

# Table des matières

|           |  |           |
|-----------|--|-----------|
| <b>I</b>  | <b>Introduction</b>  | <b>4</b>  |
| <b>1</b>  | <b>Description.</b>  | <b>4</b>  |
| 1.1       | Inspirations. . . . .  | 4         |
| <b>2</b>  | <b>License.</b>  | <b>4</b>  |
| <b>3</b>  | <b>Convention.</b>   | <b>4</b>  |
| <b>4</b>  | <b>Présentation.</b>   | <b>5</b>  |
| 4.1       | Zope possède déjà le Document Template Markup Language, pourquoi avez vous besoin d'un autre langage à Template? . . . . . | 5         |
| 4.2       | Le langage DTML peut faire des choses que les ZPT ne peuvent pas, comme envoyer des emails dynamiquement? . . . . .        | 5         |
| <b>5</b>  | <b>Comment les ZPT fonctionnent-elles ?</b>  | <b>6</b>  |
| <b>II</b> | <b>TAL : Template Attributes Language</b>  | <b>8</b>  |
| <b>6</b>  | <b>tal :content et tal :replace</b>  | <b>8</b>  |
| 6.1       | Exemple de tal :content . . . . .  | 8         |
| 6.2       | Exemple de tal :replace . . . . .  | 8         |
| 6.3       | Insérer du code html ou xml. . . . .   | 8         |
| <b>7</b>  | <b>tal :repeat ou comment itérer sur une séquence.</b>   | <b>9</b>  |
| 7.1       | Les compteurs d'itération. . . . .   | 10        |
| <b>8</b>  | <b>tal :attributes ou comment remplacer les attributs d'une balise.</b>  | <b>10</b> |
| <b>9</b>  | <b>tal :condition ou comment insérer du contenu conditionnellement.</b>  | <b>11</b> |
| <b>10</b> | <b>tal :define ou comment définir des variables.</b>   | <b>12</b> |
| 10.1      | Ordre des opérations. . . . .  | 13        |
| <b>11</b> | <b>tal :omit-tag ou comment supprimer des balises.</b>   | <b>14</b> |

|   |           |
|---|-----------|
| <b>12 tal :on-error ou comment gérer les erreurs.</b>                                   | <b>15</b> |
| <b>13 Trucs et astuces.</b>   | <b>16</b> |
| 13.1 Les commentaires. . . . .  | 16        |
| 13.2 Variables globales. . . . .  | 16        |
| <b>III TALES : TAL Expression Syntax</b>  | <b>17</b> |
| <b>14 Les expressions chimiques.</b>  | <b>17</b> |
| 14.1 Chemins alternatifs. . . . .   | 18        |
| 14.1.1 default . . . . .  | 18        |
| 14.1.2 nothing . . . . .  | 18        |
| <b>15 Les expressions Python.</b>   | <b>19</b> |
| <b>16 Les expressions de formatage de chaînes de caractères.</b>                        | <b>20</b> |
| <b>17 exists : ou comment tester l'existence d'un chemin.</b>                           | <b>20</b> |
| <b>18 not : ou comment nier (inverser) une valeur.</b>                                  | <b>20</b> |
| <b>19 nocall : ou comment .</b>   | <b>21</b> |
| <b>20 Trucs et astuces.</b>   | <b>21</b> |
| 20.1 Inclure un point-virgule dans une expression . . . . .                             | 21        |
| 20.2 Inclure le signe dollar dans une expression de formatage de chaîne de caractères.  | 22        |
| 20.3 Comment manipuler un objet du genre <i>picture.jpg</i> dans une expression Python. | 22        |
| <b>21 Variables ZPT</b>   | <b>22</b> |
| 21.1 nothing . . . . .  | 22        |
| 21.2 default . . . . .  | 22        |
| 21.3 options . . . . .  | 23        |
| 21.4 attrs . . . . .  | 23        |
| 21.5 root . . . . .   | 23        |
| 21.6 here . . . . .   | 23        |
| 21.7 container . . . . .  | 23        |

|  |           |
|--|-----------|
| 21.8 modules . . . . .                                       | 23        |
| 21.9 request . . . . .                                       | 23        |
| 21.10user . . . . .  | 23        |
| 21.11template . . . . .                                      | 23        |
| 21.12repeat . . . . .  | 24        |
| 21.13CONTEXTS . . . . .                                      | 24        |
| <br>   |           |
| <b>IV Macro Expansion Template Attribute Language</b>        | <b>25</b> |
| <br>   |           |
| <b>22 Définir une macro.</b>                                 | <b>25</b> |
| <br>   |           |
| <b>23 Trucs et astuces.</b>                                  | <b>27</b> |
| 23.1 Ma technique pour l'utilisation des ZPT. . . . .        | 27        |
| <br>   |           |
| <b>V Liste d'exemples</b>                                    | <b>29</b> |
| <br>   |           |
| <b>24 Définir des pages HTML correctes.</b>                  | <b>29</b> |
| <br>   |           |
| <b>25 Une macro <i>breadcrumbs</i>.</b>                      | <b>30</b> |
| <br>   |           |
| <b>26 Utiliser le produit Localizer.</b>                     | <b>31</b> |
| 26.1 Définir une chaîne à localiser. . . . .                 | 31        |
| 26.2 Définir un formulaire de changement de langues. . . . . | 31        |
| <br>   |           |
| <b>27 Utiliser le produit Formulator.</b>                    | <b>32</b> |
| <br>   |           |
| <b>VI Appendice</b>  | <b>34</b> |
| <br>   |           |
| <b>28 A faire</b>  | <b>34</b> |
| <br>   |           |
| <b>29 Références</b>   | <b>34</b> |
| <br>   |           |
| <b>Index</b>   | <b>35</b> |

# Première partie

## Introduction

### 1 Description.

Ce didacticiel a pour but de vous plonger de manière rapide dans l'utilisation des Pages Templates.

