4

Le langage DTML

Living ain't easy,
but it's much harder when it's slow.
— John Cale

Introduction au DTML

Le DTML (pour Document Template Markup Language) est un langage à balises pour modèle de document.

Le DTML est le langage de script utilisé par Zope dans les pages web pour rendre leur contenu dynamique. C'est donc l'aspect le plus « visible » de Zope par les programmeurs.

Avec le DTML, on peut :

- afficher le contenu d'objets Zope (`DTML Document`, `File`, `Folder`, …) ;
- exécuter des méthodes sur des objets Zope ;
- afficher des attributs d'objets (par exemple, le titre d'un `DTML Document`) ;
- effectuer des tests sur ces variables ;
- parcourir des boucles à partir de ces variables ;
- gérer les exceptions qui peuvent se produire dans du code Zope.

Les instructions DTML sont volontairement très rudimentaires : elles ne sont qu'une petite dizaine et ne couvrent pas tous les aspects de la programmation (il n'y a pas, par exemple, d'instruction permettant d'affecter une variable).

Ainsi, le DTML ne doit pas être vu comme un langage à part entière, mais comme un outil permettant d'intégrer Zope aux pages web : le DTML est un préprocesseur de texte, au même titre que le préprocesseur du C, par exemple.

Objets supportant le DTML

Sous Zope, le code DTML est interprété par les objets suivants :

- `DTML Document` ;
- `DTML Method`.

Les autres objets standard de Zope ne supportent pas le DTML ; en revanche, il est possible de construire de nouveaux objets supportant le DTML. De plus, le DTML est exécuté dans le corps d'un `DTML Document`, jamais dans ses propriétés. Si un `DTML Document` possède une propriété `title` contenant une balise DTML, la propriété ne sera jamais interprétée en tant que code DTML par Zope.

Document templates

En réalité, les `DTML Document` et `DTML Method` sont dérivés d'une classe abstraite, appelée sous Zope *Document Template* (modèle de document). Nous étudierons dans le chapitre 12 consacré aux Produits l'utilisation de ces modèles de document et verrons que leur emploi peut dépasser le cadre de Zope.

> **Remarque**
> *Il est possible de créer de nouveaux objets Zope capables d'interpréter et de rendre le DTML (voir chapitre 12).*

Écriture de code DTML

Comme nous l'avons vu, le code DTML est inséré dans les `DTML Document` et les `DTML Method`. Sa syntaxe est vérifiée par Zope lorsque l'on valide les changements apportés à l'objet : les erreurs de codage sont ainsi immédiatement détectées.

Si les erreurs sont vérifiées au moment de la validation, le code n'est réellement exécuté (ou, plus exactement, interprété) que lors du « rendu » du document (dans une page web ou au sein d'un autre document).

Bien entendu, tous les traitements DTML sont effectués sur le serveur.

Pour utiliser les exemples présentés dans ce chapitre, vous pouvez créer, dans la racine de Zope, un dossier appelé `magasin`. Placez-vous ensuite dans ce dossier et créez un `DTML_Document` appelé `index_html`, contenant le code par défaut proposé par Zope.

© Éditions Eyrolles, 2001

Syntaxe générale du DTML

Comme la plupart des langages de scripts de pages web, le DTML s'insère dans des pages HTML, sous une forme similaire au HTML (les instructions sont encadrées par < et >). Cependant, il est tout à fait possible d'utiliser du DTML dans un autre contexte : texte simple ou *structured text*. Par exemple : Zope n'interprète, dans un document DTML, que ce qui commence par `<dtml-` ou `</dtml-` et se termine par `>`.

Le DTML est sensible à la casse des caractères, mais n'est sensible ni à l'indentation ni aux sauts de ligne : il est tout à fait possible d'écrire du code DTML sur une seule ligne ou sur plusieurs. Bien que la prudence conseille de l'indenter et de ne taper qu'une instruction par ligne, les contraintes du format HTML rendent souvent cet exercice difficile (par exemple, les sauts de lignes dans les cellules d'un tableau HTML peuvent influer sur la manière dont le document est affiché).

Les trois formes du DTML

Il existe trois formes génériques pour les instructions DTML. Ainsi, pour l'instruction `var`, les trois formes suivantes sont valides :

- `%(var)x`
- `<!--#var-->`
- `<dtml-var var>`

La première forme n'est quasiment jamais utilisée. La deuxième forme est parfois utilisée, et ressemble aux instructions ASP de Microsoft. Enfin, la troisième forme est de loin la plus utilisée.

> **Remarque**
>
> *Lorsqu'un utilisateur crée un document DTML, Zope insère par défaut du code DTML affichant le nom du document dans une page web. Ce code se conformait à la deuxième forme syntaxique jusqu'à la version 2.2 de Zope. Dans les versions postérieures, il se conforme à la troisième forme, les deux autres tombant progressivement en désuétude.*

Nous ne ferons référence qu'à la troisième forme dans cet ouvrage.

Composition d'une instruction DTML

Toute instruction DTML commence par `<dtml-` (ou `</dtml`) et se termine par `>`. Réciproquement, tout texte, au sein d'un `DTML Document`, commençant par `<dtml-` et se terminant par `>` est interprété par Zope comme du DTML.

© Éditions Eyrolles, 2001

L'ensemble `<dtml-instruction ...>` est appelé une balise DTML, par analogie avec les balises HTML.

> **Remarque**
> *Les balises DTML se conforment à la syntaxe XML : elles sont donc sensibles à la casse et doivent être écrites en minuscules.*

Les balises DTML supportent des attributs. La syntaxe générale d'une balise est la suivante (les crochets indiquent des éléments optionnels) :

```
<dtml-instruction [valeur] [attribut1[=valeur1]] [attribut2[=valeur2]] ...>
```

Chaque instruction reconnaît un certain nombre d'attributs ; il n'est pas possible de spécifier des attributs non reconnus. Les valeurs doivent être spécifiées entre guillemets, comme en XML.

Certains attributs nécessitent la présence d'une valeur, d'autres pas.

Balises simples et doubles

Les balises peuvent fonctionner par singletons ou par couples. Les balises singletons sont de la forme `<dtml-instruction instruction>`. Par exemple :

```
<dtml-var id>
```

Les balises qui fonctionnent par couples se présentent sous la forme d'une paire de balises, la première de la forme `<dtml-instruction instruction>` et la deuxième généralement de la forme `</dtml-instruction>`. De telles balises peuvent contenir d'autres balises DTML imbriquées, qu'elles soient simples ou doubles, comme dans l'exemple suivant :

```
<dtml-if title>
    <dtml-var title>
</dtml-if>
```

Certaines balises doubles peuvent admettre des balises intermédiaires, comme la balise `<dtml-if>`, qui peut contenir la balise intermédiaire `<dtml-else>` :

```
<dtml-if title>
    <dtml-var title>
<dtml-else>
    Pas de titre !
</dtml-if>
```

© Éditions Eyrolles, 2001

Il est possible d'imbriquer des balises doubles, mais en respectant l'ordre d'imbrication suivant : la première balise d'ouverture doit correspondre à la dernière balise de fermeture. Ainsi l'exemple suivant est-il incorrect :

```
<dtml-if title>
    <dtml-var title>
<dtml-in PARENTS>
    <dtml-var id>
</dtml-if>
</dtml-in>
```

Les balises `<dtml-if>` et `<dtml-in>` ne sont pas fermées dans le bon ordre. Il faudrait écrire :

```
<dtml-in PARENTS>
    <dtml-var id>
    <dtml-if title>
         <dtml-var title>
    </dtml-if>
</dtml-in>
```

Généralement, un peu de bon sens et d'indentation suffisent à débusquer les problèmes.

Les attributs `name` *et* `expr`

La plupart des instructions DTML supportent un attribut particulier, `name`.

Par commodité d'écriture, il est possible de spécifier sa valeur sans le faire précéder de `name=` et sans le mettre entre guillemets. Les deux exemples suivants sont donc équivalents :

```
<dtml-var name="title">
```

et

```
<dtml-var title>
```

L'attribut `expr` est également très courant : il est souvent utilisé au lieu de `name` (on ne peut faire porter à la fois l'attribut `name` et l'attribut `expr` à une balise). Toujours pour simplifier l'écriture, il est possible de spécifier sa valeur sans la faire précéder de `expr=` mais en l'entourant de guillemets. Ainsi :

```
<dtml-var "title">
```

équivaut à :

```
<dtml-var expr="title">
```

mais est différent de :

```
<dtml-var title>
```

© Éditions Eyrolles, 2001

Cette confusion est souvent source d'erreur ; aussi est-il important de bien comprendre, chaque fois qu'une forme abrégée est utilisée, quel est l'attribut sous-jacent.

Nous verrons, lors de l'étude de la balise `<dtml-var>`, en quoi les attributs `name` et `expr` diffèrent.

Liste des balises du DTML

Zope est conçu pour pouvoir étendre la syntaxe du DTML : aussi de nouvelles balises peuvent-elles être écrites par les programmeurs (il s'agit d'une fonctionnalité non documentée).

Voici la liste des balises DTML par défaut de Zope :

Balise ouvrante	Balise fermante ou intermédiaire
`<dtml-var>`	
`<dtml-in>`	`</dtml-in>`
`<dtml-tree>`	`</dtml-tree>`

`<dtml-if>`	`<dtml-else>`
	`<dtml-elif>`
	`<dtml-endif>` ou `</dtml-if>`

`<dtml-unless>`	`</dtml-unless>`
`<dtml-with>`	`</dtml-with>`
`<dtml-let>`	`</dtml-let>`
`<dtml-call>`	
`<dtml-raise>`	`</dtml-raise>`

`<dtml-try>`	`<dtml-except>`
	`<dtml-else>`
	`<dtml-finally>`
	`</dtml-try>`

`<dtml-comment>`	`</dtml-comment>`
`<dtml-return>`	
`<dtml-sendmail>`	`</dtml-sendmail>`
`<dtml-mime>`	`</dtml-mime>`

Commentaires : `<dtml-comment>`

Comme dans tout langage qui se respecte, le DTML permet au programmeur de commenter une partie de code avec les balises `<dtml-comment>` et `</dtml-comment>`.

Ces balises sont en fait très peu utilisées pour deux raisons :

- Elles sont longues à écrire.

