Inhaltsverzeichnis

1 Einführung

1.1 TAL - Template Attribute Language

- ist die Sprache, in der alle Templates definiert werden (ein Template ist ein Muster zur Darstellung der Inhalte)
- ist eine XML Sprache
- Namensraum: xmlns=http://xml.zope.org/namespaces/tal

1.1.1 Die Anweisungen

- tal:attributes - Attribute eines Elements (Tags) modifizieren
- tal:define - Definition von Variablen
- tal:condition - Bedingungen definieren
- tal:content - den Inhalt eines Elements bestimmen
- tal:omit-tag - Element (Tag) wird entfernt, nicht jedoch der Inhalt
- tal:on-error - Fehlerbehandlung
- tal:repeat - Wiederholung / Schleife
- tal:replace - Element wird enfernt, nicht aber der definierte Inhalt

1.1.2 Besonderheiten

- Reihenfolge der Interpretation
  - define
  - condition
  - repeat
  - content oder replace
  - attributes
  - omit-tag
- gegenseiteiger Ausschluß:
  - content und replace können nicht gemeinsam innerhalb eines Tags genutzt werden

Listing 1: Syntax TAL

```
< html-tag tal:talAnweisung="Argument[; Argument]*" >
```

1.2 TALES - Template Attribute Language Expression Syntax

- Namensraum: xmlns=http://xml.zope.org/namespaces/tales

Listing 2: Syntax TALES

```
Ausdruck    ::= [typpraefix ':'] Zeichenkette
typpraefix ::= Name
```

Typpräfixe:

- exists: (zum Testen, ob eine Variable existiert)
- not: (negativiert den nachfolgenden Ausdruck)
- string: (Zeichenkette)
- python: (Pythonausdruck)

1.2.1 Beispiele

Listing 3: Beispiele TALES

```
a/b/c
path:a/b/c
nothing
path:nothing
python: 1 + 2
python: a.b.c()
string:Hallo ${user/getUserName}
```

1.2.2 Feststehende Namen (Auswahl)

- nothing - Nicht-Wert (void, None, NULL).
- options - Argumente, die der Vorlage übergeben wurden
- repeat - die repeat-Variablen
- root - der Zope-Wurzelordner
- here - das Objekt auf das die Vorlage gerade angewandt wird
- container - Der Ordner in dem sich die Vorlage befindet
- template - Die Vorlage selbst
- request - das Objekt der Veröffentlichungsanfrage
- user - das Objekt des autentifizierten Benutzers
- modules - eine Sammlung durch die Python-Module und -Pakete angesprochen werden können

2 TAL

2.1 tal:attributes - Attribute modifizieren

- ersetzt den Wert eines Attributes oder erzeugt dieses.

Listing 4: Beispiele tal:attributes

```
<a href="/einLinkNachHier"
    tal:attributes="href here/absolute_url">
    Hier
</a>

<a href="/einLinkNachHier"
    tal:attributes="href here/absolute_url;
                    title here/Title">
    Hier
</a>
```

2.2 tal:content - Inhalte setzen

Listing 5: tal:content

```
<a href="/einLinkNachHier"
   tal:attributes="href here/absolute_url;
                   title here/title_or_id"
   tal:content="here/title_or_id">
   Hier
</a>
<p tal:content="string: Der aktuelle Nutzer ist ${user/getUserName}" >
   der Inhalt des p-Tags
</p>

<div tal:content="structure here/eineVariable">
   structure bewirkt eine Interpretierung des html-textes,
   der eingebunden werden soll
</div>

<p tal:content="python:here.getEssen() or default">Dosenfleisch</p>
```

2.3 tal:replace - Inhalte setzen und Tag entfernen

- replace und content schließen einander aus
- replace und attributes zusammen in einem Tag macht keinen Sinn, da die Attribute entfernt werden

Listing 6: Beispiele tal:replace

```
<p tal:replace="string: Der aktuelle Nutzer ist ${user/getUserName}" >
   der Inhalt des p-Tags
</p>
```

2.4 tal:condition - Bedingungen testen

- wenn der Ausdruck der Condition wahr ist, dann wird der Inhalt verarbeitet, sonst wird der Inhalt aus dem erstellten Dokument entfernt
- 0(Null), None, leere Zeichenketten, Tupel und Listen werden als " falsch " interpretiert