#### 1.1 Inspirations.

Plusieurs articles m'ont inspirés :

- chapitres 5 et 9 et de l'appendice C du Livre Zope d'Amos Latteier et Michel Pelletier
- page "DTML to ZPT" : <http://www.zope.org/Members/peterbe/DTML2ZPT/>
- site <http://www.zopelabs.com>

Voir la section 29 (page 34).

### 2 License.

Ce matériel ne peut être distribué que conformément aux termes et conditions présentés dans la Licence de Publication Ouverte (Open Publication License), Draft v.1.0 ou suivantes. (La version la plus récente est actuellement disponible à <http://www.opencontent.org/openpub/>).

### 3 Convention.

Les termes *Page Templates*, *Zope Page Templates* et *ZPT* seront indifféremment utilisés dans ce didacticiel. Une traduction possible pourrait être **patron** ou **modèle de page**.

Le terme **graphiste** correspond à la traduction du terme anglais *designer* et peut être utilisé pour désigner un **infographiste**.

Le terme anglais *statement* sera souvent traduit par **instruction**, **déclaration** ou **commande**.

Le signe `_` sera utilisé pour visualiser plus facilement les espaces dans les chaînes de caractères du code de programmation. Comme ça n'est pas très clair, voici un exemple :

```
<span tal:replace="structure_request"></span>
```

Il y a donc un espace entre **structure** et **request**.

## 4 Présentation.

Les Page Templates sont un outil de génération de page Web. Elles aident les programmeurs et les graphistes à collaborer en produisant des pages Web dynamique pour le serveur d'application Zope <sup>1</sup>. Les graphistes peuvent les utiliser pour maintenir leurs pages sans devoir abandonner leurs outils, tout en préservant le travail requis pour incorporer ces pages dans une application.

Le but des Page Templates est de permettre aux programmeurs et aux graphistes de travailler ensemble facilement. Le graphiste peut utiliser un éditeur HTML WYSIWYG pour créer un patron, le programmeur peut alors l'éditer pour incorporer ce patron dans une application. Si cela est requis, le graphiste peut retravailler le patron dans son éditeur pour effectuer des changements de structure et d'apparence. En prenant certaines précautions pour préserver les changement fait par le programmeur, le graphiste ne perturbera pas l'application. Les ZPT atteignent ce but en adoptant trois principes :

- l'intégration harmonieuse avec les outils d'édition.
- ce que vous voyez est très similaire à ce que vous obtenez.
- garder le code hors des Templates, excepté pour la logique structurelle.

Une ZPT est comme un modèle des pages que vous générer. En particulier, c'est une page HTML valide.

### 4.1 Zope possède déjà le Document Template Markup Language, pourquoi avez vous besoin d'un autre langage à Template ?

Premièrement, le DTML n'est pas destiné aux graphistes. Une fois qu'une page a été convertie en Template, c'est du HTML non valide, la rendant difficile à travailler en dehors de l'application.

Deuxièmement, le DTML n'est pas la panacée en ce qui concerne la séparation entre présentation, logique et contenu.

### 4.2 Le langage DTML peut faire des choses que les ZPT ne peuvent pas, comme envoyer des emails dynamiquement ?

Le DTML n'est pas destiné à être abandonné complètement. Mais les ZPT sont destinés à être utilisé pour tout ce qui est présentation HTML/XML.

---

<sup>1</sup>Les Page Templates ne sont pas propre à Zope. Il est aussi tout à fait possible d'utiliser les Pages Templates en dehors de Zope.

## 5 Comment les ZPT fonctionnent-elles ?

Les ZPT utilisent le TAL ou Template Attribute Language. TAL consiste en une série d'attributs de balises.

*Qu'est-ce qu'une balise et qu'est-ce qu'un attribut ?*

```
<p class="article" align="justify">Mon paragraphe.</p>
```

Dans cet exemple, **<p>** est une balise. **<p>** est la balise d'ouverture délimitant les paragraphes. **</p>** est la balise de fermeture.

**class** et **align** sont des attributs de la balise **<p>**.

**"article"** et **"justify"** sont des expressions.

Par exemple, un titre de page dynamique pourrait ressembler à ceci :

```
<title tal:content="here/title">Titre de la page</title>
```

**tal:content** est une déclaration TAL. Etant donné qu'elle possède un espace de nom XML, la plupart des outils d'édition ne se plaindront pas qu'il ne la comprenne pas et, ainsi, ne l'enlèveront pas. De plus, il ne changera pas l'apparence du document lorsqu'il sera chargé dans un éditeur WYSIWYG ou un navigateur Web. Le nom **'content'** indique que le contenu de la balise **<p>** sera remplacé par la valeur de l'expression (ici *here/title*).

Lorsque le navigateur affichera la page, en supposant que le titre de la page soit 'Mon titre', cette section se présentera comme suit :

```
<title>Mon titre</title>
```

La valeur 'Titre de la page' est un commentaire uniquement destiné aux éditeurs WYSIWYG (dans ce cas-ci).

Toutes les déclarations TAL consistent en un nom commençant par **tal :** et toutes les déclarations ont une valeur associée. Cette valeur peut, éventuellement être nulle :

```
<p tal:omit-tag="">Cette balise n'apparaîtra pas.</p>
```

Cette valeur est toujours entre guillemets. Ce qui est, en fait, une spécification du XML.

Pour le graphiste utilisant un outil WYSIWYG, le titre dynamique est un parfait exemple d'HTML valide et est affiché dans son éditeur comme un titre doit être affiché. En d'autres mots, les ZPT s'intègrent harmonieusement avec les outils d'édition.

