

# Content-Management-Systeme

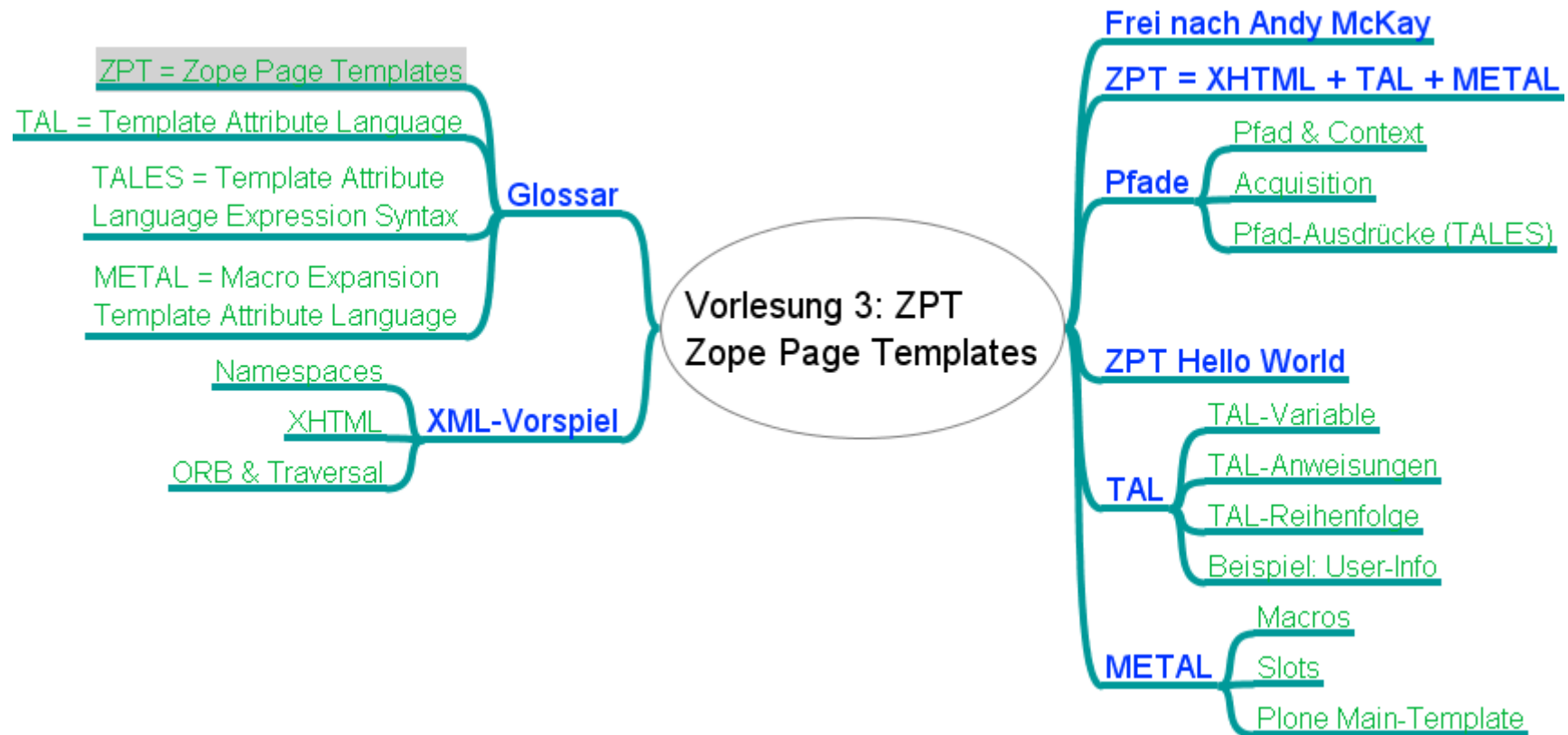
Dipl.-Inform. Roman Jansen-Winkel

Vorlesung 3: Zope Page Templates

# Inhalt und Organisation

- Übungen
  - Heute 1. Übungsblatt
  - Zope ab 2.5 reicht aus ... aktuell = 3.3.1
- Übungsaufgaben
  - **Quelle:** <http://www.satzweiss.com/VorlesungCMS>
  - Jeweils Montags ab 12:00

# Inhalt und Organisation



# Vorspiel

XML Namespaces

XHTML

# Vorspiel: XHTML

- XHTML ist HTML basierend auf XML

- Der W3C-Standard Extensible HyperText Markup Language (= XHTML) ist eine textbasierte Auszeichnungssprache zur Darstellung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.  
Es ist eine Neuformulierung von HTML 4 in XML 1.0: Im Gegensatz zu seinem Vorgänger HTML, welcher mittels SGML definiert wurde, verwendet XHTML die strengere und einfacher zu parsende SGML-Teilmenge XML als Sprachgrundlage.  
XHTML-Dokumente genügen also den Syntaxregeln von XML.