Listing 7: tal:condition

```
<p tal:condition="python: user.getUserName() != 'Anonymous'"
   tal:replace="string: Der aktuelle Nutzer ist ${user/getUserName}" >
      der Inhalt des p-Tags
</p>
<p tal:condition="python: user.getUserName() == 'Anonymous'">
   Sie sind nicht eingeloggt.
</p>

<!-- Auskommentierung von Inhalten -->
<div tal:condition="nothing">
   Dieses Tag wird nie erscheinen, da die Bedingung nie wahr wird.
</div>
```

```
<!-- Variablenexistenz testen -->
<div tal:condition="exists: request/nachricht">
   <p tal:content="request/nachricht">
      Der Inhalt der Nachricht
   </p>
</div>
```

2.5 tal:define - Variablen definieren

- Es gibt 2 Arten von Variablen:
  - lokal - diese Variablen existieren nur innerhalb des umschließenden Tags
  - global - diese Variable läßt sich in allen Elementen benutzen, die nach dem definierendem Element interpretiert werden
- ohne Angabe von lokal oder global handelt es sich um lokale Variablen

Listing 8: tal:define

```
<!-- lokal -->
<div tal:define="meineVariable python: 1">
   <span tal:replace="meineVariable">
      meineVariable
   </span>
   <span tal:define="meineVariable python: 2">
      <span tal:replace="meineVariable">
         meineVariable
      </span>
   </span>
</div>

<!-- global -->
<div tal:define="global meineNeueVariable python: 1">
     Hier ist irgendein Inhalt
</div>

<p tal:replace="meineNeueVariable">
   meineNeueVariable
</p>
```

2.6 tal:omit-tag - Element entfernen (Inhalt bleibt)

Listing 9: tal:omit-tag

```
<p tal:omit-tag="">
   Inhalt des Tags, der bestehen bleibt.
</p>

<!-- Entfernen, wenn Bedingung wahr ist -->
<b tal:omit-tag="not: exists: request/fett">
   Inhalt des Tags, der bestehen bleibt.
</b>
```

2.7 tal:repeat - Schleifen / Wiederholungen

- es wird das eingeschlossene Tag für jedes Element der Sequenz wiederholt
- ist die Sequenz leer, wird auch das Tag entfernt

Listing 10: tal:repeat(1)

```
<ul>
   <li tal:repeat="zahl python: range(5)">
        <p tal:replace="zahl">1</p>
   </li>
</ul>
<ul>
   <li tal:repeat="zahl python: range(0,0)">
        <p tal:replace="zahl">1</p>
   </li>
</ul>
```

2.7.1 Wiederholungsvariablen

- index - Anzahl der Wiederholungen, beginnend bei Null.
- number - Anzahl der Wiederholungen, beginnend bei Eins.
- even - ist true bei Wiederholungen mit geradem Index (0, 2, 4, ...).
- odd - ist true bei Wiederholungen mit ungeradem Index (1, 3, 5, ...).
- start - ist true beim Anfangsdurchlauf (Index 0).
- end - ist true beim End-, also beim letzten Durchlauf.
- first - ist true beim ersten Element einer Gruppe (Achtung: nur für sortierte Sequenzen)
- last - ist true beim letzten Element einer Gruppe
- length - Länge der Sequenz, was gleichzeitig auch der Gesamtzahl der Wiederholungen entspricht.
- letter - Anzahl der Wiederholungen als Kleinbuchstabe: 'a' - 'z', 'aa' - 'az', 'ba' - 'bz' ...
- Letter - Genau wie letter, nur mit Großbuchstaben.
- roman - Anzahl der Wiederholungen kleine römische Zahlen: 'i', 'ii', 'iii', 'iv', 'v', etc.
- Roman - Genau wie roman, nur mit Großbuchstaben.

Listing 11: tal:repeat(2)

```
<ul>
   <li tal:repeat="item python: range(5)">
        <b tal:content="item">1</b><br />
        <p tal:replace="string: index:  ${repeat/item/index}"> 1</p> <br />
        <p tal:replace="string: number: ${repeat/item/number}">1</p> <br />
        <p tal:replace="string: letter: ${repeat/item/letter}">1</p> <br />
        <p tal:replace="string: Letter: ${repeat/item/Letter}">1</p> <br />
        <p tal:replace="string: roman:  ${repeat/item/roman}"> 1</p> <br />
        <p tal:replace="string: Roman:  ${repeat/item/Roman}"> 1</p> <br />
   </li>
</ul>
```

```
<!-- Conditions -->
<ul>
   <li tal:repeat="item python: range(5)">
      <b tal:define="isEven repeat/item/even"
         tal:content="item"
         tal:attributes="style
                         python:test(isEven,'color: red','color: blue')">
         1
      </b><br />
      <b tal:condition="repeat/item/start">
         das ist der erste</b><br />
      <b tal:condition="repeat/item/end">
         das ist der letzte</b><br />
   </li>
</ul>
```