Cet exemple démontre également le principe que "Ce que vous voyez est très similaire à ce que vous obtenez". Lorsque vous visionnez le patron dans votre éditeur, le texte du titre (ici : "Titre de la page") agit comme conteneur pour le titre dynamique.

Il existe des déclaration TAL pour remplacer entièrement une balise, uniquement son contenu, ou ses attributs. On peut effectuer des boucles, omettre une balise. Ou joindre des parties de Templates ensemble et spécifier des traitements d'erreurs sommaires. Mais on ne peut créer de sous-routines, des contrôles de flux complexes ou exprimer des algorithmes complexes. Pour ça, les Scripts Python, les Zclasses ou les produits existent.

Maintenant que vous en savez un peu plus sur les ZPT, voyons comment les utiliser.

## Deuxième partie

# TAL : Template Attributes Language

## 6 tal :content et tal :replace

Ce sont les commandes de base des ZPT. **tal :content** ajoute le contenu entre les balises ouvrantes et fermantes <sup>2</sup>, **tal :replace** remplace la balise par le contenu.

### 6.1 Exemple de tal :content

```
<p>  
  <b tal:content="here/title_or_id">texte en gras</b>  
</p>
```

sera rendu comme suit :

```
<p>  
  <b>Mon titre</b>  
</p>
```

### 6.2 Exemple de tal :replace

```
<p>  
  <b tal:replace="here/title_or_id">texte en gras</b>  
</p>
```

sera rendu comme suit :

```
<p>Mon titre</p>
```

### 6.3 Insérer du code html ou xml.

Les balises html et xml sont échappées lorsqu'on les insèrent. Par exemple, le caractère > sera inséré come &gt;

Si vous voulez insérer du contenu formaté en html ou en xml, il faut utiliser la directive **structure**.

Exemple :

```
<span tal:replace="structure_request">La REQUEST vient ici</span>
```

Equivalent à

```
<dtml-var REQUEST>
```

---

<sup>2</sup>Si c'est une balise autofermante, le contenu est placé après la balise

## 7 tal :repeat ou comment itérer sur une séquence.

La commande tal :repeat permet d'itérer sur une liste. Par exemple, pour lister les objets du Folder courant :

```
<table border="1" width="100%">
  <tr>
    <th>Number</th>
    <th>Id</th>
    <th>Meta-Type</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item_container/objectValues">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Meta-Type</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

Explication :

l'expression "**container/objectValues**" renvoie la liste des objets contenu dans le Folder. Pour chaque **item** de la liste, on affiche son id (**item/getId**), son meta-type (**item/meta\_type**) et son titre (**item/title**)

C'est un peu l'équivalent de :

```
for item in container.objectValues():
  print (item.index)+1 # car commence à zero
  print item.getId()
  print item.meta_type
  print item.title
```

## 7.1 Les compteurs d'itération.

Notez que l'on affiche également le compteur d'itération **repeat/item/number**, accessible via le chemin **repeat/<nom\_de\_la\_variable>/<type\_de\_compteur\_d\_iteration>**

Différents compteurs sont disponibles :

- index : nombre, commence à 0
- number : nombre, commence à 1
- even : VRAI si nombre pair
- odd : VRAI si nombre impair
- start : VRAI si la séquence commence (index 0)
- end : VRAI si la séquence se termine
- first : [TO BE DONE]
- last : [TO BE DONE]
- length : longueur de la séquence
- letter : 'nombre', lettre minuscule ('a', 'z', 'aa', 'az', 'ba', 'bz', 'za', 'zz', 'aaa', ...)
- Letter : 'nombre', idem mais en majuscule
- roman : 'nombre', lettre romaine minuscule (i, ii, iii, iv, ...)
- Roman : 'nombre', lettre romaine majuscule (I, II, III, IV, ...)

On peut utiliser n'importe quel nom pour la variable de répétition. **item** n'est qu'un exemple. Du moment que qu'il commence par une lettre et qu'il ne contient que des lettres, des chiffres et des *underscores*.

## 8 tal :attributes ou comment remplacer les attributs d'une balise.

L'instruction tal :attributes permet de remplacer dynamiquement la valeur d'un attribut.

Exemple :

```
<p>  
Cette page a pour adresse <a tal:attributes="href_request/URL" tal:content="request/URL"></a>  
</p>
```

Note : un attribut dynamique écrase un attribut statique. exemple :

```
<a href="http://www.toto.be"  
  tal:attributes="href_request/URL"  
  tal:content="request/URL"></a>
```

produira ce code :

```
<a href="http://127.0.0.1:9080/ma_page">http://127.0.0.1:9080/ma_page</a>
```

Dans le cas, bien entendu ou votre page se trouve à `http://127.0.0.1:9080/ma_page`

## 9 tal :condition ou comment insérer du contenu conditionnellement.

L'instruction tal :condition teste la valeur de l'expression et insère la balise si cette expression est évaluée à VRAI. Si elle est évaluée à FAUX, la balise n'est pas insérée.

Les valeurs considérées comme fausses sont :

- string :"
- python :None
- nothing
- python :0
- not :<valeur\_vraie>
- Les séquences vides comme **python :[]**

Exemple : afficher une liste de mot séparés par des virgules mais à l'exception du dernier mot.