© Éditions Eyrolles, 2001

- Elles n'empêchent pas la vérification syntaxique du bloc qu'elles couvrent et peuvent provoquer des erreurs d'interprétation du DTML si elles sont mal imbriquées.

Voyez l'exemple suivant :

```
<dtml-comment>
<dtml-in>
</dtml-comment>
```

Ce code n'est pas accepté par Zope. En effet, il manque une balise fermante à `<dtml-in>`, bien qu'il soit encadré par `<dtml-comment>` et `</dtml-comment>`. Il n'est donc pas possible d'utiliser les commentaires pour faire la vérification syntaxique d'une partie du code.

D'habitude, il est plus efficace (et surtout plus rapide), lorsque l'on veut supprimer une partie de code DTML, de remplacer le `<dtml` du début par `<Xdtml`. En effet, en HTML, les balises non reconnues par le navigateur ne sont pas affichées sur la page.

Astuce

Pour mettre en commentaire une balise DTML, vous pouvez écrire `<Xdtml-`.

Insertion de variables : `<dtml-var>`

La balise `<dtml-var>` est de loin la plus connue et la plus utilisée : elle permet d'insérer le contenu d'une variable dans la page courante.

Par « variable », nous entendons :

- le contenu d'une variable présente dans l'environnement Zope (voir la variable `REQUEST` ci-après) ;
- le contenu d'un attribut de l'objet courant ;
- le résultat de l'appel d'une méthode sur l'objet courant ;
- le contenu d'un objet Zope (éventuellement acquis) ;
- le contenu de la propriété d'un objet Zope, ou l'appel d'une méthode d'un objet Zope ;
- une expression Python.

Créer un nouveau document DTML dans le dossier `magasin`, puis l'ouvrir par défaut. Zope y insère le code suivant, que nous allons décortiquer :

```
<dtml-var standard_html_header>
<h2><dtml-var title_or_id></h2>
<p>
This is the <dtml-var id> Document.
</p>
<dtml-var standard_html_footer>
```

© Éditions Eyrolles, 2001

La première ligne, `<dtml-var standard_html_header>`, commande l'insertion du document appelé `standard_html_header`. Ce document est créé lors de l'installation de Zope ; si tel n'était pas le cas (ou si l'on détruisait le document `standard_html_header` existant, ce qui, d'ailleurs, ne représente pas beaucoup d'intérêt), Zope indiquerait une erreur.

La deuxième ligne, `<dtml-var title_or_id>`, permet d'insérer le résultat de l'appel de la méthode `title_or_id ()` d'un `DTML Document`. Cette méthode renvoie la valeur de l'attribut `title` du document courant si elle existe, ou la valeur de l'attribut `id` si `title` n'existe pas.

La quatrième ligne, `<dtml-var id>`, insère le contenu de l'attribut `id` de l'objet courant, et la dernière ligne insère le contenu du document `standard_html_footer`.

En cliquant sur l'onglet View du document et en consultant le source HTML produit, on constate que les balises `<dtml-var>` ont chaque fois été remplacées par la valeur qu'elles représentent.

Nous verrons plus loin dans cette section comment Zope détermine s'il faut insérer un document, appeler une méthode ou afficher un attribut.

Spécification du contenu

La balise `<dtml-var>` est dotée de nombreux attributs qui permettent de spécifier la manière dont les valeurs sont affichées. Les deux attributs les plus importants sont `name` et `expr`, qui indiquent le contenu à insérer.

Attribut `name`

Comme nous l'avons vu lors de la présentation de la syntaxe du DTML, le code précédent est parfaitement équivalent à celui-ci :

```
<dtml-var name="standard_html_header">
<h2><dtml-var name="title_or_id"></h2>
<p>
This is the <dtml-var name="id"> Document.
</p>
<dtml-var name="standard_html_footer">
```

On ne peut spécifier, avec l'attribut `name`, *que* des noms de variables (objets, attributs, méthodes). Le comportement de l'attribut `name` est le suivant, suivant le type de variable passé en valeur :

Type de variable	Comportement	Exemple
Attribut d'un objet	*Afficher la valeur de cet attribut*	`<dtml-var id>`
Méthode d'un objet	*Afficher le résultat de l'appel de cette méthode sans paramètre*	`<dtml-var title_or_id>`
Objet Zope	*Afficher le rendu de l'objet*	`<dtml-var standard_html_header>`

© Éditions Eyrolles, 2001

Attribut expr

L'attribut expr permet d'insérer le contenu d'une *expression* Python plutôt qu'une variable, à la différence de l'attribut name.

Remarque

La syntaxe d'une expression python est relativement simple et intuitive. Au besoin, le lecteur pourra se reporter au chapitre 9, consacré à Python, pour une description détaillée de la syntaxe de ce langage.

Comme nous l'avons vu, l'attribut expr peut être omis si une valeur est entre guillemets dans la balise dtml-var.

Alors que, pour l'attribut name, le comportement dépend du type de variable, le comportement pour l'attribut expr est toujours le même : insérer le résultat de l'expression python correspondante. Le résultat est le même pour un attribut d'objet, mais diffère complètement pour une méthode ou un objet. Ainsi, le code <dtml-var "title_or_id"> sera remplacé par le texte <Python Method object at 183f3e0> ! En effet, title_or_id étant une méthode, l'expression title_or_id sans guillemets affiche l'objet méthode en Python.

Le résultat est encore plus déroutant pour l'appel d'un objet : le code <dtml-var "standard _html_header"> insère le contenu de l'objet standard_html_header, c'est-à-dire sa zone de saisie DTML, sans interpréter le code correspondant.

Attention

La confusion entre les attributs name *et* expr *est à l'origine de nombreuses erreurs, difficiles à retrouver. L'erreur est d'autant plus difficile à déceler que les attributs sont implicites et ne dépendent que de la présence de guillemets.*

Aussi est-il conseillé de n'utiliser implicitement dans les balises <dtml-var> *que l'attribut* name*, et de toujours préciser* expr= *lors de l'utilisation de l'attribut* expr*.*

L'attribut expr permet d'effectuer des calculs simples : l'expression <dtml-var expr="123 * 456"> est remplacée dans un DTML Document par son résultat. Il est également possible d'effectuer des calculs sur des attributs : si un document est doté d'une propriété nombre portant la valeur « 123 », l'expression DTML <dtml-var expr="nombre * 456"> insère la valeur 56088 sur le rendu du document DTML.

L'attribut expr est indispensable pour exécuter des méthodes ou récupérer certains attributs de l'espace de noms. L'environnement Zope présente ainsi une variable PARENTS. C'est un tableau contenant la liste des parents de l'objet courant. Pour accéder au parent immédiat de l'objet courant, il faut récupérer la variable PARENTS[0] : cela n'est possible qu'avec l'attribut expr. Par exemple, pour afficher le titre ou l'identifiant du parent :

```
<dtml-var expr="PARENTS[0].title_or_id ()">
```

© Éditions Eyrolles, 2001

L'environnement Zope définit aussi des méthodes pour interagir avec Zope, ou pour manipuler les chaînes. Ces méthodes ne peuvent être appelées que si l'on recourt à l'attribut `expr`.

Remarque

Les guillemets ayant une signification dans la syntaxe des balises DTML, il est interdit de les utiliser dans des expressions : il faut les remplacer par des guillemets anglais, simple (') ou triple (''').

Par exemple, pour afficher une chaîne en tant qu'expression, il ne faudrait pas écrire `<dtml-var ""chaîne"">` *mais* `<dtml-var "'chaîne'">`.

Voici un tableau qui permet, à titre de curiosité, d'effectuer une traduction entre attributs `name` et `expr` ; il n'est présenté ici qu'à titre purement didactique, mais peut servir à « retracer » certaines erreurs dans la pratique…

Type de variable	Attribut `name`	Attribut `expr`
Attribut d'un objet	`<dtml-var id>`	`<dtml-var expr="id">`
Méthode d'un objet	`<dtml-var title_or_id>`	`<dtml-var expr="title_or_id ()">`
Objet Zope	`<dtml-var standard_html_header>`	`<dtml-var expr="standard_html_header">` *ou* `<dtml-var "_.getitem ('standard_html_header', 1)">`

Zope propose, pour une utilisation au sein des balises du DTML, un ensemble de fonctions disponibles dans l'espace de noms (_). Le chapitre 3 nous a permis de voir quels objets étaient accessibles au travers de l'espace de noms (_).

Pour faciliter l'accès à certaines fonctions python bien pratiques, Zope autorise également l'accès, à travers la variable _, à quelques modules Python standard : `string`, `math`, `random` et `whrandom`. En voici une description exhaustive.

Remarque

Pour accéder aux modules, il est indispensable d'utiliser la notation `_.<module>.<méthode>`.

Les paragraphes suivants rappellent l'essentiel des méthodes et attributs définis par les modules exportés par Zope. Le lecteur pourra se référer à la documentation de Python pour de plus amples informations.

© Éditions Eyrolles, 2001

Le module `string`

Le module `string`

Attribut	Définition
`digits`	*La chaîne* `'0123456789'`
`hexdigits`	*La chaîne* `'0123456789abcdefABCDEF'`
`octdigits`	*La chaîne* `'01234567'`
`lowercase`	*La chaîne* `'abcdefghijklmnopqrstuvwxyz'`[a]
`uppercase`	*La chaîne* `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`
`letters`	*La concaténation de* `lowercase` *et* `uppercase`.
`whitespace`	*Une chaîne contenant tous les caractères considérés comme des espaces : saut de ligne, tabulation, caractère d'espacement, etc.*
`atof(chaine)`	*Convertit* `chaine` *en un flottant.*
`atoi(chaine, base)`	*Convertit* `chaine` *en un entier en notation en base* `base` *(par défaut,* `base = 10`*).*
`capitalize(chaine)`	*Renvoie la chaîne passée en paramètre, chaque mot comportant une majuscule initiale.*
`capwords(chaine)`	*Supprime les espaces doubles et transforme toutes les lettres en majuscules.*
`lower(chaine)`	*Convertit la chaîne en minuscules.*
`upper(chaine)`	*Transforme la chaîne en majuscules.*
`swapcase(chaine)`	*Transforme les minuscules en majuscules, et réciproquement.*
`find(chaine, sub, start=0)`	*Renvoie l'index de la première occurrence, en partant de la gauche, de la sous-chaîne* `sub` *dans la chaîne* `chaine`.
`rfind(chaine, sub, start=0)`	*Idem* `find`*, mais en partant de la droite.*
`index(chaine, sub, start=0)`	*Idem* `find`*, mais provoque une exception* `ValueError` *si la chaîne* `sub` *n'est pas trouvée.*
`rindex(chaine, sub, start=0)`	*Idem* `find`*, mais provoque une exception* `ValueError` *si la chaîne* `sub` *n'est pas trouvée.*
`count(chaine, sub, start=0)`	*Renvoie le nombre d'occurrences de la chaîne* `sub` *dans la chaîne* `chaine`*, en commençant par la position* `start`.
`maketrans(source, cible)`	*Renvoie une table de transformation associant à chaque caractère de* `source` *le caractère de* `cible` *à la même position. N'a de sens qu'avec la fonction* `translate`.
`translate(chaine, table, suppr)`	*Retire de* `chaine` *tous les caractères de* `suppr`*, puis applique la transformation de* `table`. `table` *est le résultat de l'appel d'une fonction* `maketrans`.
`join(liste, separateur)`	*Concatène les chaînes contenues dans* `liste` *en les séparant par la chaîne* `separateur`.