Quelle: Wikipedia, XHTML

- Wichtige Merkmale von XHTML

- XHTML beginnt mit DOCTYPE-Deklaration
- Alle Tags in Kleinbuchstaben: `<h1>...</h1>`
- Attribut-Werte in Anführungszeichen `<... color="red" ...>`
- Alle öffnenden Tags müssen geschlossen werden
- Auch leere Elemente sind zu schließen: `<br/>` oder `<br><br/>`

- <http://www.w3.org/TR/xhtml1/#xhtml>

# Vorspiel: XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD//XHTML 1.0 Transitional//EN"
    "http://www.w3.org/1999/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    lang="en" xml:lang="en">
  <head>
    <title>Beispiel-Datei mit XHTML</title>
  </head>
  <body>
    <h1>Eine &Uuml;berschrift</h1>
    <p>Und noch etwas Text dazu</p>
  </body>
</html>
```

# Vorspiel: XML-Namespaces

- XML-Namespaces; XML-Namensräume

- Erweiterung von XML um mehrere XML-Definition zu mischen

- ```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:m="http://www.w3.org/1999/Math/MathML">
  ... XHTML-Elemente ...
  <m:math>
    ... MathML-Elemente mit m:-Präfix ...
  </m:math>
  ... XHTML-Elemente ...
</html>
```

- Definition

- `xmlns=""` : Default-Namespace
- `xmlns:nsid=""`: Namespace mit Präfix *nsid*

- Anwendung

- Validator berücksichtigt eigenen Namespace; Ignoriert andere

# Vorspiel: XML-Namespaces

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg" lang="en" xml:lang="en">
  <head>
    <title>Beispiel-Datei mit mehreren Namensräumen</title>
  </head>
  <body>
    <h1>Eine Mathe-Formel:</h1>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mi>x</mi><mo>=</mo><mn>2</mn>
    </math>
    <p>Und noch ein kleines Bild dazu:</p>
    <svg:svg>
      <svg:rect x="0" y="0" width="10" height="10" />
      <svg:text>
        <svg:tspan>Eine Formel in der Grafik:</svg:tspan>
        <svg:tspan>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <mi>y</mi><mo>=</mo><mn>1</mn>
          </math>
        </svg:tspan>
      </svg:text>
    </svg:svg>
    <p svg:width="100">
      Eine SVG Grafik kann auch ohne Präfix verwendet werden:</p>
    <svg xmlns="http://www.w3.org/2000/svg">
      <circle cx="10" cy="10" r="5" fill="red" />
    </svg>
  </body>
```

# Serverbasierte HTML-Erweiterungen

Prozedural versus Attribuiert

# Ziel: Server-basierte Web-Seiten

- Standard-Webserver:  
zeigt statische Webseiten (HTML-Dateien)
- CMS erzeugt Inhalte dynamisch.
- Stilmittel:
  - HTML-Dateien erweitern;
  - Erweiterungen werden im CMS-Server interpretiert
  - Erzeugt HTML/CSS/Script-Code, der zum Client geschickt wird
  - Ergebnis wird im Client angezeigt
- Alternativen:
  - Skript erzeugt HTML mit „print“-Befehlen
  - Server generiert DHTML-Code (z.B. Javascript); Client generiert Seite

# Prozedurale HTML-Erweiterungen

- Technik analog Client-side `<script>`-Element
  - Nutzt spezielle Tags um Server-sided-Code zu signalisieren
  - Mischt Script-Code (Java, VB, C#) direkt in den HTML-Code
  - Beispiel JSP:  
`<%= new java.util.Date() %>`
  - Erweiterung: eigene Tags für Skript-Code:  
`<dtml-if>....`
- Kritik an prozeduraler Erweiterung
  - Keine Vorschau ohne Server möglich
  - WYSIWYG-Editoren neigen dazu, Skript-Code zu zerstören
  - Unleserliche Mischung aus Server-Script und Client-HTML

