-Qualifier

DTML

```
<dtml-var "'?a=b'" url_quote>
```

ZPT

```
<span tal:define="Std modules/Products/PythonScripts/standard"
      tal:replace="python:Std.url_quote('?a=b')"></span>
```

newline_to_br-Qualifier

DTML

```
<dtml-var "attr('text')" newline_to_br>
```

ZPT

```
<span tal:define="Std modules/Products.PythonScripts/standard;
    newline_to_br nocall:Std/newline_to_br;"
    tal:content="structure python:newline_to_br(context. attr('text'))">
</span>
```

Literatur

1. Using Zope Page Templates
   http://docs.zope.org/zope2/zope2book/ZPT.html
2. Advanced Page Templates
   http://docs.zope.org/zope2/zope2book/AdvZPT.html
3. Zope Page Templates Reference
   http://docs.zope.org/zope2/zope2book/AppendixC.html
4. TAL - Die Sprache der PageTemplates
   http://zope3.mpg.de/cgi-bin/twiki/view/Zope/TaL
5. TAL/TALES & METAL Reference Guide
   http://www.owlfish.com/software/simpleTAL/tal-guide.html
6. Zope Page Templates (ZPT)
   http://www.plone-entwicklerhandbuch.de/plone-entwicklerhandbuch/erscheinungsbild/zope-page-templates-zpt
7. DTML2ZPT Conversion examples
   http://old.zope.org/Members/peterbe/DTML2ZPT/
8. TAL Specification Version 1.4
   http://wiki.zope.org/ZPT/TALSpecification14
9. Zope Page Templates: Getting Started
   http://wiki.zope.org/ZPT/TutorialPart1
10. RenderErrorHandlingStrategies
    http://wiki.zope.org/ZPT/RenderErrorHandlingStrategies