© Éditions Eyrolles, 2001

Le module `string` *(suite)*

Attribut	Définition
`split(chaine, separateur, max=-1)`	*L'inverse de* `join` *: transforme la chaîne en une liste, en créant un nouvel élément à chaque* `separateur`*. Renvoie au maximum* `max` *fois.*
`lstrip(chaine)`	*Ôte les espaces au début de* `chaine`*.*
`rstrip(chaine)`	*Ôte les espaces à la fin de* `chaine`*.*
`strip(chaine)`	*Ôte les espaces au début et à la fin de* `chaine`*.*
`ljust(chaine, largeur)`	*Ajoute autant d'espaces que nécessaire au milieu et à la fin d'une chaîne de façon à renvoyer une chaîne qui soit justifiée à gauche lorsqu'elle est affichée avec une largeur de* `largeur` *caractères.*
`rjust(chaine, largeur)`	*Idem mais justifié à droite*
`center`	*Idem, mais centré.*
`zfill(chaine, largeur)`	*Idem mais aligne avec des zéros à gauche (utilisé lorsque* `chaine` *représente un nombre).*

a. *Cette chaîne,* `uppercase`, `letters` *et* `whitespaces`, *peut en réalité varier suivant la plate-forme : consulter la documentation de Python pour plus d'informations.*

Le module `math`

Le module `math` définit des fonctions qui sont souvent peu utiles dans des sites web… Sauf mention contraire, les arcs sont exprimés et retournés en radians.

Le module `math`

Attribut	Définition
`acos(x)`	*Renvoie l'arc cosinus de* x.
`asin(x)`	*Renvoie l'arc sinus de* x.
`atan(x)`	*Renvoie l'arc tangent de* x.
`atan2(x, y)`	*Renvoie l'arc tangent du quotient de* x *et* y.
`ceil(x)`	*Renvoie le plus petit entier supérieur à* x.
`cos(x)`	*Renvoie le cosinus de* x.
`cosh(x)`	*Renvoie le cosinus hyperbolique de* x.
`e`	*La valeur de* e
`exp(x)`	*Renvoie l'exponentielle de* x.
`fabs(x)`	*Renvoie la valeur absolue de* x.
`floor(x)`	*Renvoie le plus grand entier inférieur à* x.
`fmod(x, y)`	*Renvoie le modulo de* x *par* y.
`frexp(x)`	*Renvoie la mantisse et l'exposant en base 2 de* x *de sorte que la valeur absolue de la mantisse soit nulle ou comprise entre* `0,5` *et* `1,0`.

© Éditions Eyrolles, 2001

Le module `math` *(suite)*

Attribut	Définition
`hypot(x, y)`	*Renvoie la longueur de l'hypoténuse pour les longueurs des côtés du triangle passés en paramètres.*
`ldexp(x,y)`	*Renvoie x fois 2 puissance* y *(*$x \times 2^y$*).*
`log(x)`	*Renvoie le logarithme base* e *de* x*.*
`modf(x)`	*Renvoie un tuple composé de la valeur entière de* x *et de sa valeur fractionnelle.*
`pi`	*La constante* `pi`*.*
`pow(x, y)`	*Renvoie* x *puissance* y*.*
`sin(x)`	*Renvoie le sinus de* x*.*
`sinh(x)`	*Renvoie le sinus hyperbolique de* x*.*
`sqrt(x)`	*Renvoie la racine carrée de* x*.*
`tan(x)`	*Renvoie la tangente de* x*.*
`tanh(x)`	*Renvoie la tangente hyperbolique de* x*.*

Le module `random`

Le module `random`

Attribut	Définition
`betavariate(a, b)`	*Renvoie une distribution bêta.*
`choice(sequence)`	*Renvoie un élément de* `sequence` *au hasard.*
`cunifvariante(moyenne, arc)`	*Renvoie une distribution circulaire uniforme.*
`expovariate(l)`	*Renvoie une distribution exponentielle.*
`gamma(alpha, beta)`	*Renvoie une distribution gamma.*
`gauss(mu, sigma)`	*Renvoie une distribution de Gauss.*
`lognormvariate(mu, sigma)`	*Renvoie une distribution normale.*
`normalvariate(mu, sigma)`	*Renvoie une distribution normale.*
`paretovariate(alpha)`	*Renvoie une distribution de Pareto.*
`randint(a, b)`	*Renvoie un entier au hasard, compris entre* a *et* b *inclus.*
`random()`	*Renvoie un flottant entre 0 et 1 inclus.*
`uniform(a, b)`	*Renvoie un flottant entre a et b inclus.*
`vonmisesvariate(mu, kappa)`	*Distribution de Von Mises.*
`weilbullvariate(alpha, beta)`	*Renvoie une distribution de Weilbull.*

© Éditions Eyrolles, 2001

Le module whrandom

Le module vhrandom

Attribut	Définition
choice(sequence)	*Renvoie un élément de* sequence *au hasard.*
randint(a, b)	*Renvoie un entier x tel que* a<=x<=b.
random()	*Renvoie un flottant* x *tel que* 0<=x<=1.
seed(X, Y, Z)	*Initialise le générateur de nombre aléatoire avec les entiers* X, Y *et* Z.
uniform(a, b)	*Renvoie n réel* x *tel que* a<=x<=b.

Les autres variables de l'espace de noms

L'espace de noms définit d'autres variables pour simplifier la vie de l'utilisateur. Ces variables ne sont accessibles qu'à travers l'écriture _.attribut, c'est-à-dire en désignant explicitement l'attribut depuis l'espace de noms.

Attribut	Définition
abs(x)	*Renvoie la valeur absolue de* x.
chr(x)	*Renvoie le caractère correspondant au code ASCII de* x. *Le code ASCII qu'il faut absolument connaître est celui des guillemets :* ". *Il s'agit du code 34. C'est le seul moyen, nous le verrons par la suite, d'insérer en DTML des chaînes contenant des guillemets.*
DateTime(…)	*Crée un objet* DateTime *en fonction des arguments.*
divmod(a, b)	*Renvoie un tuple contenant la valeur entière et le reste de la division de* a *et* b.
float(x)	*Convertit* x *en un flottant.*
getattr(O, attr)	*Renvoie l'attribut* attr *de l'objet* O (*il est utilisé lorsque* attr *n'est pas un symbole python valide).*
hasattr(O, attr)	*Renvoie vrai si l'attribut attr est accessible pour l'objet O. Attention, l'attribut peut ne pas appartenir à O, mais doit être acquis à travers l'espace de noms.*
getitem(nom, execute)	*Renvoie l'objet* nom *s'il existe dans l'espace de noms. Si* execute *est vrai, renvoie la valeur correspondant à l'appel de l'objet, sinon renvoie la valeur de l'objet lui-même ; voir chapitre 8 pour plus d'informations.*
hash(O)	*Renvoie une valeur entière identifiant l'objet.*
hex(x)	*Convertit l'entier* x *en une chaîne représentant sa valeur hexadécimale.*
int(x)	*Convertit le nombre* x *en un entier.*
len(x)	*Renvoie la longueur de* x.
max(x)	*Renvoie le plus grand élément de la séquence* x.
min(x)	*Renvoie le plus petit élément de la séquence* x.
namespace(nom=valeur, nom=valeur, …)	*Fonction utilisée pour ajouter des variables « nom » à l'espace de noms courant. Cette forme est désuète : voir chapitre 8 pour plus d'informations.*

© Éditions Eyrolles, 2001

Attribut	Définition
None	*Équivalent au* None *de Python.*
oct(x)	*Renvoie une chaîne représentant la valeur octale de* x.
ord(x)	*Renvoie la valeur numérique du code ASCII de la lettre* x *: il s'agit de la fonction inverse à* chr().
pow(x, y)	*Renvoie* x *à la puissance* y.
round(x, n)	*Renvoie la valeur* x *arrondie à* n *décimales.*
str(x)	*Renvoie la valeur* x *convertie en chaîne.*

Dans ce tableau, certaines variables ou fonctions sont strictement équivalentes à leurs homonymes en Python : None, chr, str, int… En fait, la sécurité de Zope est telle qu'il n'est possible d'appeler dans du code Zope que des fonctions ou méthodes qui appartiennent à l'espace de noms. Ainsi, il est nécessaire d'y redéfinir les variables et méthodes vraiment indispensables.

Spécification de la mise en forme

La balise <dtml-var> possède de nombreux attributs afin de contrôler la façon dont les données sont insérées.

Format des données

L'attribut fmt permet de convertir les données de la valeur à insérer dans un format lisible.

L'attribut prend en paramètre (obligatoire) le nom d'une méthode qui convertit les données de la balise en une chaîne de caractères.

Cet attribut est souvent utilisé pour convertir des valeurs de dates. La méthode bobobase_modification_time renvoie la date de dernière modification d'un document ; voici quelques exemples de ce que peut insérer la balise dtml-var associée à l'attribut fmt :

Balise	Valeur insérée
<dtml-var bobobase_modification_time>	*2001/01/01 18:57:47.55 GMT+1*
<dtml-var bobobase_modification_time fmt="aCommon">	*Jan 1, 2001 7:05 pm*
<dtml-var bobobase_modification_time fmt="PreciseAMPM">	*07:06:57.080 pm*

D'autres méthodes sont disponibles, telles que, structured-text : cette dernière insère le contenu de la variable, convertit au format structured text (voir le chapitre 6 pour plus d'informations sur le structured text).