```
<div tal:define="liste_python:['alpha ',' beta ',' gamma']"
  tal:repeat="item_liste"
  tal:omit-tag="">
  <span tal:replace="item">Le mot</span>
  <span tal:condition="not:repeat/item/end"
    tal:omit-tag="">,</span>
</div>
```

Explication :

- On définit une liste de mot.
- On itère sur chaque élément de cette liste.
- On affiche chaque mot.
- On teste si nous ne sommes pas encore à la dernière itération : **not :repeat/item/end**  
Si c'est le cas, on affiche la virgule.

## 10 tal :define ou comment définir des variables.

L'instruction **tal :define** permet de définir de nouvelles variables. Par défaut, ces variables n'existent qu'à l'intérieur de la balise qui les définit mais il est possible de définir des variables globales également.

Exemple : nous désirons lister les objets du Folder triés par titre et par date de modification :

```
<table tal:define="objects_here/objectValues;
                sort_on python:((' title ', ' nocase ', ' asc '),
                                ('bobobase_modification_time', ' cmp ', ' desc '));
                sorted_objects python:sequence.sort(objects, sort_on)">
  <tr tal:repeat="item_sorted_objects">
    <td tal:content="item/title">title</td>
    <td tal:content="item/bobobase_modification_time">modification date</td>
  </tr>
</table>
```

Nous définissons une variable '**objects**' qui contiendra la liste des objets du Folder dans lequel on se trouve.

Nous définissons également la variable '**sort\_on**' qui contiendra les arguments pour la fonction '**sort**'. Ceci par souci de clarté.

Enfin, nous définissons la variable '**sorted\_objects**' qui contiendra la liste des objets triés.

## 10.1 Ordre des opérations.

Remarquez que l'on peut inclure plusieurs instruction TAL dans une même balise. D'où la nécessité de connaître l'ordre des opérations qui est :

- define
- condition
- repeat
- content or replace
- attributes
- omit-tag

Note : L'instruction on-error n'étant utilisée que lorsqu'une erreur survient, elle n'apparaît pas dans la liste

Explication :

Il arrive souvent que l'on veuille définir des variables pour utiliser dans les autres instructions. C'est pourquoi l'instruction '**define**' arrive en premier.

Le raisonnement suivant est de se demander si cet élément sera inclu ou non. C'est donc l'instruction '**condition**' qui suit. Etant donné que cette condition dépend souvent de variables définies auparavant, elle se place après l'instruction '**define**'.

Il est très utile d'être capable de remplacer plusieurs parties d'un élément avec des valeurs différentes à chaque itération d'un *repeat*. Donc, le *repeat* vient après.

Vient ensuite le contenu de la balise en lui même, suivit de l'édition des attributes de la balise.

Pour **omit-tag**, voir la section suivante.

## 11 tal :omit-tag ou comment supprimer des balises.

Nous n'avons pas encore parlé de l'instruction **omit-tag**. Elle permet de supprimer la balise tout en gardant son contenu s'il existe.

Exemple :

```
<div tal:define="liste_python:['alpha ','beta ','gamma']"
  tal:repeat="item_liste"
  tal:omit-tag="" >
  <b tal:content="item">Le mot</b>
  <span tal:condition="not:repeat/item/end"
    tal:omit-tag="" >,</span>
</div>
```

Sera transformé en :

```
<b>alpha</b>,<b>beta</b>,<b>gamma</b>
```

Alors que :

```
<div tal:define="liste_python:['alpha ','beta ','gamma']"
  tal:repeat="item_liste" >
  <b tal:content="item">Le mot</b>
  <span tal:condition="not:repeat/item/end">,</span>
</div>
```

Aurait été transformé en :

```
<div><b>alpha</b><span>,</span></div>
<div><b>beta</b><span>,</span></div>
<div><b>gamma</b></div>
```

Ce qui n'est pas très joli et n'apporte rien. A part des retours à la ligne embêtants.

On n'utilise jamais **tal :omit-tag** avec les instructions **tal :content**, **tal :replace** ou **tal :attributes**. En effet :

- **tal :attributes** a besoin d'une balise pour être utile.
- **tal :content** + **tal :omit-tag** est équivalent au **tal :replace** seul
- **tal :replace** + **tal :omit-tag** ne produira aucun effet.

L'expression utilisée par **omit-tag** est souvent vide mais vous pouvez utiliser n'importe quelle expression TALEX valide comme expression. Si cet expression est évaluée à FAUX, la balise est enlevée. Sinon, elle reste en place. Pour rappel, une expression TALEX vide est évaluée à FAUX.

## 12 tal :on-error ou comment gérer les erreurs.

Lorsqu'une instruction TAL produit une erreur, l'interpréteur TAL recherche une déclaration `tal :on-error` dans cette balise, puis dans les balises incluses. La première déclaration découverte est invoquée et est traitée comme une instruction `tal :content`.

Exemple :

```
<span tal:on-error="string:Ce_nom_n'existe_pas."  
tal:content="here/My_name">Oli</span>
```

Pour enlever complètement une balise lors d'une erreur :

```
<span tal:on-error="nothing"  
tal:content="here/My_name">Oli</span>
```

Une variable locale `error` est créée. Mais elle n'est pas facilement accessible. L'idéal est donc de créer un Script Python qui gèrera de manière plus fine l'erreur.

Exemple :

```
<span tal:on-error="structure_here/errorScript"  
tal:content="here/my_name">Oli</span>
```

Le script :

```
## Script (Python) "errorScript"  
##bind namespace=NS  
##  
error = NS['error']  
  
if error.type==NameError:  
    return "<p>Ce_nom_n'existe_pas.</p>"  
else:  
    return """<p>An_error_ocurred.</p>  
        <p>Error type: %s</p>  
        <p>Error value: %s</p>""" %_(error.type,  
                                    error.value)
```

Note : La deuxième ligne du script permet de créer une variable `NS` (pour namespace) qui permettra d'aller rechercher la variable `error` créée par l'instruction `on-error`. Sinon, vous pouvez également modifier la variable namespace dans l'onglet `bindings` du Script Python.

## 13 Trucs et astuces.

### 13.1 Les commentaires.

En DTML, la balise **dtml-comment** permet de mettre une partie de code en commentaire.

Voici la façon de faire avec les ZPT :