2.8 tal:on-error - Fehlerbehandlung

- ermöglicht Fehlerbehandlung in der Vorlage
- Attribute der lokalen Variable “error“
  - type
  - value
  - traceback (darf aber von “ nicht-vertrauenswürdigem Code “ nicht aufgerufen werden)

Listing 12: tal:on-error

```
<b tal:on-error="string: Benutzername ist nicht definiert!"
         tal:content="here/getUsername">Ishmael</b>
<b tal:on-error="string: value:${error/value} type:${error/type}"
   tal:content="here/getUsername">Ishmael</b>
```

3 METAL - Macro Expansion Template Attribute Language

- dient zur Vorverarbeitung von HTML/XML-Makros
- ein Stück zur Darstellung in einer Vorlage wird definiert und mit anderen Vorlagen geteilt
- Änderungen am Makro wirken sich auf alle Vorlagen aus, die das Makro nutzen
- METAL-Anweisungen
  - metal:define-macro - Ein Makro definieren.
  - metal:use-macro - Ein Makro verwendet.
  - metal:define-slot - Einen anpassbaren Makropunkt definieren.
  - metal:fill-slot - Ein Makro anpassen.

3.1 define-macro: ein Makro definieren

Listing 13: metal:define-macro

```
<p metal:define-macro="adresse">
   <a href="mailto:hans@mustermann.de">Hans Mustermann</a>,
   Telefon: 1234567,
   Musterstrasse 123,
   12345 Musterhausen
</p>
```

3.2 use-makro: ein Makro benutzen

Listing 14: metal:use-macro

```
<p metal:use-macro="template/macros/adresse">
   hier stehr die Adresse
</p>
```

3.3 define-slot: einen anpassbaren Makropunkt definieren

Listing 15: metal:define-slot

```
<p metal:define-macro="anrede">
   Hallo <b metal:define-slot="name">Hans Mustermann</b>
</p>
```

3.4 fill-slot: ein Makro anpassen

Listing 16: metal:fill-slot

```
<p metal:use-macro="template/macros/anrede">
   <b metal:fill-slot="name">Kevin Bacon</b>
</p>
```

4 Die Praxis - Plone

4.1 Generelle Funktionsweise der Templates

- es gibt ein generelles Template zur Ansicht, das " main_template "
- das ist ein großes Makro, in dem viele verschiedene Slots definiert sind
- ein Inhaltstyp weiß von sich, welche(s) Template(s) für seine Ansicht definiert sind
- in dem Template zur Ansicht eines Inhaltstyps wird nun das Makro aus dem main_template gerufen und verschiedene Slots darin gefüllt (normalerweise mindestens "main")

4.2 Beispiel: Vorlage für Dokumente

Listing 17: document-view

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
    lang="en"
    metal:use-macro="here/main_template/macros/master"
    i18n:domain="plone">
<body>
<metal:main fill-slot="main">
    <tal:main-macro metal:define-macro="main"
     tal:define="len_text python:len(here.text);">
       <div metal:use-macro="here/document_firstHeading/macros/firstHeading">
           title
       </div>
       <div metal:use-macro="here/document_actions/macros/document_actions">
           Document actions (print, sendto etc)
       </div>
       <div class="contentWithLeftPadding">
          <div class="documentDescription"
             tal:content="structure here/Description">
             description
          </div>
          <p tal:condition="python: not len_text and is_editable"
             i18n:translate="no_body_text"
             class="discreet">
             This item does not have any body text,
             click the edit tab to change it.
          </p>
          <div class="stx"
             tal:condition="len_text"
             tal:attributes="class python:
                test(here.Format() in ('text/structured','text/x-rst',),
                   'stx', 'plain')">
             <div tal:replace="structure
                   python:here.CookedBody(stx_level=2)" />
          </div>
       </div>
    </tal:main-macro>
</metal:main>
<metal:block fill-slot="hu_subslot">
    <div metal:use-macro="here/document_byline/macros/byline">
       Get the byline - contains details about author and modification date.
    </div>
</metal:block>
</body>
</html>
```

4.3 Eine Eigene Ansicht für Dokumente Definieren

- document_view per "customize" in den custom-Ordner kopieren
- umbenennen in document_view_NACHNAME
- eintragen als gültiges Template

  - unter “archetype_tool“ → “templates“ ganz unten die Id des neuen Templates hinzufügen
  - nun bei “ATContentTypes:ATDocument“ das Template mit anwählen
- nun ist das neue Template verfügbar und kann den eigenen Ansprüchen entsprechend angepasst werden.

4.3.1 Dokument-Ansicht mit Auflistung der aktuellen News (Nachrichten)

Ein Ausflug in die Struktur von Plone (2) - Internationalisierung (i18n)

- Namensraum i18n
- dahinter verbirgt sich nichts weiter als eine grosse Liste von von id-str-tupeln
- den entsprechenden Listen wurde mitgegeben, welcher Domain sie angehören (siehe eigene Produkte)
- die Ausdrücke:
  - domain
  - translate
  - attributes
  - name

Listing 18: i18n:translate