© Éditions Eyrolles, 2001

Créons par exemple, dans un dossier, un `DTML Document` appelé `stext_test`, et entrons-y le texte suivant :

```
Voici un exemple de document en **structured text**.


   Il est possible de traiter ces documents avec Zope de manière naturelle.

   Le structured text offre la possibilité d'écrire des documents
   relativement complexes,
   sans utiliser de HTML, mais contenant :

   * des titres et une structuration,

   * des gras, des soulignés, des italiques,

   * des liens symboliques

   * et même des tableaux !
```

Puis insérons, dans un `DTML Document`, la balise suivante :

```
<dtml-var stext_test fmt="structured-text">
```

Lors du rendu de ce dernier document, `stext_test` est transformé en HTML.

Voici une liste de méthodes disponibles pour la balise `fmt` au sein des `DTML Document` et des `DTML Method` :

Méthode	Description
`whole-dollars`	*Ajoute le signe* `$` *à une valeur numérique entière.*
`dollars-and-cents`	*Formate une valeur numérique avec deux décimales et le signe* `$`*.*
`collection-length`	*Insère la longueur d'une collection d'objets (tableau, tuple ou dictionnaire).*
`structured-text`	*Formate une chaîne en structured text.*
`aCommon`	*Insère la représentation courante (aux États-Unis) d'une date.*
`aCommonZ`	*Idem* `aCommon`*, avec spécification du fuseau horaire.*
`Day`	*Insère le nom du jour (en anglais).*
`day`	*Insère le numéro du jour dans le mois.*
`HTML4`	*Insère la date au format recommandé par le W3C.*
`rfc822`	*Insère la date au format RFC822 (e-mails).*

Hélas, la plupart des méthodes de formatage des dates ou des nombres supportées par Zope sont spécifiques à l'américain. Aussi, s'il est relativement courant d'utiliser `structured-text`, il est beaucoup plus rare d'employer `aCommon` dans un site qui se veut multilingue.

© Éditions Eyrolles, 2001

En fait, l'attribut `fmt` permet de spécifier n'importe quelle méthode pour prendre en charge le formatage de la donnée.

Enfin, l'attribut `fmt` supporte la même syntaxe de spécification de format que les chaînes Python : le signe `%` suivi d'un spécificateur de format. Par exemple, la balise `<dtml-var expr="123" fmt="%04d">` insère le texte `0123`.

Valeur par défaut

La balise `<dtml-var>` peut être appelée avec des variables qui représentent des valeurs incorrectes, impossibles à afficher en HTML ou n'ayant pas de sens (la valeur `None` de Python, par exemple). L'attribut `null` permet de spécifier une valeur par défaut pour ces valeurs :

```
<dtml-var "methode()" null="0">
```

Si l'appel à `methode()` renvoie `None`, la balise insère `0`.

En principe, lorsque la balise `<dtml-var>` n'arrive pas à résoudre la variable spécifiée par l'attribut `name` ou `expr`, il déclenche une exception.

L'attribut `missing` permet de ne pas déclencher d'exception, mais d'afficher la valeur spécifiée (ou une chaîne vide si aucune valeur n'est spécifiée). Par exemple :

```
<dtml-var xyz missing="Document introuvable">
```

Cette balise insère la chaîne `Document introuvable` si `xyz` n'est pas un attribut, une méthode ou un objet résoluble par Zope.

Troncature

L'attribut `size` prend en charge la troncature de texte : lorsque celui-ci dépasse la taille spécifiée, la fin de la valeur est tronquée et remplacée par la valeur de l'attribut `etc` (ou « … » par défaut). Ainsi :

```
<dtml-var "'Une longue chaîne'" size="10">
```

est remplacé par :

```
Une longue...
```

et

```
<dtml-var "'Une longue chaîne'" size="10" etc="XXX">
```

est remplacé par :

```
Une longueXXX
```

© Éditions Eyrolles, 2001

Changement de casse

Les attributs `lower` et `upper` permettent de changer la casse d'une valeur chaîne :

`<dtml-var title_or_id upper>` sur un document dont le nom est « `MonDocument` » insère « MONDOCUMENT ».

L'attribut `capitalize` permet de mettre en majuscule le caractère initial de chaque mot.

Formatages spécifiques au Web

Certains attributs de la balise `<dtml-var>` permettent de modifier automatiquement la valeur retournée par la balise `<dtml-var>` pour la rendre compatible avec le format du document (HTML, URL, etc.). Pour ce faire, les caractères qui ont une signification particulière dans un langage sont transformés de façon qu'ils perdent cette signification et soient traités comme des caractères normaux.

Pour afficher une variable susceptible de contenir du code entrant en conflit avec du HTML, on utilise l'attribut `html_quote`. Par exemple, en HTML, la chaîne `<BR>` est dotée d'une signification particulière. On se prémunit de ses effets dans du code HTML en utilisant l'instruction suivante :

```
<dtml-var expr="'<BR>'" html_quote>
```

Ce code insère la chaîne « `&lt;BR&gt;` ». Bien entendu, l'attribut `expr` sera remplacé par une variable ou une méthode, renvoyant une chaîne, comme par exemple `<dtml-var title html_quote>`.

L'attribut `newline_to_br` permet, dans une chaîne, de remplacer les sauts de ligne par des balises `<BR>`.

Lorsqu'on souhaite utiliser une chaîne dans une URL, il est indispensable d'éviter l'interprétation des caractères spéciaux (&, =, etc.) par le navigateur : l'attribut `url_quote` s'en charge. Ainsi :

```
<A HREF="<dtml-var propriete url_quote>">
```

Ce code permet d'utiliser la valeur de `propriete` comme URL, en l'épurant des caractères non reconnus dans les URL. De même, lorsqu'une URL comporte des informations de requête, l'attribut `url_quote_plus` remplit le même rôle, mais en remplaçant également les espaces par des signes + :

```
<A HREF="<dtml-var propriete url_quote>?PARAMETRE=<dtml-var
➥parametre url_quote_plus>">
```

L'attribut `sql_quote` permet de réaliser la même tâche pour des chaînes destinées à des moteurs SQL. Nous aurons l'occasion d'en reparler dans le chapitre 8 qui traite de l'interaction entre Zope et SQL.

© Éditions Eyrolles, 2001

Autres formatages

L'attribut `spacify` permet de remplacer, dans une chaîne insérée, les underscores (signe de soulignement) (_) par des espaces.

L'attribut `thousands_commas` formate un nombre au format américain (les milliers séparés par des virgules).

Résolution des noms

Nous avons vu que la balise `<dtml-var>` était capable, du moins avec l'attribut `name`, d'appeler correctement la variable invoquée, suivant qu'il s'agit d'un attribut, d'une méthode ou d'un objet. Nous avons également vu, au chapitre précédent, combien l'acquisition pouvait étendre la portée d'un attribut. Enfin, l'environnement Zope propose de nombreux attributs et méthodes. Il est donc important de connaître l'ordre dans lequel ces éléments sont recherchés lorsqu'une balise `dtml-var` y fait référence.

Voici, dans l'ordre décroissant de priorité, les éléments qui sont recherchés par Zope :

1. les variables passées explicitement en paramètre d'une `DTML Method` ou d'un `DTML Document` ;
2. les variables particulières `document_id` et `document_title` (sans cela, une `DTML Method` risquerait de ne pas avoir accès à son propre identifiant ou à son propre titre) ;
3. les propriétés (ou attributs) du `DTML Document` contenant le code DTML (une `DTML Method` ne peut pas porter de propriétés, excepté `title` et `id`) ;
4. les attributs du dossier contenant l'objet contenant lui-même le code DTML, et ainsi de suite jusqu'à la racine du site (au travers de l'acquisition) ;
5. les méthodes définies par Zope lui-même ;
6. les variables de requêtes HTTP (voir chapitre 8) ;
7. les variables définies dans les formulaires (`REQUEST`.form) ;
8. les variables définies dans les cookies (voir chapitre 8) ;
9. le cas particulier des variables dont le nom est `URL`x, où `x` est un chiffre (voir chapitre 6) ;
10. les variables CGI (voir chapitre 8) ;
11. les en-têtes HTTP pour les variables commençant par `HTTP_` (voir chapitre 6) ;
12. les variables dont le nom est `BASE`x, où x est un chiffre (voir chapitre 8).

© Éditions Eyrolles, 2001

Insertion conditionnelle : <dtml-if>

La balise <dtml-if>

La balise <dtml-if> permet de soumettre l'exécution de code DTML à une condition – existence d'un attribut ou vérification d'une expression. Les balises <dtml-else> et <dtml-elif> se comportent comme en Python.

La balise <dtml-if> doit être fermée avec une balise </dtml-if> ou <dtml-endif>, au choix.

Par exemple :

```
<dtml-if HelloWorld>
    <H1>Hello, World !</H1>
    <dtml-var HelloWorld>
<dtml-elif Hello>
    <H1>Hello, World !</H1>
    <dtml-var Hello>
<dtml-else>
    Document HelloWorld introuvable.
</dtml-if>
```

Les balises <dtml-if> et <dtml-elif> supportent les attributs name et expr, qui ont le même comportement que pour <dtml-var>.

Remarque

Lorsque l'attribut name *d'une balise* <dtml-if> *fait référence à une propriété ou un document inexistant (comme dans l'exemple précédent), sa valeur est considérée comme fausse. C'est la façon la plus pratique de vérifier l'existence d'un document dans du code DTML.*

Cette remarque ne s'applique pas à l'attribut expr : si un attribut auquel il fait référence est inexistant, une exception est déclenchée.

Proposition négative : <dtml-unless>

Les balises <dtml-unless> et </dtml-unless> fonctionnent à l'inverse de <dtml-if> : le bloc qu'elles couvrent n'est inséré que si la condition est fausse. Ces balises ne permettent l'utilisation ni de <dtml-else> ni de <dtml-elif>.

Imbrication des balises

Voici, pour référence, les différentes architectures possibles pour l'imbrication des balises <dtml-if>, <dtml-else> et <dtml-elif>.