# Prozedurale HTML-Erweiterungen (Beispiel)

```
<HTML>
  <HEAD>
    <TITLE>hello jsp</TITLE>
    <!-- the variable, message, is declared
          and initialized -->
    <%! String message = "Hello World from JSP"; %>
  </HEAD>
  <BODY>
    <!-- the value of the variable message, is inserted between h2 tags -->
    <h2><font color="#AA0000">
      <%= message%></font></h2>
    <h3><font color="#AA0000">
      <!-- the java.util.Date method is executed
            and inserted between h3 tags -->
      <%= new java.util.Date() %>
    </font></h3>
  </BODY>
</HTML>
```

# Attributierte HTML-Erweiterungen

- Script-Attribute in eigenen Namespaces
  - XML-konforme Mischung aus XHTML und Skript
  - Wird in WYSIWYG korrekt behandelt
  - Kann in XML validiert werden
  - Bei Markup beliebter Ansatz: XLINK, XLIFF, ...
  - HTML-Preview simuliert Server-Verarbeitung
- ZOPE definiert 3 zusätzliche Namespaces
  - tal: Template Attribute Language
  - metal: Macro Expansion Template Attribute Language
  - i18n: Internationalization

# Attributierte HTML-Erweiterungen (Beispiel)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xml:lang="en" lang="en">
  <head>
    <title tal:content="template/title">The title</title>
  </head>
  <body>
    <h2><span tal:replace="here/title_or_id">
      content title or id</span>
      <span tal:condition="template/title"
            tal:replace="template/title">
        optional template title</span></h2>
    This is Page Template
    <em tal:content="template/id">template id</em>.
  </body>
</html>
```

# Zope Page Template

=

XHTML + TAL + METAL

# Pfade und Ausdrücke

## Pfade und Ausdrücke

# Pfade und Ausdrücke: Object-Request-Broker

- Zope speichert Objekte ineinander verschachtelt ab.  
Analogie zu Datei-System mit Verzeichnissen und Dateien

- URL adressiert Inhalte über einen (hierarchischen) Pfad  
Beispiel:

[http://www.satzweiss.com/Firma/anja3.jpg/base\\_view?searchterm=roman](http://www.satzweiss.com/Firma/anja3.jpg/base_view?searchterm=roman)

- **Zope-Object-Request-Broker:**  
bildet Pfad auf Objekt/Methode ab:

- `http:` legt http-Protokoll fest
- `//www.satzweiss.com:` der Host
- `/Firma/anja3.jpg:` der Pfad zum Objekt
- `/base_view:` optionale Methode
- `?searchterm=anja:` optionale Methoden-Parameter



# Pfade und Ausdrücke: Acquisition

- Suchen/Finden entlang Object-Hierarchie
- Beispiel 1: Pfad in normaler Objekt-Hierarchie
  - Members
    - Romanjw
      - Beispiel
        - Bild.jpg

Pfad: .../Members/Romanjw/Beispiel/Bild.jpg zeigt Bild.jpg an.

- Beispiel2: Pfad mit Acquisition in Object-Hierarchie
  - Members
    - Romanjw
      - Beispiel
        - Bild.jpg

Pfad: .../Members/Romanjw/Beispiel/Bild.jpg zeigt Bild.jpg an.

# Pfade und Ausdrücke: Acquisition

- Beispiel2: Pfad mit Acquisition in Object-Hierarchie

- Members
  - Romanjw
    - Beispiel
  - Bild.jpg

Pfad: ../Members/Romanjw/Beispiel/Bild.jpg zeigt Bild.jpg an.

- Context versus Container

- Context von Bild.jpg = ../Members/Romanjw/Beispiel/
- Container von Bild.jpg = ../Members/

- Bei Acquisition gilt im Normalfall:

- Context  $\neq$  Container

# Pfade und Ausdrücke: Pfadausdrücke

- Mit Pfad-Ausdrücken adressiert man Objekte und Methoden
  - Beginnt mit einer oder mehreren Variablen
  - Getrennt durch „/“
  - Gefolgt von einer oder mehreren Object-Ids
- Beispiele:
  - context/message
  - container/ordnerA/title
  - context/Members/Romanjw/Beispiel/Bild.jpg
  - context/Members/Romanjw/Beispiel/Bild.jpg/base\_view

# Pfade und Ausdrücke: Operatoren

## Pfade kann man mit Operatoren zu Ausdrücken erweitern

- „|“ = „oder“-Operator
  - Testet von links nach rechts, ob Pfad existiert. Erster existierender Pfad wird ausgegeben und testen stoppt.
  - Beispiel: context/message | container/ordnerA/title | nothing
- „not:“-Operator
  - Testet, ob Pfad existiert, invertiert den Test und gibt aus, falls Ergebnis wahr ist
  - Beispiel: not :context/message | nothing
- „nocall“-Operator
  - Zope „rendert“ das Ergebnis des Pfads implizit nach HTML.
  - Will man mit dem Pfad das „echte“ Objekt und nicht seine HTML-Darstellung
  - Beispiel: nocall: context/Members/Romanjw/Beispiel/Bild.jpg

# Pfade und Ausdrücke: Operatoren

## Weitere Operatoren

- „string:“-Operator
  - Flexible Kombination aus Ausdrücke und Konstanten
  - Beispiel: string: Ein Beispiel mit ID `${context/id}` und Titel `${context/Title | nothing}`
- „python:“-Operator
  - Schnittstelle zur „darunter“ liegenden Programmier-Ebene
  - Beispiele:  
python: 5+3  
python: context.getId()[0:-2]  
python: test(1-1, 'wahr', 'siebzehn')
- „structure“-Präfix
  - Verhindert, das HTML-Text gerendert wird und übernimmt ihn, wie er ist
  - Beispiel: structure context/description

## Dive Into ZPT

Erstes Beispiel: „showme.pt“

# Dive into ZPT: der Code

## Das Template „showme“

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xml:lang="en" lang="en">
  <head>
    <title tal:content="context/title">The title</title>
  </head>
  <body>
    <h2><span tal:replace="context/title_or_id">the id</span>
      in
      <span tal:replace="container/title_or_id">
        the container id</span></h2>
    <h6 tal:condition="here/title">
      Titel: <span tal:replace="here/title">
        optional context title</span></h6>
    This is Page Template
    <em tal:content="template/id">template id</em>.
  </body>
</html>
```

