

# Introduction à Zope

par Guillaume Piolle ([Page perso sur DVP](#))

Date de publication : 06/09/06

Dernière mise à jour : 24/09/06 (version 1.02)

Ce tutoriel constitue une première approche du serveur d'application Zope. Il est destiné à des personnes ayant une certaine culture, même basique, des principes du développement web, et qui veulent découvrir Zope.

Ce tutoriel est incomplet et ne traite que de certains outils présents dans Zope. Des notions importantes n'y sont pas traitées, comme les ZPT, les ZClasses, le Virtual Host Monster, les méthodes externes...

Avertissement.....	3
1 - Introduction.....	3
1.1 - A qui s'adresse ce tutoriel ?.....	3
1.2 - Qu'est-ce que Zope ?.....	3
1.3 - Cadre du tutoriel : une bibliothèque municipale.....	4
2 - Installation de Zope.....	4
2.1 - Télécharger Zope.....	4
2.2 - Sous Windows.....	4
2.3 - Vérification de l'installation.....	5
3 - Premier contact avec Zope.....	6
3.1 - Premier contact avec la ZMI, outils de base.....	6
3.2 - La ZODB.....	7
3.3 - Les types de composants programmables.....	8
3.4 - Le principe d'acquisition.....	9
3.5 - Les objets et leur publication.....	10
3.6 - Gestion des transactions.....	11
4 - Documents et méthodes DTML.....	13
4.1 - Une méthode DTML de base : page d'accueil de la bibliothèque.....	14
4.2 - Un peu plus de DTML !.....	18
5 - Remerciements.....	21
Annexes - .....	21
A - L'objet REQUEST.....	21
B - L'objet RESPONSE.....	24

## Avertissement

Je ne suis pas un spécialiste de Zope, et je ne prétends pas écrire un document de référence du monde Zope... Ce tutoriel n'est sans doute pas exempt de coquilles, de maladresses, de passages obscurs. Tout commentaire constructif visant à son amélioration sera donc le bienvenu.

Merci d'avance.

## 1 - Introduction

### 1.1 - A qui s'adresse ce tutoriel ?

Ce tutoriel constitue une première approche du serveur d'application Zope. Il est destiné à des personnes ayant une certaine culture, même basique, des principes du développement web, et qui veulent découvrir Zope. A l'issue de ce tutoriel vous ne serez certainement pas des pros de Zope (ou alors pas grâce au tutoriel) mais vous aurez une certaine compréhension des principes de fonctionnement de base de Zope.

Pour profiter pleinement de ce tutoriel et de Zope en général, vous devriez avoir une certaine familiarité avec HTML et la création de sites web "classiques", ainsi que des connaissances de base en programmation orientée objet et en Python. En effet Zope est entièrement objet et basé sur le langage Python, vous ne pourrez pas faire l'impasse là-dessus. Je ne peux que vous recommander **le tutoriel de Guido van Rossum**, créateur du langage, traduit en Français sur developpez.com.

### 1.2 - Qu'est-ce que Zope ?

**Zope** est un logiciel gratuit et Open Source, développé et distribué par **Zope Corporation**.

Zope est un serveur d'application objet. Vu de l'extérieur c'est en première approximation un serveur web, comme apache, mais en fait c'est bien plus que cela, à tel point qu'on ne l'utilise que rarement comme serveur HTTP : on le place en général derrière un frontal apache, pour bénéficier à la fois des fonctionnalités de Zope, et des performances d'apache. Dans un premier temps nous utiliserons le serveur HTTP intégré à Zope.

Zope signifie *Zope Object Publishing Environment*, soit "Environnement de publication d'objets Zope". C'est un acronyme récursif, comme GNU par exemple. Pour la petite histoire, Zope est le nom d'un poisson (une brème, *abramis ballerus*), trouvé dans un dictionnaire anglais-breton (l'épouse d'un des concepteurs, Paul Everitt, est bretonne).



A la différence d'apache, Zope n'est pas un serveur de publication de fichiers, mais d'objets... Zope dispose d'une base de données objets (ZODB, *Zope Object DataBase*) et ce sont certains de ces objets qui constituent les pages web d'un site, ou ses parties non visibles (traitements et données).

## 1.3 - Cadre du tutoriel : une bibliothèque municipale

Dans le cadre de ce tutoriel, je vous propose de nous imaginer responsables du site web d'une bibliothèque municipale. Evidemment nous allons développer ce site sous Zope. Nous allons commencer par un site très simple de quelques pages, puis rajouter des fonctionnalités jusqu'à obtenir une petite application web. Je ne saurais donc trop vous conseiller de suivre les étapes dans l'ordre !

Le tutoriel suppose que vous travaillez sur une seule machine, en local.

Le chapitre 2 traite de l'installation du serveur. Le chapitre 3 traite de ce qu'il est bon de savoir sur Zope avant de commencer à développer. A partir du chapitre 4, nous utiliserons Zope pour créer véritablement notre site web.