© Éditions Eyrolles, 2001

Remarque

Les balises `</dtml-if>` *et* `<dtml-endif>` *sont strictement équivalentes.*

Condition simple

```
<dtml-if condition>
    bloc si vrai
</dtml-if>
```

Alternative

```
<dtml-if condition>
    bloc si vrai
<dtml-else>
    bloc si faux
</dtml-if>
```

Conditions multiples

```
<dtml-if condition>
    bloc si vrai
<dtml-elif condition2>
    bloc si vrai
<dtml-elif condition3>
    bloc si vrai
...
</dtml-if>
```

Conditions multiples avec alternative

```
<dtml-if condition>
    bloc si vrai
<dtml-elif condition2>
    bloc si vrai
<dtml-elif condition3>
    bloc si vrai
...
<dtml-else>
    bloc si aucune condition vraie
</dtml-if>
```

© Éditions Eyrolles, 2001

Gestion des valeurs de retour

Il peut arriver que l'on souhaite « appeler » un document ou une méthode sans se soucier de son rendu. Par exemple, on peut envisager une méthode `faire_traitement`, qui effectue un traitement sur le serveur mais qui ne produise pas de code HTML (nous verrons de tels exemples aux chapitres 7 et 8).

Dans ce cas, on utilise, en lieu et place de `<dtml-var>`, la balise `<dtml-call>`. Celle-ci n'accepte que les deux attributs `name` et `expr` étudiés plus haut.

Inversement, il est parfois nécessaire, au sein d'une méthode ou d'un document, de renvoyer une valeur précise au lieu de retourner le rendu du document. Par exemple, un document peut tester une condition et renvoyer la chaîne « OUI » ou la chaîne « NON ». On utilise dans ce cas la balise `<dtml-return>` qui accepte les attributs `name` et `expr` :

```
<dtml-if CONDITION>
    <dtml-return "'OUI'">
<dtml-else>
    <dtml-return "'NON'">
</dtml-if>
```

Avec la balise `<dtml-return>`, il est possible de renvoyer des données de n'importe quel type : chaînes, entiers, voire des objets Zope. Par exemple, pour créer une méthode qui renvoie l'objet `magasin`, on peut écrire :

```
<dtml-return "magasin">
```

Insertion itérative : `<dtml-in>`

Python fait un usage intensif des structures de données telles que les dictionnaires et les séquences. Ainsi, en Python l'instruction `for` supporte en standard les séquences. Le DTML tire parti de cette facilité, en proposant une balise `<dtml-in>` qui, à l'instar du `for` de Python, parcourt les listes.

Or, l'insertion de listes dans du code HTML est un problème récurrent : ainsi, que l'on construise un catalogue, un Caddie électronique, un carnet d'adresses virtuel ou un serveur d'e-mails en ligne, l'usage des listes en HTML est omniprésent. Notons enfin qu'un tableau, en HTML, est une liste de lignes.

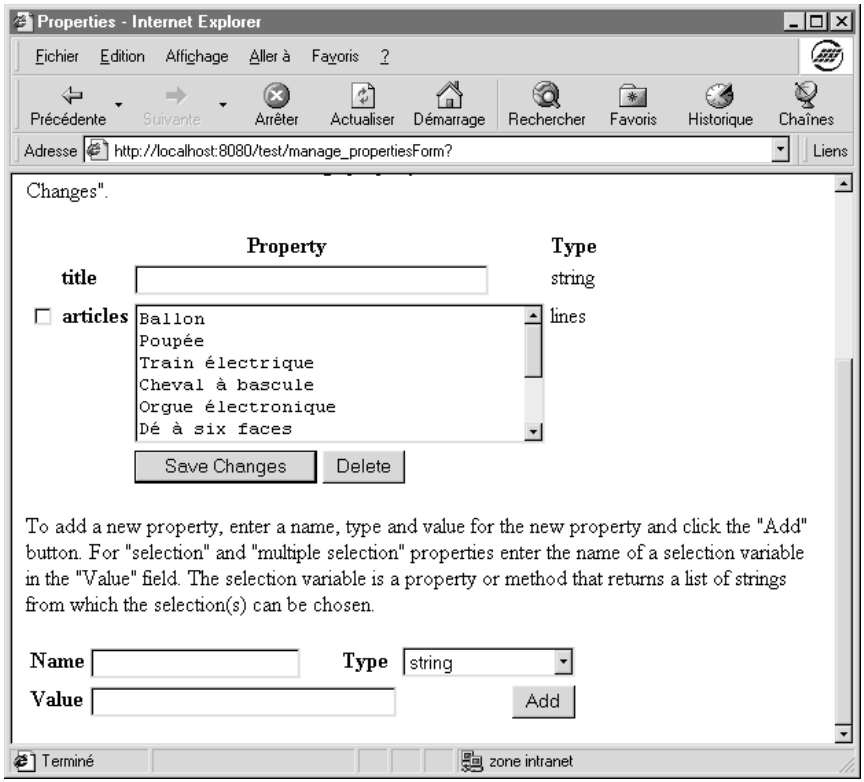
Introduction aux listes en Zope

Afin de présenter les listes, nous allons imaginer un exemple : l'affichage d'une liste d'articles. Pour commencer, on va créer un `DTML Document` vide appelé `catalogue`, puis lui affecter une propriété `articles` de type `lines` comme suit :

© Éditions Eyrolles, 2001

Entrer une dizaine d'articles dans la propriété `articles`.

Figure 4-1
Propriété `articles` *de l'objet* `catalogue`



Dans le corps du DTML Document, on entre le code suivant :

```
<dtml-var standard_html_header><dtml-in articles>
➥<dtml-var sequence-item><BR></dtml-in><dtml-var standard_html_footer>
```

Puis on clique sur l'onglet View : le rendu de la page HTML contient alors la liste des articles que l'on a entrés. On peut revenir dans l'interface de saisie des propriétés et rajouter quelques articles : la page HTML est mise à jour automatiquement lorsqu'elle est rechargée.

Spécification de la liste

La balise `<dtml-in>` fonctionne elle aussi avec les attributs `name` et `expr` (même syntaxe, même comportement). Mais elle n'accepte que les attributs, méthodes ou fonctions renvoyant des listes python. L'attribut `articles`, de type `lines`, est stocké sous forme de liste par Zope.

© Éditions Eyrolles, 2001

Il est toutefois possible de spécifier toute autre variable ou expression valide, par exemple :

```
<dtml-in "['Ballon', 'Poupée']">
    <dtml-var sequence-item><BR>
</dtml-in>
```

Remarque

N'oubliez pas, dans les exemples suivants, de rajouter `<dtml-var standard _html_header>` *et* `<standard_html_footer>` *au début et à la fin du* `DTML Document`.

En revanche, si l'on tente de parcourir une structure autre qu'une séquence, Zope renvoie une erreur au moment du rendu du DTML. Par exemple, créons une propriété `articles_texte` dans le document, de type `text` (qui ne peut contenir qu'une seule chaîne – pouvant contenir des sauts de ligne – et non une séquence de chaînes), puis copions-y le contenu de la propriété `articles`. On remplace ensuite le corps du `DTML Document` comme suit :

```
<dtml-in articles_texte>    <dtml-var sequence-item><BR></dtml-in>
```

Si on essaye d'afficher le document, Zope renvoie une erreur, spécifiant que les chaînes de caractères sont interdites dans une balise `<dtml-in>`. On remplace alors le code comme suit :

```
<dtml-in _".string.split(articles_texte, '\n')">    <dtml-var sequence-item><BR>
➥</dtml-in>
```

Alternatives : `<dtml-else>`

Lorsqu'une liste est vide, il est souvent pratique de le signaler d'une manière ou d'une autre à l'utilisateur. La balise intermédiaire `<dtml-else>` peut être utilisée à ces fins :

```
<dtml-in "[]">
    <dtml-var sequence-item><BR>
<dtml-else>
    <I>Aucun article.</I>
</dtml-in>
```

Remarque

Effacer le contenu de la propriété `articles` *ne suffit pas : une propriété de type* `lines` *contient au moins un élément – même s'il est vide.*

Contrairement à la balise `<dtml-if>` qui ne renvoie pas d'erreur lorsqu'une propriété passée à l'attribut `name` n'existe pas, la balise `<dtml-in>` renvoie une erreur si la propriété `articles` est supprimée.

© Éditions Eyrolles, 2001

Ainsi, pour tester le cas où `articles` n'existe pas, il faudrait écrire :

```
<dtml-if "_.has_key('articles') and articles">
    <dtml-in articles>
        <dtml-var sequence-item><BR>
    </dtml-in>
<dtml-else>
    <I>Aucun article.</I>
</dtml-if>
```

Ce problème, bien que relativement simple et soluble (moins simplement, avec l'usage de `_.has_key()` pour tester l'existence de `articles` dans l'espace de noms), montre que le DTML ne doit surtout pas être pris pour un langage de programmation. Pour que le DTML reste simple, il est indispensable de conditionner le bon fonctionnement d'une page à un ensemble d'assertions simples : ici, imposer l'existence de la variable `articles`, par exemple. Nous verrons que la gestion des exceptions en DTML permet de vérifier ces assertions et de simplifier très sensiblement le code : ainsi, le plus souvent, il sera suffisant d'écrire le code suivant, en admettant que la variable `articles` existe :

```
<dtml-in articles>
    <dtml-var sequence-item><BR>
<dtml-else>
    <I>Aucun article.</I>
</dtml-in>
```

Parcours d'une liste d'objets

Dans notre exemple, nous avons parcouru les éléments d'une propriété. Si nous suivons notre idée qui consiste à établir une liste d'articles, cela va poser un problème : il est probable que ces articles seront associés à des références et ces références à un prix unitaire ; ou bien, nous associeront une page explicative pour chaque article. Il faut donc un moyen de parcourir des objets au lieu de parcourir des chaînes.

On ne peut pas gérer cela avec une seule liste de chaînes de caractères : il faut stocker les articles dans des objets Zope, avec un objet distinct par article. Pour ce faire, on crée un sous-dossier `articles` dans le dossier courant, puis, dans ce dossier, on crée un `DTML Document` par article, en appelant chaque document par une référence arbitraire (par exemple, `ART_01`, ART_02, etc.). Pour chaque document, on supprime le contenu inséré par défaut : tous les documents doivent être vides.

Pour chaque document, on utilise la propriété `title` pour indiquer le nom de l'article.