# Dive into ZPT: Preview OHNE Code



# Dive into ZPT: Objekt um „showme“ anzuwenden

The screenshot shows a Mozilla Firefox browser window with the address bar at `http://vorlesungcms.satzweiss.com/Members/romanjw/sondermeldung`. The page title is "Sondermeldung — Demo-Site fuer CMS-Vorlesung - Mozilla Firefox". The website header includes the logo "satzweiss.com" and navigation links like "Website-Übersicht", "Barrierefreiheit", "Kontakt", and "Konfiguration". A search bar is present with the text "Suche".

The main content area is titled "Sondermeldung" and includes the following text:

- erstellt von [Roman Jansen-Winkel](#) zuletzt verändert: 04.11.2007 22:37
- Mitwirkende: Elke Weiss, Anja Mechenbier
- Historie**
- Medieninhalte kann man mit Plone verwalten**
- Medieninhalte**
- Basistypen**  
Plone kennt eine Reihe vordefinierter Dokumenttypen
- Seite
- Datei
- Bild
- Ordner
- Link
- Termin
- Nachricht
- Intelligenter Ordner (= Filter)

An inset diagram titled "Ausschnitt Plone Document Types" shows a tree structure for "Plone Content Types" with the following items:

- Seite
- Datei
- Bild
- Ordner
- Link
- Termin
- Nachricht
- Filter

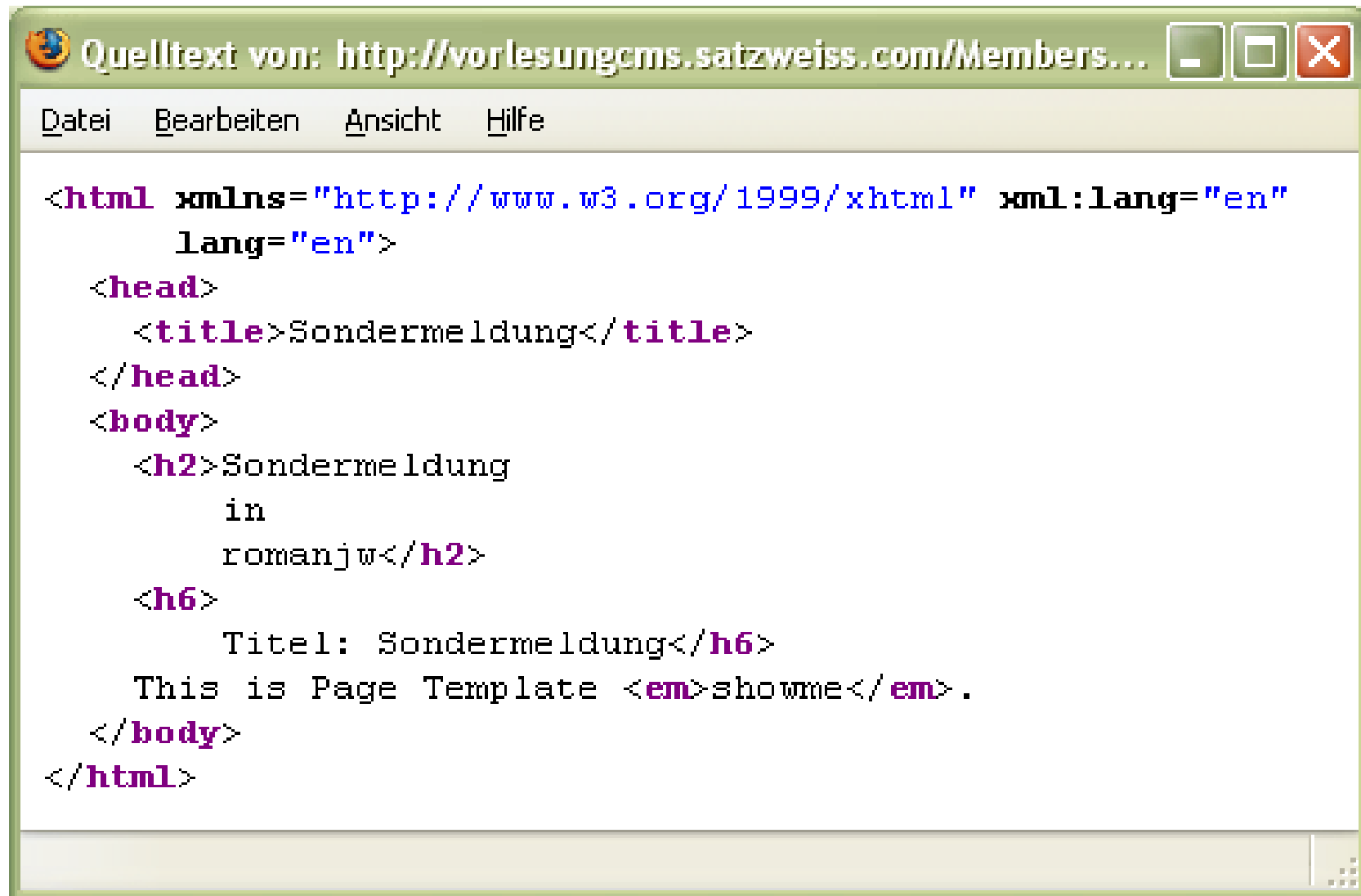
The browser's status bar at the bottom shows "Fertig", a search field, and system information including "1.470s" and "N/A".

# Dive into ZPT: Anwendung von „showme“



<http://vorlesungcms.satzweiss.com/Members/romanjw/sondermeldung/showme>