```
<tal:comment condition="nothing">
    ***** Mes commentaires viennent ici *****
    <p tal:content="here/story">Partie de texte</p>
</tal:comment>
```

### 13.2 Variables globales.

Si vous incluez la directive **global** lors de la définition d'une variable. Celle-ci devient une variable globale et devient, ainsi, utilisable en dehors de la balise où elle est définie.

Exemple :

```
<span tal:define="global_company_name_string:Zope_Corp,_Inc."
    tal:omit-tag="">Copyright</span>
```

```
<p>Bobo</p>
```

```
<p>Baba</p>
```

```
<p>Bubu</p>
```

```
<p tal:content="company_name">Company Name</p>
```

## Troisième partie

# TALES : TAL Expression Syntax

On ne peut pas mettre n'importe quoi comme valeur aux déclarations. Les expressions permises sont définies par le Zope Page Template Expression Syntax ou TALES.

Les différents types d'expressions autorisées sont :

|                       |                                      |
|-----------------------|--------------------------------------|
| path : <i>ou rien</i> | localise les valeurs par leur chemin |
| python :              | execute une expression Python        |
| string :              | formate une chaîne de caractères     |
| exists :              | teste si un chemin existe            |
| nocall :              | empêche le rendu d'un objet          |
| not :                 | nie une expression                   |

## 14 Les expressions cheminiales.

L'expression **here/title\_or\_id** est appelée une expression cheminiale <sup>a</sup> Il existe d'autres types d'expression mais les expressions cheminiales sont les plus utilisées.

Celles-ci commencent avec le nom d'une variable. Si la variable contient la valeur que vous voulez, vous vous arrêtez là. Sinon, vous pouvez ajouter d'autres variables séparées des autres par des barres obliques.

Zope définit un certain nombre de variables qui vous sont probablement déjà connues :

- request
- user
- container
- here
- template
- ...

Par exemple, en DTML, si vous vouliez afficher l'url de la page courante, vous faisiez ceci :

L'url est `<dtml-var "REQUEST.URL">`

En ZPT, vous ferez plutôt ceci :

L'url est `<span tal:replace="request/URL">http://127.0.0.1:8080/my_page</span>`

La section 21 (page 22) décrit ces variables.

---

<sup>a</sup>Traduction libre de *path expression*.

## 14.1 Chemins alternatifs.

Il est possible de donner des chemins alternatifs à l'expression. Ainsi, si le premier chemin n'est pas trouvé, le second chemin sera utilisé.

Pour ajouter un chemin supplémentaire dans l'expression, il faut ajouter le caractère *"pipe"* qui est, en fait, le caractère |.

Exemple :

```
<form method="post" action="do_add">
  <input type="text" name="firstname"
    tal:attributes="value_request/form/firstname_|_here/firstname"><br>
  <input type="submit">
</form>
```

Evidemment, il faut que le second chemin existe sinon, vous aurez quand même une erreur.

Il est possible d'utiliser des expressions autres que chimiques. Exemple :

```
<input type="text" name="age"
  tal:attributes="value_request/form/age_|_python:18"><br>
<input type="submit">
```

ou

```
<input type="text" name="firstname"
  tal:attributes="value_request/form/firstname_|_string:Toto"><br>
<input type="submit">
```

Vous n'êtes pas limité à 1 chemin alternatif. Plusieurs sont possibles :

```
<input type="text" name="firstname"
  tal:attributes="value_request/form/firstname_|_here/firstname_|_string:Toto">
```

Deux expressions sont souvent utilisées comme chemin alternatif : **default** et **nothing**.

### 14.1.1 default

[TO BE DONE].

### 14.1.2 nothing

[TO BE DONE].

## 15 Les expressions Python.

Les expressions Python permettent d'accéder aux objets Zope de manière plus fine. Si l'on désire lister le contenu d'un Folder, en utilisant les expressions cheminiales, nous procéderions comme ceci :

```
<table border="1" width="100%">
  <tr>
    <th>Number</th>
    <th>Id</th>
    <th>Meta-Type</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item_container/objectValues">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Meta-Type</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

par contre, si nous voulons ne lister que les objets d'un certain type, nous sommes plus limité avec les expressions cheminiales. Les expressions Python nous permettent d'aller plus loin :

```
<table border="1" width="100%">
  <tr>
    <th>Number</th>
    <th>Id</th>
    <th>Meta-Type</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item_python:container.objectValues('ANews_Entry')">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Meta-Type</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

Il faut donc préfixer l'expression par **python :** et utiliser la notation pointée. Cfr la section 20.3 (page 22) pour une astuce concernant la notation pointée.

C'est un peu l'équivalent du **expr=""** en DTML.

## 16 Les expressions de formatage de chaînes de caractères.

Elles sont préfixées par **string** :

Les variables accessibles sont utilisables en les incluant dans des accolades et en préfixant le tout du signe dollar.

Exemple :

```
<p tal:define="page_url_python:here.absolute_url()"
  tal:content="string:le_titre_est_${here/title}_et_l'url_est_${page_url}"></p>
```

Note : on peut omettre les accolades lorsque la variable n'est pas composée de plusieurs parties.

## 17 exists : ou comment tester l'existence d'un chemin.

Cette expression teste l'existence d'un chemin. La valeur renvoyée est 1 ou 0, c'est-à-dire VRAI ou FAUX. Elle est donc plus souvent utilisée avec l'instruction tal:condition

Renvoie 0 si la valeur n'existe pas ou si la valeur renvoie une valeur FAUSSE. Et 1 dans le cas contraire.

Exemple :

```
<p tal:condition="exists:request/form/age"
  tal:content="string:Ton_âge_est_${request/form/age}"></p>
```

## 18 not : ou comment nier (inverser) une valeur.

Permet d'inverser (nier) une valeur.

Exemple :

```
<p tal:condition="not:exists:request/form/age">
  Vous n'avez pas introduit votre âge!
</p>
```

## 19 nocall : ou comment .

Permet d'éviter le rendu d'un objet. Par défaut, les expressions essaient de rendre les objets qu'elles appellent. Si cet objet est un Script, une fonction , une méthode ou un objet exécutable, alors, l'expression retourne la valeur de retour de la fonction.

L'instruction **nocall** permet d'empêcher cela. C'est l'équivalent de ne pas mettre les parenthèses après l'appel à une fonction en Python.