© Éditions Eyrolles, 2001

Par exemple, nous pourrions avoir :

id	title
ART_01	*Poupée*
ART_02	*Train électrique*
ART_03	*Ballon*
ART_04	*Bilboquet*
ART_05	*Billes*
ART_06	*Cheval à bascule*
ART_07	*Dé à six faces*
ART_08	*Orgue électronique*
ART_09	*Corde à sauter*
ART_10	*Dînette*

Les objets `Folder` de Zope possèdent une méthode intéressante, `objectIds()`, qui renvoie la liste des objets contenus dans un dossier (sous forme d'une liste de chaînes, chaque chaîne contenant l'identifiant d'un objet). Ainsi, pour parcourir la liste des objets du dossier `articles`, il faudrait saisir :

```
<dtml-in "articles.objectIds()"
    <dtml-var sequence-item><BR>
</dtml-in>
```

Ce code affiche la liste des références des articles. La méthode `objectIds` renvoie tous les types d'objets, ce qui n'est pas forcément souhaitable dans notre cas : nous voulons uniquement les objets de type `DTML Document`, même si le dossier `articles` contient d'autres types d'objets (images, dossiers, méthodes DTML, etc.). Il est possible de spécifier à la méthode à quels types d'objets elle doit se restreindre :

Astuce

Pour spécifier plusieuers types d'objets à parcourir, utilisez la syntaxe d'une liste Python, par exemple : `(['DTML Document'], ['DTML Method']).`

```
<dtml-in "articles.objectIds(['DTML Document'])"
    <dtml-var sequence-item><BR>
</dtml-in>
```

© Éditions Eyrolles, 2001

Variables d'itération

Élément courant

En Python, lorsque l'on utilise une boucle `for`, on peut choisir le nom de la variable qui va successivement représenter chacun des éléments de la liste. En DTML, cette facilité n'existe pas : pour toutes les boucles `<dtml-in>`, chacun des éléments de la liste est stocké dans une variable appelée `sequence-item`. C'est la variable à laquelle nous faisons référence pour insérer le nom de l'article.

> **Remarque**
>
> *Par quelque bizarrerie,* `sequence-item` *n'est pas un nom de variable valide en Python. Il n'est donc pas possible de l'utiliser directement dans une expression et* `<dtml-var "sequence-item">` *provoque une erreur. Pour l'utiliser au sein d'une expression, il faut y accéder au travers de l'espace de noms :* `<dtml-var "_['sequence-item']">` *est une expression valide, équivalente à* `<dtml-var sequence-item>`.

Il est possible d'imbriquer des boucles : dans ce cas, conformément aux règles de résolution de noms, la variable `sequence-item` correspondra à la variable de la boucle la plus imbriquée. Par exemple, pour obtenir un produit cartésien :

```
<dtml-in "articles.objectIds()">
    ARTICLE: <dtml-var sequence-item><BR>
    <dtml-in "articles.objectIds()">
         <dtml-var sequence-item><BR>
    </dtml-in>
</dtml-in>
```

> **Remarque**
>
> *Il n'est pas possible de récupérer directement, dans la boucle la plus imbriquée, la valeur de* `sequence-item` *de la boucle de niveau supérieur. Pour ce faire, il faudrait affecter* `sequence-item` *de la première boucle à une autre variable, accessible dans la seconde : nous verrons comment y parvenir avec l'instruction* `<dtml-let>`.

Attributs des objets parcourus

Pour avoir le nom des objets, il faudrait récupérer leur propriété `title`. Les objets `Folder` possèdent une autre méthode, `objectValues`, similaire à `objectIds`, mais qui, au lieu de renvoyer une liste de chaînes, renvoie directement la liste des objets.

En outre, lors du parcours d'objets Zope, la balise `<dtml-in>` place systématiquement au-dessus de l'espace de noms les attributs définis par l'objet parcouru.

© Éditions Eyrolles, 2001

Ainsi, nous affichons la liste des titres avec le code suivant :

```
<dtml-in "articles.objectValues(['DTML Document'])">
    <dtml-var title><BR>
</dtml-in>
```

Nous pouvons aussi préciser la référence au moyen du code suivant :

```
<dtml-in "articles.objectValues(['DTML Document'])">
    <dtml-var id> <dtml-var title><BR>
</dtml-in>
```

Il est possible, au moyen de la variable `sequence-item`, d'insérer le contenu des objets parcourus.

On commence par modifier le corps de chaque `DTML Document` des articles, en entrant une description sommaire de l'objet. Puis, on insère le code de parcours des articles suivant :

```
<dtml-in "articles.objectValues(['DTML Document'])">
    <dtml-var id> <dtml-var title> <dtml-var sequence-item><BR>
</dtml-in>
```

Il est même possible, dans les documents du dossier `articles`, d'insérer du DTML : le code sera interprété correctement. Par exemple, on peut insérer le titre d'un article dans sa description en ajoutant :

```
<dtml-var title>
```

Parcourir des objets pour afficher leur contenu est courant sous Zope. Ainsi, dans le dossier `magasin`, on peut écrire une méthode `index_html` qui affiche tous les `DTML Document` les uns à la suite des autres. Pour ce faire, on crée une `DTML Method` (et non un `DTML Document`) :

```
<dtml-var standard_html_header>
<dtml-in "objectValues(['DTML Document'])">
    <H1><dtml-var title_or_id></H1>
    <dtml-var sequence-item>
</dtml-in>
<dtml-var standard_html_footer>
```

Attention

C'est bien une méthode et non un document DTML qu'il faut créer ici : si `index_html` *était lui-même un* `DTML Document`*, on aurait récursivement une boucle infinie car le document se serait lui-même trouvé dans le parcours. Le DTML interprété aurait déclenché une nouvelle boucle, qui aurait trouvé le document* `index_html`*, et ainsi de suite.*

© Éditions Eyrolles, 2001

En cliquant sur l'onglet View, le document `catalogue` apparaît à l'écran. On peut maintenant, dans l'objet `catalogue`, retirer les ligne `<dtml-var standard_html_header>` et `<dtml-var standard_html_footer>` : l'en-tête HTML est systématiquement ajouté lorsque l'on consulte l'URL *http://localhost:8080/magasin*.

À ce stade, pour réviser l'acquisition, on se place dans le dossier `articles` et on clique sur l'onglet View. Tous les articles apparaissent les uns à la suite des autres : la méthode `index_html` a été acquis.

Début et fin de liste

L'instruction DTML `<dtml-in>` procure certains avantages par rapport au `for` de Python : ainsi, avec Python il n'est pas possible, dans une boucle `for`, de détecter le début ou la fin de la séquence que l'on parcourt. Cette fonctionnalité est particulièrement utile dans les tableaux HTML.

Reprenons notre liste d'articles : comment l'afficher dans un tableau ? La réponse évidente est celle-ci :

```
<TABLE BORDER="1">
<TR><TH>Article</TH></TR>
<dtml-in articles>
  <TR>
  <TD>
    <dtml-var sequence-item><BR>
  </TD>
  </TR>
<dtml-else>
    Aucun article.
</dtml-in>
</TABLE>
```

Mais, si le dossier `articles` était vide, le code HTML suivant serait généré :

```
<TABLE BORDER="1">
<TR><TH>Article</TH></TR>
    Aucun article.
</TABLE>
```

Le tableau apparaît (titre des colonnes) et le texte « Aucun article » n'est pas inscrit dans une cellule. Le comportement idéal consisterait à ne *pas* faire apparaître de tableau si aucun article n'est trouvé. On peut utiliser un `<dtml-if>` pour tester la liste avant de créer le tableau, mais ce n'est pas très élégant : il serait plus judicieux de créer le tableau (et ses titres) lors du premier parcours de la boucle `<dtml-in>`, et de le refermer au dernier parcours.

© Éditions Eyrolles, 2001

Les variables `sequence-start` et `sequence-end` fournissent cette indication : ce sont des variables booléennes, qui sont vraies si l'élément courant est respectivement le premier ou le dernier de la séquence. Ainsi, on peut écrire :

```
<dtml-in "articles.objectValues(['DTML Document'])">
    <dtml-if sequence-start>
        <TABLE BORDER="1">
        <TR><TH>Article</TH></TR>
    </dtml-if>

    <TR><TD><dtml-var title></TD></TR>

    <dtml-if sequence-end>
        </TABLE>
    </dtml-if>
<dtml-else>
    Aucun article
</dtml-in>
```

Lors du parcours du premier élément, `<sequence-start>` est vraie, et l'en-tête du tableau est inséré. Lors des autres parcours, `<sequence-start>` est fausse, et l'en-tête du tableau est omis.

Si la séquence est vide, le texte `Aucun article` est directement inséré, sans passer par le corps du `<dtml-in>`, et donc sans que l'en-tête du tableau ne soit inséré.

Index d'un élément de liste

Il est parfois utile d'avoir l'index de l'élément que l'on est en train de parcourir (le premier élément ayant l'index 0, le deuxième l'index 1, etc.). La variable `sequence-index`, définie à l'intérieur d'une boucle `<dtml-in>`, contient cette information.

Dans l'exemple précédent, on peut ainsi remplacer la ligne :

```
    <TR><TD><dtml-var title></TD></TR>
```

par :

```
    <TR><TD><dtml-var sequence-index> : <dtml-var title></TD></TR>
```

pour observer le résultat.

La balise `<dtml-in>` est dotée d'une autre variable pour gérer les index. Considérons le cas du parcours d'un dictionnaire, par exemple : `{'ART_01': 'Poupée', 'ART_02': 'Train électrique'}`. La balise `<dtml-in>` ne fonctionnant qu'avec une liste ou une séquence, l'instruction suivante provoque une erreur :

```
<dtml-in "{'ART_01': 'Poupée', 'ART_02': 'Train électrique'}">
    ...
</dtml-in>
```

© Éditions Eyrolles, 2001

Il faut, pour parcourir les éléments du dictionnaire, utiliser la méthode `items()` de Python :

```
<dtml-in "{'ART_01': 'Poupée', 'ART_02': 'Train électrique'}.items()">
    <dtml-var sequence-index> : <dtml-var sequence-item><BR>
</dtml-in>
```

Dans ce cas, curieusement, la variable `<sequence-item>` n'affiche que les noms des articles, et pas leur référence. En effet, lorsque la boucle `<dtml-in>` est parcourue avec des tuples (comme c'est le cas ici avec la méthode `items()` qui renvoie un tuple clé), le premier indice du tuple est placé dans la variable `sequence-key`, le second indice est placé dans `sequence-item` :

```
<dtml-in "{'ART_01': 'Poupée', 'ART_02': 'Train électrique'}.items()">
    <dtml-var sequence-key> : <dtml-var sequence-item><BR>
</dtml-in>
```

Cette particularité permet de parcourir très simplement des dictionnaires, en gardant à chaque fois la clé et la valeur dans une variable distincte. De plus, on dispose de deux méthodes des objets `Folder` : `objectValues()` et `objectKeys()`. Et, par chance, une troisième méthode : `objectItems()`, renvoie une liste de tuples dont le premier indice est l'identifiant de l'objet (propriété `id`), et le second indice est l'objet lui-même.