# Dive into ZPT: Quelltext „showme“ angewendet



```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
  lang="en">
  <head>
    <title>Sondermeldung</title>
  </head>
  <body>
    <h2>Sondermeldung
      in
      romanjw</h2>
    <h6>
      Titel: Sondermeldung</h6>
    This is Page Template <em>showme</em>.
  </body>
</html>
```

<http://vorlesungcms.satzweiss.com/Members/romanjw/sondermeldung/showme>

## Template Attribute Language

Die Details

# TAL: Vordefinierte Variable

- container
- context = here
- default: Standardmethode am Objekt
- modules = Containerobjekt für Zope-Programmmodule
- nothing analog python:None
- options = Aufrufparameter
  - options/searchForm
- repeat = loop
  - Wiederholungsobjekt zu tal:repeat
  - request/entry/index
- request = Environmentobjekt
- root = Wurzelobjekt
- template = Containerobjekt des aufgerufenen Templates
- traverse\_subpath
- user = aktuelles Userobjekt
  - user/getUsername

# TAL: größeres Beispiel

```
<html>
  <head />
  <body>
    <h1>Debug information</h1>
    <h2>CONTEXTS</h2>
    <ul>
      <tal:block tal:repeat="item CONTEXTS">
        <li tal:condition="python: item != 'request'"
            tal:define="context CONTEXTS;">
          <b tal:content="item" />
          <span tal:replace="python: context[item]" />
        </li>
      </tal:block>
    </ul>
    <h2>REQUEST</h2>
    <p tal:replace="structure request" />
  </body>
</html>
```