```
<!-- ohne Uebersetzung -->
<i>Hier steht ein Text</i>

<!-- mit Uebersetzung -->
<i i18n:tranlate="label_irgendetwas">Hier steht ein Text</i>

<h1 tal:content="here/title"
    i18n:translate="">
     Der Titel
</h1>
```

Mit der **domain** kann angegeben werden, aus welchem Raum die Übersetzung genommen wird.

Listing 19: i18n:domain

```
<body i18n:domain="plone">
<body i18n:domain="meinProdukt">
```

Mit der **attributes** können Attribute eines Tags übersetzt werden.

Listing 20: i18n:attributes

```
<img src="myProduct.gif"
   alt="MyProduct Icon"
   title="MyProduct Icon Title"
   i18n:attributes="alt myproduct_icon;
                    title title_myproduct_icon">
```

Mit **name** widerum lassen sich einzelne feste Elemente innerhalb einer Übersetzung definieren.

Listing 21: i18n:name

```
<!-- im Quelltext -->
<p i18n:translate="batch_downloads">There have been
      <span tal:content="here/download_count"
```

```
            i18n:name="count">
                100,000
        </span>
        downloads.
</p>

<!-- die Definition (.po - Datei)-->
<!-- en -->
msgid "batch_downloads"
msgstr "There have been ${count} downloads."
<!-- de -->
msgid "batch_downloads"
msgstr "Anzahl der Downloads: ${count}."
```

Also für die Nachrichtenauflistung:

Listing 22: Nachrichten - der Titel

```
<h5 i18n:translate="box_news">News</h5>
```

Ein Ausflug in die Struktur von Plone (1) - der Katalog

- die Inhalte der einzelnen Dokumente und ihrer Attribute werden von Plone in einem Katalog vorghehalten
- dieser Katalog ("portal_catalog") bietet bequemen und schnellen Zugriff auf die katalogisierten Inhalte
- brains: - der Katalog gibt nicht die Objekte selber zurück, sondern sogenannte brains (resultobjects)
- metadata: (Liste der Attribute, die der Katalog aufnimmt, alles worauf ich aus einem Brain zugreifen will, muss in dieser Liste stehen)
- indizies: gibt zu den Attributen an, auf welche Art und weise sie indiziert werden sollen
  - Fieldindex
  - Textindex
  - Keywordindex

Listing 23: Nachrichten - die Katalogabfrage

```
<!-- die einfachste Moeglichkeit -->
   <div tal:define="results
       here.portal_catalog.searchResults(portal_type='News Item')">
    ...
   </div>
<!-- mit beachtung des Workflows und Sortierung -->
   <div tal:define="results python: here.portal_catalog.searchResults(
       portal_type='News Item',     <!-- Nachrichten -->
       sort_on='Date',              <!-- nach Datum sortiert -->
       sort_order='reverse',        <!-- neueste zuerst -->
       review_state='published');"><!-- nur veroeffentlichte -->
       ...
   </div>
```

Und nun eine Liste der eigentlichen Nachrichten:

Listing 24: Nachrichten - die Ansicht

```
<ul tal:define="results python: here.portal_catalog.searchResults(
    portal_type='News Item',
    sort_on='Date',
    sort_order='reverse',
    review_state='published');">

   <li tal:repeat="nachricht results">

      <!-- Link auf die Nachricht -->
      <a href="nachrichtenadresse" title="nachrichtentitel"
         tal:attributes="title nachricht/Title;
                         href nachricht/getPath;"
         tal:content="nachricht/Title">
            Titel der Nachricht
      </a>

      <!-- Datum der Nachricht -->
      <span tal:replace="python:here.toPortalTime(nachricht.Date)" />

      <!-- Beschreibung der Nachricht -->
      <metal:beschreibung tal:condition="nachricht/Description">
      <br /><span tal:replace="nachricht/Description" />
      </metal:beschreibung>

   </li>
</ul>
```