Il est donc désormais possible d'écrire :

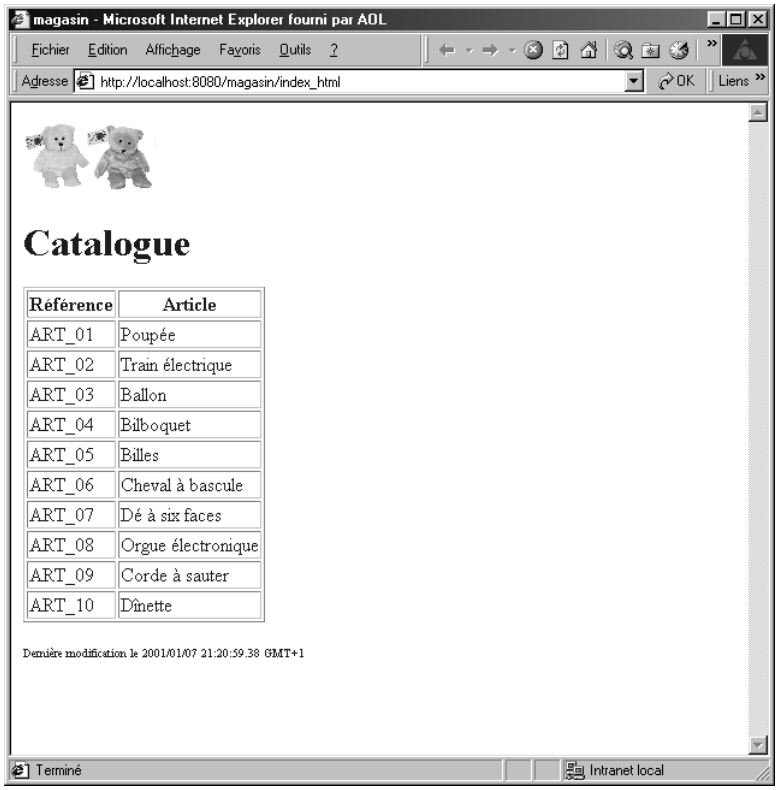
```
<dtml-in "articles.objectItems(['DTML Document'])">
    <dtml-if sequence-start>
        <TABLE BORDER="1">
        <TR>
            <TH>Référence</TH>
            <TH>Article</TH>
        </TR>
    </dtml-if>

    <TR>
        <TD><dtml-var sequence-key></TD>
        <TD><dtml-var title></TD>
    </TR>

    <dtml-if sequence-end>
        </TABLE>
    </dtml-if>
<dtml-else>
    Aucun article
</dtml-in>
```

© Éditions Eyrolles, 2001

Figure 4-2
Aperçu du dossier `magasin` *résultant*



Remarque

Dans cet exemple, nous aurions pu remplacer la ligne `<dtml-var sequence-key>` *par* `<dtml-var id>`*, car la « clé » des objets parcourus est portée par leur propriété* `id`*.*

En guise d'exercice, on peut éditer quelques `DTML Document` d'articles, en leur ajoutant une propriété `prix` de type `integer` (il n'est pas nécessaire de le faire pour tous les documents). Puis, on peut rajouter, dans le document `catalogue`, une colonne au tableau mentionnant le prix ou affichant « N/A » si aucun prix n'a été défini pour l'article (voir figure 4-3).

© Éditions Eyrolles, 2001

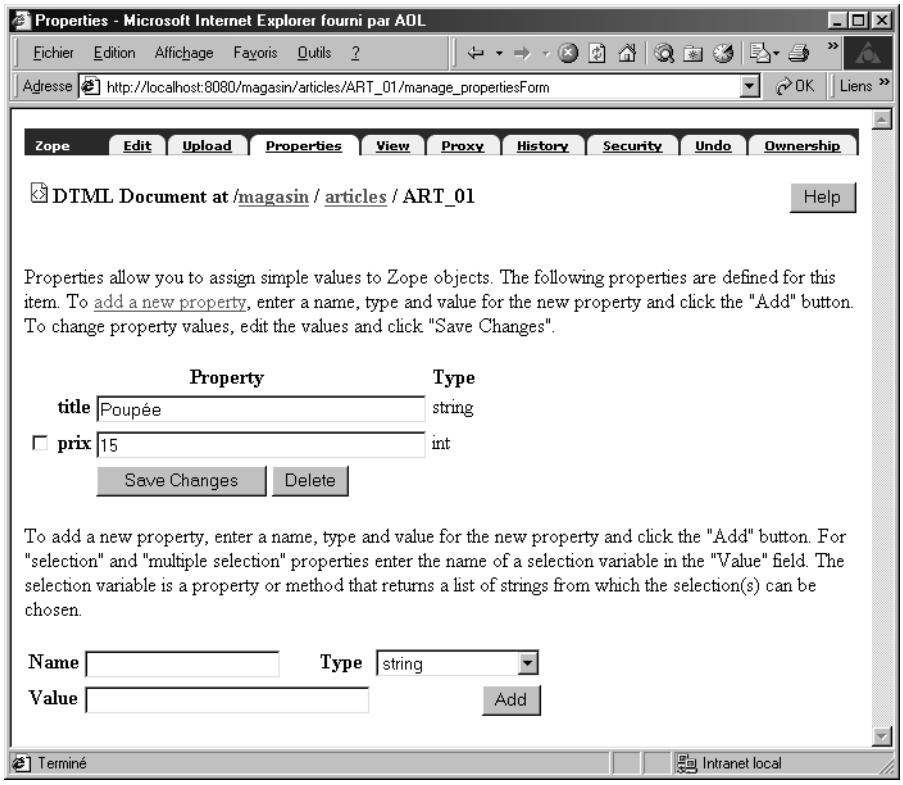


Figure 4-3
Ajout d'une propriété `prix` *sur un* `DTML Document`

Tri

La balise `<dtml-in>` permet de trier simplement les objets rendus, en spécifiant dans la propriété `sort` de la balise le nom de l'attribut sur lequel s'effectue le tri.

Ainsi, pour trier les articles par ordre de prix (croissant), il suffit de remplacer :

```
<dtml-in "articles.objectItems(['DTML Document'])">
```

par :

```
<dtml-in "articles.objectItems(['DTML Document'])" sort="prix">
```

Pour trier dans l'ordre décroissant, on utilise l'attribut `reverse` :

```
<dtml-in "articles.objectItems(['DTML Document'])" sort="prix" reverse>
```

> **Remarque**
> *Lorsque* `sort` *n'est pas spécifié, les éléments de la liste ne sont pas triés, mais sont parcourus dans le même ordre que le résultat de la méthode* `objectItems`.

© Éditions Eyrolles, 2001

Traitement par lots

Pour l'instant, le catalogue se compose d'une dizaine d'articles. Mais s'il venait à grossir, que se passerait-il lors de sa consultation ? Une énorme page HTML apparaîtrait, longue à charger, et longue à regarder.

Zope propose avec la balise `<dtml-in>` toute une série de variables pour disposer les informations page par page, où chaque page présente un ensemble limité d'informations.

Les attributs `size` et `start`

L'attribut `size` de la balise `<dtml-in>` permet de limiter le nombre d'éléments parcourus dans le corps de la boucle. Ainsi, si on remplace la ligne :

```
<dtml-in "articles.objectItems(['DTML Document'])">
```

par :

```
<dtml-in "articles.objectItems(['DTML Document'])" size="3">
```

seuls les trois premiers articles apparaissent.

> **Remarque**
> *Le nombre maximal d'articles affichés peut excéder 3, pour éviter la présence d'éléments orphelins sur une page à la suite de cet article : voir plus loin dans ce chapitre la description de l'attribut* `orphan`.

La propriété `start` permet de spécifier un autre élément que l'élément 0 (début de la liste) pour commencer l'énumération. Ainsi, la ligne suivante listerait les troisième, quatrième et cinquième articles :

```
<dtml-in "articles.objectItems(['DTML Document'])" size="3" start="2">
```

Construire une boucle de traitement par lots

Pour construire, en HTML, une page qui soit capable d'afficher trois articles à la fois, avec un lien vers les trois précédents et les trois suivants, il faut passer une variable HTTP entre les pages (au moyen d'un formulaire, par exemple), que nous appellerons `DEBUT`. Puis, utiliser cette variable pour stocker l'index de début de la liste courante, que l'on passe à l'attribut `start` de la balise `<dtml-in>` :

```
<dtml-in "articles.objectItems(['DTML Document'])" size="3" start="DEBUT">
```

La balise `<dtml-in>` définit les variables `next-sequence-start-index` ainsi que `previous-sequence-next-index`, permettant de connaître respectivement la valeur de la variable `DEBUT` pour la séquence d'après ou la séquence d'avant.