# TAL: größeres Beispiel

## Debug information

### CONTEXTS

- **repeat** <SafeMapping instance at 3caa8b0>
- **container** <PloneSite instance at 3c75d70>
- **template** <ZopePageTemplate at test>
- **default**
- **modules** <Products.PageTemplates.ZRPythonExpr.\_SecureModuleImporter instance at 0x23eed28>
- **here** <PloneSite instance at 3c75d70>
- **traverse\_subpath** []
- **user** admin
- **context** <PloneSite instance at 3c75d70>
- **nothing**
- **root** <Application instance at 3c757a0>
- **options** {'args': ()}
- **loop** <SafeMapping instance at 3caa8b0>

### REQUEST

#### form

#### cookies

```
__cp 'x%DA%D3%60b%60%60%C8%04b%86hF%20%A1%C1%02%24%8AA%DCbV%20%11%90%  
wstyle "  
_ZopeId '98022352A1spjpZ9IQA'
```

#### lazy items

# TAL: die TAL-Anweisung im Einzelnen

- **tal:attributes = Element-Attribute ändern**

- Beispiel:

```
<a href="#" title="#"  
  tal:attributes="href context/absolute_url ; title=context/description">Link</a>
```

- **tal:content = Text / Children hinzufügen**

Der Wert der Expression wird zum Inhalt des Elements

- Beispiel:

```
<em tal:content="context/title_or_id">Titel hervorgehoben</em>
```

- **tal:replace = Element ersetzen**

Der Wert der Expression wird an Stelle des Elements verwendet

- Beispiel:

```
<span tal:content="context/title_or_id">Titel hervorgehoben</span>
```

# TAL: die TAL-Anweisung im Einzelnen

- **tal:condition = Bedingungen auswerten**

Element inkl. Unterelementen erscheint nur, wenn tal:condition zu 'wahr' evaluiert

- Beispiel:

- <p tal:condition="request/message">Es gibt eine message</p>

- <p tal:condition="not: request/message">Es gibt keine message</p>

- **tal:omit-tag = Entfernt den Tag**

Konditionales Entfernen eines Elements unter Beibehaltung der Children

- Beispiel:

- <a href='...' tal:omit-tag="context/title">Titelseite abrufen</a>

- Rendert „Titelseite abrufen“ als Link, wenn ein Titel angegeben ist, sonst als normalen Text

- **tal:on-error: Catch für Fehlerbehandlung**

- Beispiel:

- <p tal:content="request/message"

- tal:on-error="string: No Message">Message</p>

# TAL: die TAL-Anweisung im Einzelnen

- **tal:define** = definiert lokale Variable im Element und Children

- Beispiel:

```
<p tal:define="ctit string: Titel=${context/title}">  
  ... <em tal:content="ctit">Der formatierte Titel</em> ...  
</p>
```

- **tal:repeat** = Schleife über Liste mit Schleifenvariablen

- Beispiel:

```
<table>  
  <tr tal:repeat="row context/portal_catalog">  
    <td tal:content="row/id">ID:</td>  
    <td tal:content="row/Title">Titel</td>  
  </tr>  
</table>
```

- Zusätzliches Schleifenobjekt, z.B. repeat/row/... mit

- repeat/.../index, repeat/.../number, repeat/.../even, repeat/.../odd,  
 repeat/.../start, repeat/.../end, repeat/.../length,
- repeat/.../letter, repeat/.../Letter, repeat/.../roman

# TAL: die TAL-Anweisung im Einzelnen

- tal:repeat : ausführliches Beispiel

```
<ul>
  <li tal:repeat="val context/objectValues">
    First: <i tal:content="repeat/val/first/meta_type" />,
    Last: <i tal:content="repeat/val/last/meta_type" />:
    <b tal:content="val/meta_type" />,
    <b tal:content="val/title_or_id" />
  </li>
</ul>
```

# TAL: Ausführungsreihenfolge

- Stehen mehrere tal-Attribute an einem Element, so werden sie in folgender Reihenfolge ausgewertet:
  - 1) define
  - 2) condition
  - 3) repeat
  - 4) content
  - 5) replace
  - 6) attributes
  - 7) omit-tag
- on-error: bei Bedarf

## Macro Expansion for Template Attribute Language

Die Details

- Makro-Prozessor für TAL
  - eigener Namespace „metal:“
  - erlaubt Wiederverwertung von TAL-Fragmenten
  - Makro-Prozessor  $\approx$  Textuelle Ersetzung
- Zwei Anwendungen: Makros und Slots
- Makro:
  - Makro definieren
  - Makro aufrufen
- Slot:
  - Slot als Platz im Code definieren
  - Slot an anderer Stelle füllen