```
<div tal:define="puissance_nocall:modules/math/pow">
  <p tal:content="python:puissance(2,-8)"></p>
</div>
```

Cet exemple montre comment se servir d'un module. Ici, on définit **puissance** comme étant une référence à la fonction **pow** du module **math**.

## 20 Trucs et astuces.

### 20.1 Inclure un point-virgule dans une expression

Le point-virgule servant à délimiter les instructions dans une expression, on se pose la question : *"Comment alors afficher un point-virgule ?"*

Exemple :

L'attribut HTML 'style' permet de séparer les différents styles appliqués à la balise par des points-virgules. Si vous voulez composer dynamiquement l'attribut style d'une balise comportant plusieurs styles, vous devez doubler les points-virgules que vous voulez afficher.

```
<a tal:attributes="href string:mailto:${here/toto_email};
                 style string:color:${here/color_01};;
                 font-size:${here/font_01};;
                 font-family:${here/font_family}">Mon lien</a>
```

Ce qui donnera ceci :

```
<a href="mailto:toto@trucmuche.be
  style="color:#336699;
  font-size:10pt;
  font-family:arial, helvetica, sans-serif;">Mon lien</a>
```

## 20.2 Inclure le signe dollar dans une expression de formatage de chaîne de caractères.

Comme pour la remarque de la section 20.1 (page 21), il faut doubler le signe dollar.

Exemple :

```
<p tal:content="string:_prix_$$$cost" >
  prix: $42.00
</p>
```

## 20.3 Comment manipuler un objet du genre *picture.jpg* dans une expression Python.

Le problème de la notation pointée utilisée dans les expressions Python est que si l'objet possède un point dans son identifiant, il devient difficile de le manipuler.

Exemple :

```
<a tal:attributes="href_python:here.images.picture.jpg.absolute_url()" >l'URL de mon image.</a>
```

ne fonctionnera pas. Il voudra accéder à l'attributs **jpg** de l'objet **picture**. Or il n'existe pas d'objet **picture**. L'objet s'appelle **picture.jpg**.

Voici la solution :

```
<a tal:attributes="href_python:here.images['picture.jpg'].absolute_url()" >Mon image</a>
```

Si, exceptionnellement, l'objet ne supporte pas d'être accédé par clé, vous pouvez utiliser **getattr** :

```
<a tal:attributes="href_python:getattr(here.images,_'picture.jpg').absolute_url()" >Mon image</a>
```

## 21 Variables ZPT

Tout comme avec les DTML ou les Script Python, les ZPT définissent plusieurs variables.

Note : Le terme variable n'est pas le plus approprié. On devrait plutôt parler d'objet.

### 21.1 nothing

Une valeur FAUSSE, similaire à une chaîne de caractères vide.

On peut l'utiliser dans les instructions `tal:replace` ou `tal:content` pour enlever une balise ou son contenu. Utilisée dans un `tal:attributes`, l'attribut correspondant est enlevé de la balise (à la différence d'une chaîne de caractères vide).

### 21.2 default

Une valeur spéciale qui ne change rien dans les `tal:content`, `tal:replace` ou les `tal:attributes`.

## 21.3 options

Représente les arguments (s'ils existent) qui ont été passé au Template.

Note : Les options ne sont présentes que si le Template a été appelé depuis un script Python.

## 21.4 attrs

Un dictionnaire des attributs de la balise dans laquelle cette variable est utilisée. Rarement utilisé.

## 21.5 root

L'objet *Root* ou racine. utilisé pour accéder aux objets Zope à partir d'emplacement fixe, sans se soucier de l'endroit d'où est appelé le Template.

## 21.6 here

L'objet sur lequel le Template est appelé. Souvent le même que **container** mais parfois différent si on utilise l'aquisition.

Analogue à la variable **context** utilisée dans les Script Python.

## 21.7 container

L'objet conteneur (souvent un Folder) dans lequel le Template se trouve.

Note : les variables **here** et **container** sont souvent identiques.

## 21.8 modules

Contient les modules accessibles aux templates comme, par exemple, le module **math**, ou le module **random**.

## 21.9 request

Identique à la variable DTML du même nom.

## 21.10 user

Correspond à l'objet **AUTHENTICATED\_USER** du DTML.

Pour afficher votre nom :

```
<p tal:content="user"></p>
```

## 21.11 template

Le Template proprement dit. Peu utilisé.

## 21.12 repeat

La variable utilisée lors des répétition.

Exemples :

```
<table border="1" width="100%">
  <tr>
    <th>Number</th>
    <th>Id</th>
    <th>Meta-Type</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item_container/objectValues">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Meta-Type</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

## 21.13 CONTEXTS

Un dictionnaire contenant toutes les variables définies plus haut. Utile pour déboguer ou accéder à une variable masquée par l'une de vos définition, par exemple si vous définissez une variable appelée **user**.

Pour l'afficher :

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

## Quatrième partie

# Macro Expansion Template Attribute Language

Les macros permettent de définir des sections de page qui pourront être utilisées dans d'autres pages. Les macros peuvent être de courtes sections ou une page entière.