© Éditions Eyrolles, 2001

Attention

Ces deux variables ne sont définies que si un attribut `next` *(ou* `previous` *pour la variable* `previous-sequence-start-index`*) est spécifié dans la balise* `<dtml-in>`*. De plus, dans ce cas, la balise n'est pas parcourue plus d'une seule fois. Ce comportement, déroutant au premier abord, fait que l'on est pas tenu de placer le code de génération des boutons Précédent ou Suivant dans la boucle principale.*

Ainsi, pour afficher un bouton Suivant, il faut exécuter le code ci-après au début du document `catalogue` :

```
<dtml-in "articles.objectItems(['DTML Document'])" size="3" next>
    <FORM ACTION="<dtml-var URL>">
    <INPUT TYPE="HIDDEN" NAME="DEBUT" VALUE="<dtml-var next-sequence-start-index>">
    <INPUT TYPE="SUBMIT" VALUE="Suivant">
    </FORM>
</dtml-in>
```

Pour créer un bouton Précédent, la procédure est la même, mais avec la variable `previous-sequence-start-index` et l'attribut `previous` dans la balise `<dtml-in>` :

```
<dtml-in "articles.objectItems(['DTML Document'])" previous size="3" start="DEBUT">
   <FORM ACTION="<dtml-var URL>">
   <INPUT TYPE="HIDDEN" NAME="DEBUT" VALUE="<dtml-var previous-sequence-start-index>">
   <INPUT TYPE="SUBMIT" VALUE="Précédent">
   </FORM>
</dtml-in>
```

Les boucles `previous` et `next` ne sont pas parcourues lorsqu'il n'est pas nécessaire d'afficher le bouton correspondant. Se placer sur la page `catalogue` et observer le résultat…

Remarque

L'utilisation de `<dtml-var URL>` *pour indiquer la page de destination du formulaire n'est pas correcte : ainsi, lorsque l'on clique sur Suivant, l'URL de la page devient http://localhost:8080/magasin/catalogue/index_html et ce car la variable* `URL` *contient l'adresse de la page vue, qui n'est pas le dossier* `catalogue`*, mais la* `DTML Method` *nommée* `index_html`*.*

Pour éviter ce comportement, il suffit d'utiliser la variable qui stocke le parent du document courant : on remplace `<dtml-var URL>` *par* `<dtml-var URL1>` *(voir chapitre précédent pour plus d'informations sur les variables URL).*

Nous verrons ultérieurement dans cet ouvrage que l'emploi de la méthode `absolute_url` *répond plus élégamment à ce problème.*

© Éditions Eyrolles, 2001

Affiner le parcours

Lorsque beaucoup d'articles sont affichés sur chaque page, il peut être pratique de rappeler, pour chaque Précédent ou Suivant, un ou plusieurs articles du lot précédent. Ainsi, la deuxième page rappellerait le dernier article de la première, et ainsi de suite.

L'attribut `overlap` de la balise `<dtml-in>` permet de spécifier le nombre d'éléments devant se superposer entre chaque page (par défaut aucun). Ajouter `overlap="1">` dans chaque boucle pour constater le résultat.

> **Remarque**
>
> *Lorsque l'on modifie une des boucles* `<dtml-in>`*, ne pas oublier de mettre à jour les deux autres boucles dans le document : elles doivent être parfaitement identiques (sauf* `next` *et* `previous`*) pour que le traitement par lot soit cohérent.*

Lorsqu'un seul élément est listé dans une boucle, on dit qu'il est « orphelin ». Par défaut, Zope évite les orphelins en rajoutant, le cas échéant, un article à la page précédente. Ainsi, si votre catalogue comporte dix entrées, la troisième page listera quatre articles au lieu de trois, pour éviter d'avoir besoin d'une nouvelle page pour un seul article (n'oubliez pas que, sur le Web, le temps de chargement d'une page est crucial : il vaut mieux qu'une page compte quelques éléments en plus que de devoir recharger une page entière).

L'attribut `orphan` de la balise `<dtml-in>` permet de contrôler le nombre minimal d'éléments considérés comme « orphelins » (par défaut : 1). On peut, par exemple, spécifier la valeur 2 à cet attribut afin qu'aucune page ne contienne moins de trois articles.

La balise `<dtml-in>` propose d'autres variables pour renseigner l'utilisateur sur l'état du parcours : nombre total d'éléments, index des éléments en cours de parcours, etc.

Voici la liste de ces variables (disponibles pour les balises `<dtml-in>` dotées des attributs `next`, `previous` ou aucun, suivant les cas).

Variable	Description	Disponibilité
`sequence-step-size`	*Taille spécifiée pour l'attribut* `size` *de la balise* `<dtml-in>`	`next`, `previous` *et aucun*
`previous-sequence`	*Variable vraie si une boucle précédente existe, fausse sinon (donc, fausse pour la première page, vraie pour toutes les autres)*	`next`, `previous` *et aucun*
`next-sequence`	*Idem pour boucle suivante*	`next`, `previous` *et aucun*
`previous-sequence-size`	*Nombre d'éléments de la boucle précédente*	`previous`
`next-sequence-size`	*Nombre d'éléments de la boucle suivante*	`next`
`previous/next-sequence-start/end-index`	*Index de début/fin de la boucle précédente/suivante*	`previous` *ou* `next`

© Éditions Eyrolles, 2001

Enfin, les variables de la forme `previous-sequence-start-` (ou respectivement `next` et `end`) peuvent être agrémentées pour utiliser autre chose que des nombres comme index :

`previous-sequence-start-index`	*Nombres de 0 à n*
`previous-sequence-start- number`	*Nombres de 0 à n*
`previous-sequence-start-letter`	*Lettres à partir de « a »*
`previous-sequence-start-Letter`	*Lettres à partir de « A »*
`previous-sequence-start-roman`	*Chiffres romains à partir de « i »*
`previous-sequence-start-Roman`	*Chiffres romains à partir de « I »*

Voici le code du document `catalogue` faisant appel à ces fonctionnalités :

```
<TABLE>
<TR>
<TD WIDTH="100">
<dtml-in "articles.objectValues(['DTML Document'])" previous size="3"
➥start="DEBUT">
    <FORM ACTION="<dtml-var URL1>">
    <INPUT TYPE="HIDDEN" NAME="DEBUT" VALUE="<dtml-var previous-sequence-
    ➥start-index>">
    <INPUT TYPE="SUBMIT" VALUE="de <dtml-var previous-sequence-start-Roman>
    ➥à <dtml-var previous-sequence-end-Roman>">
    </FORM>
</dtml-in>
</TD>
<TD WIDTH="100">
<dtml-in "articles.objectItems(['DTML Document'])" next size="3" start="DEBUT">
    <FORM ACTION="<dtml-var URL1>">
    <INPUT TYPE="HIDDEN" NAME="DEBUT" VALUE="<dtml-var next-sequence-start-
    ➥number>">
    <INPUT TYPE="SUBMIT" VALUE="de <dtml-var next-sequence-start-Roman>
    ➥à <dtml-var next-sequence-end-Roman>">
    </FORM>
</dtml-in>
</TD>
</TR>
</TABLE>



<dtml-in "articles.objectItems(['DTML Document'])" size="3" start="DEBUT">
    <dtml-if sequence-start>
        <TABLE BORDER="1">
        <TR>
```

© Éditions Eyrolles, 2001

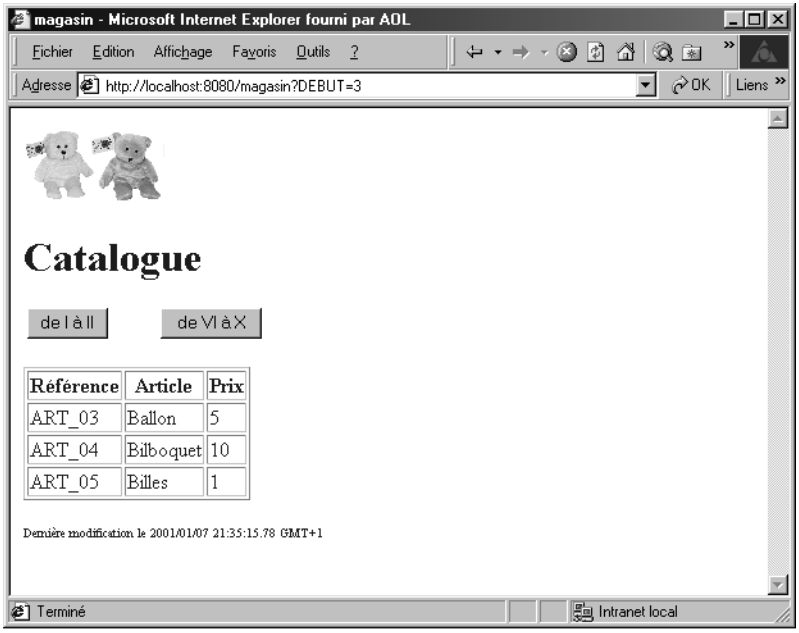
```
            <TH>Référence</TH>
            <TH>Article</TH>
            <TH>Prix</TH>
        </TR>
    </dtml-if>

    <TR>
        <TD><dtml-var sequence-key></TD>
        <TD><dtml-var title></TD>
        <TD><dtml-var prix></TD>
    </TR>

    <dtml-if sequence-end>
        </TABLE>
    </dtml-if>
<dtml-else>
    Aucun article
</dtml-in>
```

Et voici le résultat (voir figure 4-4) :

Figure 4-4
Un parcours de boucle sophistiqué



© Éditions Eyrolles, 2001

Sécurité

Dernier point concernant la balise `<dtml-in>` : avec l'utilisation des méthodes `objectIds`, `objectValues` ou `objectItems`, se pose le problème de la sécurité.

En effet, si l'un des articles possède des droits restreints (en particulier, pas de droit `View` ni `accessContentInformation` pour l'utilisateur `anonymous`), une fenêtre de demande de login (ou une page d'erreur si le login entré est erroné) s'affiche. Ce désagrément empêcherait, par exemple, notre magasin de fournir certains articles uniquement à leurs abonnés : si les utilisateurs anonymes ne peuvent certes pas voir ces articles, ils ne peuvent pas non plus consulter le catalogue.

Pour se prémunir de cette erreur, la balise `<dtml-in>` possède un attribut `skip_unauthorized` : dans ce cas, lors du parcours d'un objet pour lequel l'utilisateur courant ne possède pas de droits, l'objet sera simplement ignoré, et aucune erreur ne sera déclenchée.

En résumé…

Dans ce chapitre, nous avons passé en revue les bases du DTML : le principe des scripts DTML et l'intégration du DTML dans des pages HTML.

Nous avons vu comment insérer des variables Zope (appels de méthodes, attributs d'objets ou expressions Python) dans des documents.

Nous avons également vu comment utiliser la balise `<dtml-if>` pour utiliser une gestion rudimentaire de flot dans du code DTML.

Nous avons enfin étudié l'utilisation de la balise `<dtml-in>` pour parcourir des boucles, et surtout pour générer très rapidement du code HTML prenant en compte de nombreux problèmes de parcours de boucles : liste vide, parcours d'une boucle d'après une liste d'objets, tri et traitement par lot, permettant de répartir sur plusieurs pages l'affichage de la valeur d'une boucle.

Nous allons, dans le chapitre suivant, nous intéresser plus précisément à la programmation en DTML, en parcourant les différents problèmes que peut rencontrer un développeur Zope – et surtout en expliquant comment les résoudre efficacement.

© Éditions Eyrolles, 2001