# METAL: Makros

- „metal:define-macro“: Makro definieren

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal">
  <body>
    <div metal:define-macro="boxA">
      ...
    </div>
    <div metal:define-macro="boxB">
      ...
    </div>
  </body>
</html>
```

- „metal:use-macro“: Makros aufrufen

```
<div metal:use-macro="context/beispiel/macros/boxA">
  Der Macro aus Beispiel oben kommt hierhin
</div>
```

# METAL: Slots

- „metal:define-slot“: Slot hält Platz frei

```
<div metal:define-macro="master">  
  <div metal:define-slot="main">  
    ...  
  </div>  
</div>
```

- „metal:fill-slot“: Spezifischer Code wird in den Slot eingesetzt

```
<div metal:use-macro="master">  
  <div metal:fill-slot="main">  
    Der Haupt-Slot kommt hierhin  
  </div>  
</div>
```

- Mit Slots kann Verhalten von Macros anpassen.
- Slots sind ähnlich zu Macro-Parametern

# METAL: Anwendung in Plone „main\_template“

```
<metal:page define-macro="master"><metal:doctype define-slot="doctype"><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<metal:block define-slot="top_slot" />
<metal:block use-macro="here/global_defines/macros/defines" />

<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en"
      lang="en"
      tal:attributes="lang language;
                    xml:lang language">

<tal:cache tal:define="lang language;
                    charset site_properties/default_charset|string:utf-8">
  <metal:cache use-macro="here/global_cache_settings/macros/cacheheaders">
    Get the global cache headers located in global_cache_settings.
  </metal:cache>
</tal:cache>

<head metal:use-macro="here/header/macros/html_header">

  <metal:fillbase fill-slot="base">
    <metal:baseslot define-slot="base">
      <base href="" tal:attributes="href here/renderBase" />
    </metal:baseslot>
  </metal:fillbase>

  <metal:headslot fill-slot="head_slot">
    <metal:headslot define-slot="head_slot" />
    <tal:comment replace="nothing"> A slot where you can insert elements in the header from a template </tal:comment>
  </metal:headslot>

  <metal:styleslot fill-slot="style_slot">
    <tal:comment replace="nothing"> A slot where you can insert CSS in the header from a template </tal:comment>
    <metal:styleslot define-slot="style_slot" />
  </metal:styleslot>

  <metal:csssot fill-slot="css_slot">
    <tal:comment replace="nothing"> This is deprecated, please use style_slot instead. </tal:comment>
    <metal:csssot define-slot="css_slot" />
  </metal:csssot>
```

# METAL: Anwendung in Plone „main\_template“

```
<metal:cssslot fill-slot="css_slot">
  <tal:comment replace="nothing"> This is deprecated, please use style_slot instead. </tal:comment>
  <metal:cssslot define-slot="css_slot" />
</metal:cssslot>

<metal:javasscriptslot fill-slot="javascript_head_slot">
  <tal:comment replace="nothing"> A slot where you can insert javascript in the header from a template </tal:comment>
  <metal:javasscriptslot define-slot="javascript_head_slot" />
</metal:javasscriptslot>
</head>

<body tal:attributes="class here/getSectionFromURL;
  dir python:test(isRTL, 'rtl', 'ltr')">
  <div id="visual-portal-wrapper">

    <div id="portal-top" i18n:domain="plone">

      <div id="portal-header">
        <p class="hiddenStructure">
          <a accesskey="2"
            tal:attributes="href string:${current_page_url}#documentContent"
            i18n:translate="label_skiptocontent">Skip to content.</a> |

          <a accesskey="6"
            tal:attributes="href string:${current_page_url}#portlet-navigation-tree"
            i18n:translate="label_skiptonavigation">Skip to navigation</a>
        </p>

        <div metal:use-macro="here/global_siteactions/macros/site_actions">
          Site-wide actions (Contact, Sitemap, Help, Style Switcher etc)
        </div>

        <div metal:use-macro="here/global_searchbox/macros/quick_search">
          The quicksearch box, normally placed at the top right
        </div>

        <a metal:use-macro="here/global_logo/macros/portal_logo">
          The portal logo, linked to the portal root
        </a>
      </div>
    </div>
  </div>
</body>
```

# METAL: Anwendung in Plone „main\_template“

```
<a metal:use-macro="here/global_logo/macros/portal_logo">
  The portal logo, linked to the portal root
</a>

<div metal:use-macro="here/global_skinswitcher/macros/skin_tabs">
  The skin switcher tabs. Based on which role you have, you
  get a selection of skins that you can switch between.
</div>