## 22 Définir une macro.

C'est l'instruction **metal :define-macro** qui permet de définir une macro.

```
<p metal:define-macro="email">  
  <a tal:attributes="href_here/mon_adresse_email"  
    tal:content="here/mon_nom">Adresse email</a>  
</p>
```

Lorsque vous aurez besoin de cette macro dans une page, vous écrirez :

```
<div metal:use-macro="container/template_master/macros/email">  
  Adresse email  
</div>
```

où **template\_master** est l'id de la Page Template contenant la macro.

Notez que la balise `<div>` disparaît.



## 23 Trucs et astuces.

### 23.1 Ma technique pour l'utilisation des ZPT.

L'idéal, lorsque l'on crée une application, est d'éviter les redondances de codes et d'informations. Lorsque l'on crée une nouvelle Page Template, il est nécessaire de lui définir quelques directives concernant les macros.

On se retrouve donc, pour chaque page que l'on veut créer avec une structure semblable à celle-ci :

```
<html metal:use-macro="container/master_zpt/macros/page">
  <span metal:fill-slot="body" valign="top">

    </span>
</html>
```

Le système que j'utilise, permet de créer de nouvelles pages facilement. Chaque nouvelle page contiendra uniquement le contenu de la page et aucune directive concernant les macros. Ces dernières étant, dans la majorité des cas, toujours les mêmes. Voici comment faire :

Je crée une Page Template '**index.html**' dans la racine de mon site. Elle ne contient qu'une ligne :

```
<div metal:use-macro="container/template_standard/macros/page">Ma Page</div>
```

Je crée également une Page template '**template\_standard**' qui contiendra mes macros :

```
<div metal:define-macro="page" .....>
<html>
<head>
<link ... lien vers une feuille de style>
</head>
<body>
< ... >
Entête de la page ici
< ... >
<div tal:replace="structure_python:here.get_content(here,_request,
options)">Contenu_de_la_page.</div>
</body>
</html>
</div>
```

Un script python, appelé **get\_content** permet de retrouver le contenu de la page. il attend 3 paramètres : (client, mapping, options)

```
##parameters=client, mapping, options
##

from Products.PythonScripts.standard import structured_text

content_html = apply(context.content_html, (client, mapping), kwargs)

if hasattr(context, 'stx'):
    if context.stx:
        tmp_content_html = apply(context.get_content_html, (client, mapping), kwargs)
        context_html = structured_text(tmp_content_html)
```

Avec cette structure, si je désire créer une nouvelle page, j'ajoute un Folder contenant un objet appelé '**content\_html**' à l'intérieur. Cet objet peut être une DTML Method ou une Page Template.

Si le Folder possède une propriété '**stx**', le contenu peut être du texte structuré.

En résumé, une page est juste un Folder avec un objet 'content\_html' à l'intérieur.

Si je veux une page imprimable, j'ajoute une Page Template '**print**' contenant une seule ligne :

```
<div tal:use-macro="container/template_print/macros/page">Page imprimable</div>
```

Je crée une Page Template '**template\_print**' contenant un header plus simpliste et un lien vers une feuille de style tout aussi simple.

il suffit d'ajouter **/print** à la fin d'une URL pour obtenir la version imprimable.

## Cinquième partie

# Liste d'exemples

## 24 Définir des pages HTML correctes.

Ce que je considère comme une page HTML correcte contient :

- une ligne spécifiant la version HTML utilisée, ainsi qu'un lien vers la DTD
- un header contenant :
  - un titre
  - des balises meta
  - un lien vers une feuille de style

Voici une macro définissant une page correcte dans son entièreté :

```
<div metal:define-macro="page"
  tal:omit-tag=""><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title tal:content="here/title_or_id">The Title</title>
<div tal:content="structure_here/meta"
  tal:omit-tag="">Meta tags</div>
<link tal:attributes="href_python:container.css_main.absolute_url()"
  type="text/css" rel="stylesheet" />
</head>
<body>

</body>
</html>
</div>
```

Note : **meta** est un objet contenant les balises meta.

## 25 Une macro *breadcrumbs*.

Les Breadcrumbs (littéralement 'morceau de pain') sont un moyen utilisé pour se ballader dans la hiérarchie des répertoires plus aisément. Le terme provient de la fable 'Le Petit Poucet', qui avait semer des cailloux <sup>a</sup> pour éviter de se perdre dans la forêt. L'exemple le plus connu pour son utilisation des breadcrumbs est le site <http://dir.yahoo.com>

Avant tout, définir un Script Python qui renverra une liste de tuples. Le premier élément du tuple sera le lien vers un document parent. Le second sera le titre d'un document parent.

```
folder_list = []

# Nombre de répertoires à ignorer (exemple: Root):
skip = 1

for folder in context.REQUEST.PARENTS:
    folder_list .insert (0, folder)

for folder in range(skip):
    folder_list .pop(0)

# on travaille avec une copie sinon, on modifie un objet
# dont on peut encore avoir besoin plus tard:
copy = folder_list [:]

items = []

for folder in copy:
    if hasattr(folder, 'no_bc'):
        ''' Si le folder possède une propriété no_bc: on ignore ce Folder .'''
        if folder.no_bc:
            continue
    items.append((folder.absolute_url (), folder . title_or_id ()))

return items
```

La macro :

```
<div metal:define-macro="breadcrumbs"
    tal:repeat="item_container/breadcrumbs"
    tal:omit-tag="">
  <a tal:attributes="href_python:item[0]"
    tal:content="python:item[1]" class="breadcrumbs">Breadcrumbs
  </a>
  <span tal:condition="not:repeat/item/end"
    class="breadcrumbs">&gt;&gt;
  </span>
</div>
```

Il ne vous reste plus qu'à appeler cette macro dans vos page :

```
<div metal:use-macro="container/template_standard/macros/breadcrumbs">
  Breadcrumbs
</div>
```

---

<sup>a</sup>mais, apparemment, des morceaux de pain dans certaines versions.