<div metal:use-macro="here/global_sections/macros/portal_tabs">
  The global sections tabs. (Welcome, News etc)
</div>
</div>

<div metal:use-macro="here/global_personalbar/macros/personal_bar">
  The personal bar. (log in, logout etc...)
</div>

<div metal:use-macro="here/global_pathbar/macros/path_bar">
  The breadcrumb navigation ("you are here")
</div>
</div>

<div class="visualClear" id="clear-space-before-wrapper-table"><!-- --></div>

<tal:comment replace="nothing">
The wrapper table. It contains the three columns. There's a table-less
alternative in the plone_tableless skin layer that you can use if you
prefer layouts that don't use tables.
</tal:comment>

<table id="portal-columns">
  <tbody>
    <tr>
      <tal:comment replace="nothing"> Start of the left column </tal:comment>
      <td id="portal-column-one"
        metal:define-slot="column_one_slot"
        tal:condition="sl">
        <div class="visualPadding">
          <metal:portlets define-slot="portlets_one_slot">
```

# METAL: Anwendung in Plone „main\_template“

```
<!--condition= "1" -->
<div class="visualPadding">
  <metal:portlets define-slot="portlets_one_slot">
    <metal:leftportlets use-macro="here/portlets_fetcher/macros/left_column">
      This instruction gets the portlets (boxes) for the left column.
    </metal:leftportlets>
  </metal:portlets>
  &nbsp;
</div>
</td>
<tal:comment replace="nothing"> End of the left column </tal:comment>

<tal:comment replace="nothing"> Start of main content block </tal:comment>
<td id="portal-column-content"
  tal:define="tabindex python:Iterator(pos=0, mainSlot=True)">

  <metal:block define-slot="content">
    <div id="content"
      metal:define-macro="content"
      tal:define="show_border python:here.showEditableBorder(template_id=template_id, actions=actions);"
      tal:attributes="class python:test(show_border,'documentEditable',')">

      <metal:ifborder tal:condition="show_border" >
        <div metal:use-macro="here/global_contentviews/macros/content_views">
          The content views (View, Edit, Properties, Workflow)
        </div>

        <div metal:use-macro="here/global_contentviews/macros/content_actions">
          The content bar
        </div>
      </metal:ifborder>

      <div class="documentContent" id="region-content">

        <a name="documentContent"></a>

        <div metal:use-macro="here/global_statusmessage/macros/portal_message">
          Portal status message
        </div>

        <metal:header metal:define-slot="header" tal:content="nothing">
```

# METAL: Anwendung in Plone „main\_template“

```
<metal:header metal:define-slot="header" tal:content="nothing">
  Visual Header
</metal:header>

<metal:bodytext metal:define-slot="main" tal:content="nothing">
  Page body text
</metal:bodytext>

<metal:sub metal:define-slot="sub">
  <metal:discussion use-macro="here/viewThreadsAtBottom/macros/discussionView" />
</metal:sub>

</div>

</div>

</metal:block>
</td>
<tal:comment replace="nothing"> End of main content block </tal:comment>

<tal:comment replace="nothing"> Start of right column </tal:comment>
<td id="portal-column-two"
  metal:define-slot="column_two_slot"
  tal:condition="sr">
  <div class="visualPadding">
    <metal:portlets define-slot="portlets_two_slot">
      <metal:rightportlets use-macro="here/portlets_fetcher/macros/right_column">
        This instruction gets the portlets (boxes) for the right column.
      </metal:rightportlets>
    </metal:portlets>
    &nbsp;
  </div>
</td>
<tal:comment replace="nothing"> End of the right column </tal:comment>
</tr>
</tbody>
</table>
<tal:comment replace="nothing"> end column wrapper </tal:comment>

<div class="visualClear" id="clear-space-before-footer"><!-- --></div>
```

# METAL: Anwendung in Plone „main\_template“

```
<div class="visualClear" id="clear-space-before-footer"><!-- --></div>

<hr class="netscape4" />

<metal:block i18n:domain="plone">

  <metal:footer use-macro="here/footer/macros/portal_footer">
    Footer
  </metal:footer>

  <metal:colophon use-macro="here/colophon/macros/colophon">
    The colophon area - contains details about the production of
    the site. Typically "powered by" buttons, standards, tools used.
  </metal:colophon>
</metal:block>

</div>

</body>
</html>
</metal:page>
```

Portal-Aufbau im Detail

Python-Skripts

Eigene Dokumenttypes programmieren: Archetypes