## 26 Utiliser le produit Localizer.

On suppose que vous avez un objet `MessageCatalog` appelé `gettext`.

### 26.1 Définir une chaîne à localiser.

Très simple :

```
<i tal:content="python:here.gettext('description')">Description</i>
```

Voici une alternative (à vous de choisir suivant vos besoins) :

```
<div tal:define="gettext_python:here.gettext">
```

```
<i tal:content="python:gettext('description')">Description</i>
```

```
</div>
```

Ici, on définit une référence à 'gettext' en début de document. On peut alors utiliser cette référence partout dans ce document.

### 26.2 Définir un formulaire de changement de langues.

```
<metal:define-macro="change_language_form"
  tal:omit-tag="">
<table>
  <tr>
    <td tal:repeat="lang_here/gettext/get_languages">
      <a tal:attributes="href
string:${here/gettext/absolute_url}/changeLanguage?lang=${lang}"
      tal:content="python:here.gettext(lang)">A lang</a></td>
    </tr>
  </table>
</div>
```

Observation :

- ce n'est pas, à proprement parler, un formulaire mais, le principe est là.
- c'est une macro.

## 27 Utiliser le produit Formulator.

A l'opposé des méthodes DTML, les ZPT ne savent pas gérer les erreurs et validations de Formulator directement. ce n'est pas leur vocation de toute façon. Il faut donc passer par des Script Python pour y arriver.

On suppose que vous avez un objet Formulator appelé 'article' dans un Folder nommé 'forms'.

On suppose que l'on veuille insérer un article dans une base de données relationnelle. La Méthode ZSQL est /admin\_methods/insert/article

Définissons le formulaire dans la page 'index.html' :

```
<small tal:condition="options/error_fields != nothing" >
  <font color="red" >Votre formulaire comporte des erreurs !</font>
</small>

<form method="POST"
  tal:attributes="action_container/article/do_validate/absolute_url" >
<table>
  <tr tal:repeat="item_here/forms/article/get_fields" >
    <td>
      <font color="red" >
        <span tal:define="error_python:options.get(' error_fields ',_{'})"
          tal:condition="python:error.has_key(item.id)" >!
        </span>
      </font>
    </td>
    <td>
      <strong tal:content="python:item.get_value('title ')" >Title</strong>
    </td>
    <td>
      <font color="red" ><span
tal:condition="python:item.is_required()" >*</span></font>
    </td>
    <td><input type="text"
      tal:define="field_value_python:request.get(item.id ,_)"
      tal:replace="structure_python:item.render(field_value)" />
      <br>
      <font color="red"
        tal:define="error_python:options.get(' error_fields ',_{'})"
        tal:condition="python:error.has_key(item.id)" >
        <small tal:content="python:error.get(item.id ,)" >error message</small>
      </font>
    </td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td><font color="red" >*</font></td>
    <td>
      <i><font size="1" color="#FF0000" tal:content="string:champ
requis" ></font></i>
    </td>
  </tr>
</table>
```

```

        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>
            <input type="submit" value="Ajouter" >
        </td>
    </tr>
</table>
</form>

```

Définissons maintenant le script Python appelé 'do\_validate'

```

from Products.Formulator.Errors import ValidationError, FormValidationError
request = container.REQUEST
form = context.forms.article
error_fields = {}

try:
    form.validate_all_to_request(request)
except FormValidationError, error_value:
    for i in error_value.errors:
        error_fields[i.field_id] = i.error_text
    return container.index_html(error_fields = error_fields)
else:
    container.admin_methods.insert.article(request)
    return container.une_page.index_html()

```

Si une erreur Formulator survient, on renvoie vers le même formulaire mais avec, comme paramètres, la liste d'erreurs.

Sinon, on appelle la Méthode ZSQL d'insertion de l'article. Puis on renvoie vers une page quelconque.

La structure hiérarchique des objets est donc la suivante :

```

/forms/article
/admin_methods/insert/article
/une_page/index_html
/index_html
/do_validate

```

## Sixième partie

# Appendice

## 28 A faire

- équivalent du dtml-tree

## 29 Références

- Le ZopeBook (en) : <http://http://www.zope.org/Documentation/ZopeBook/>
- Convertir les DTML en ZPT (en) : <http://www.zope.org/Members/peterbe/DTML2ZPT/>
- ZopeLabs (en) : <http://www.zopelabs.com>

# Index

- attrs (variable), 23
- Breadcrumbs, 30
- cheminales (expressions), 17
- commentaires, 16
- compteurs d'itération (les), 10
- container (variable), 23
- CONTEXTS (variable), 24
- default (variable), 22
- dollar (signe), 22
- exists :, 20
- Formulator, 32
- globales (variables), 16
- here (variable), 23
- insérer du code html ou xml, 8
- Localizer, 31
- modules (variable), 23
- nocall :, 21
- not :, 20
- notation pointée (problème de la), 22
- nothing (variable), 22
- options (variable), 23
- ordre des opérations, 13
- path : (expressions cheminales), 17
- point-virgule, 21
- Python (expressions), 19
- repeat (variable), 24
- request (variable), 23
- root (variable), 23
- string : (expressions de formatage de chaînes de caractères), 20
- structure, 8
- TAL, 8
- tal :attributes, 10
- tal :condition, 11
- tal :content, 8
- tal :define, 12
- tal :omit-tag, 14
- tal :on-error, 15
- tal :repeat, 9
- tal :replace, 8
- TALES, 17
- template (variable), 23
- user (variable), 23
- valeurs fausses, 11