## 2 - Installation de Zope

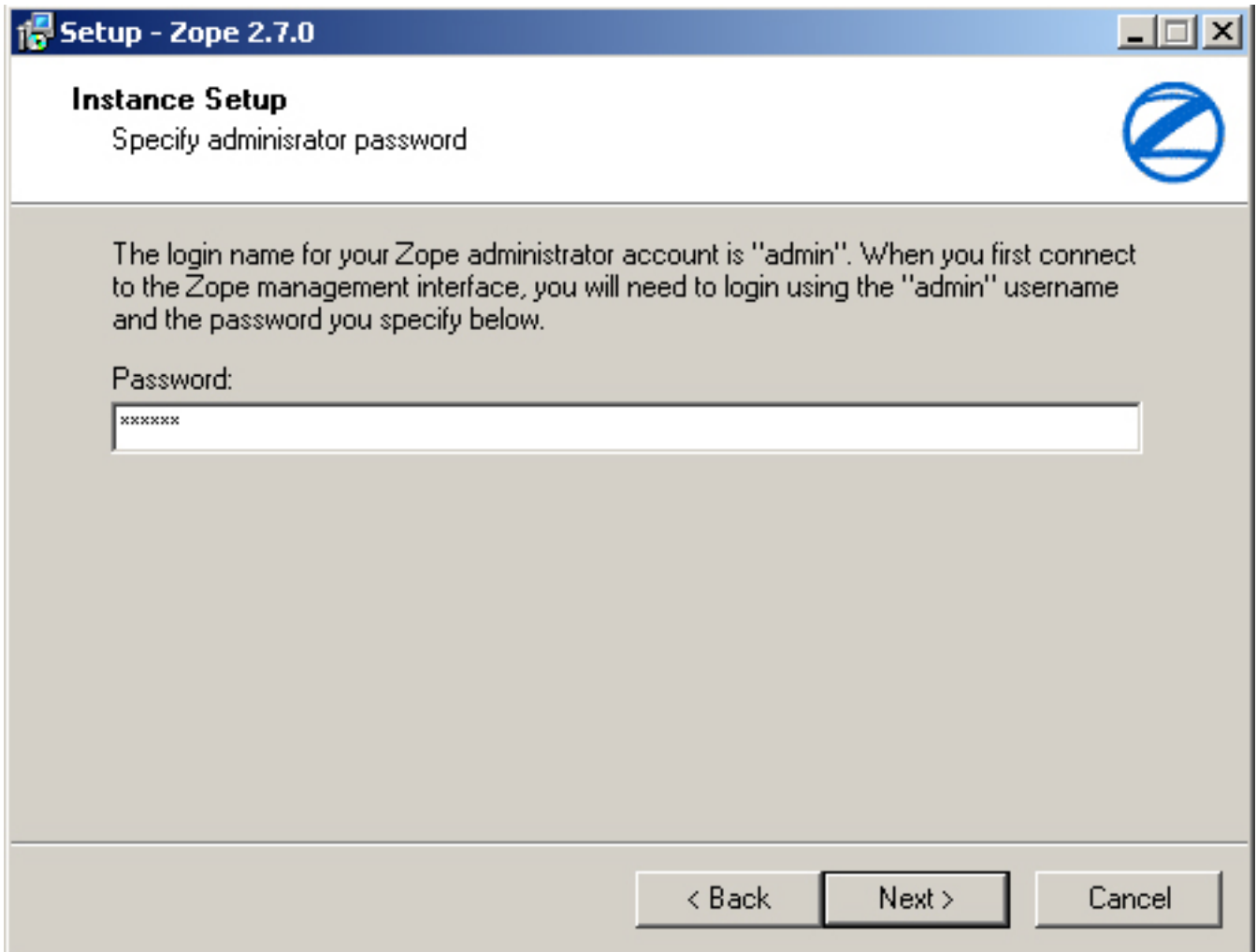
### 2.1 - Télécharger Zope

A l'heure de la rédaction de ce tutoriel, la dernière version stable est la 2.7.0. Une version 2.7.1 est en bêta, et Zope 3 est en cours de développement. Ce tutoriel concernera donc la version 2.7.0. La dernière version de Zope peut être téléchargée [ici](#).

### 2.2 - Sous Windows

Si vous installez Zope sous Windows, je vous conseille d'utiliser la version exécutable correspondante. C'est un programme d'installation classique, qui ne vous dépaysera pas et ne vous posera sans doute aucun problème particulier. Je vous conseille de conserver les options par défaut (et donc de laisser Zope s'installer en tant que service).

A ce stade, le programme d'installation vous demande le mot de passe de l'utilisateur "admin", soyez sûr de vous en souvenir, car si vous l'oubliez vous serez obligé de réinstaller Zope (*been there, done that*) :



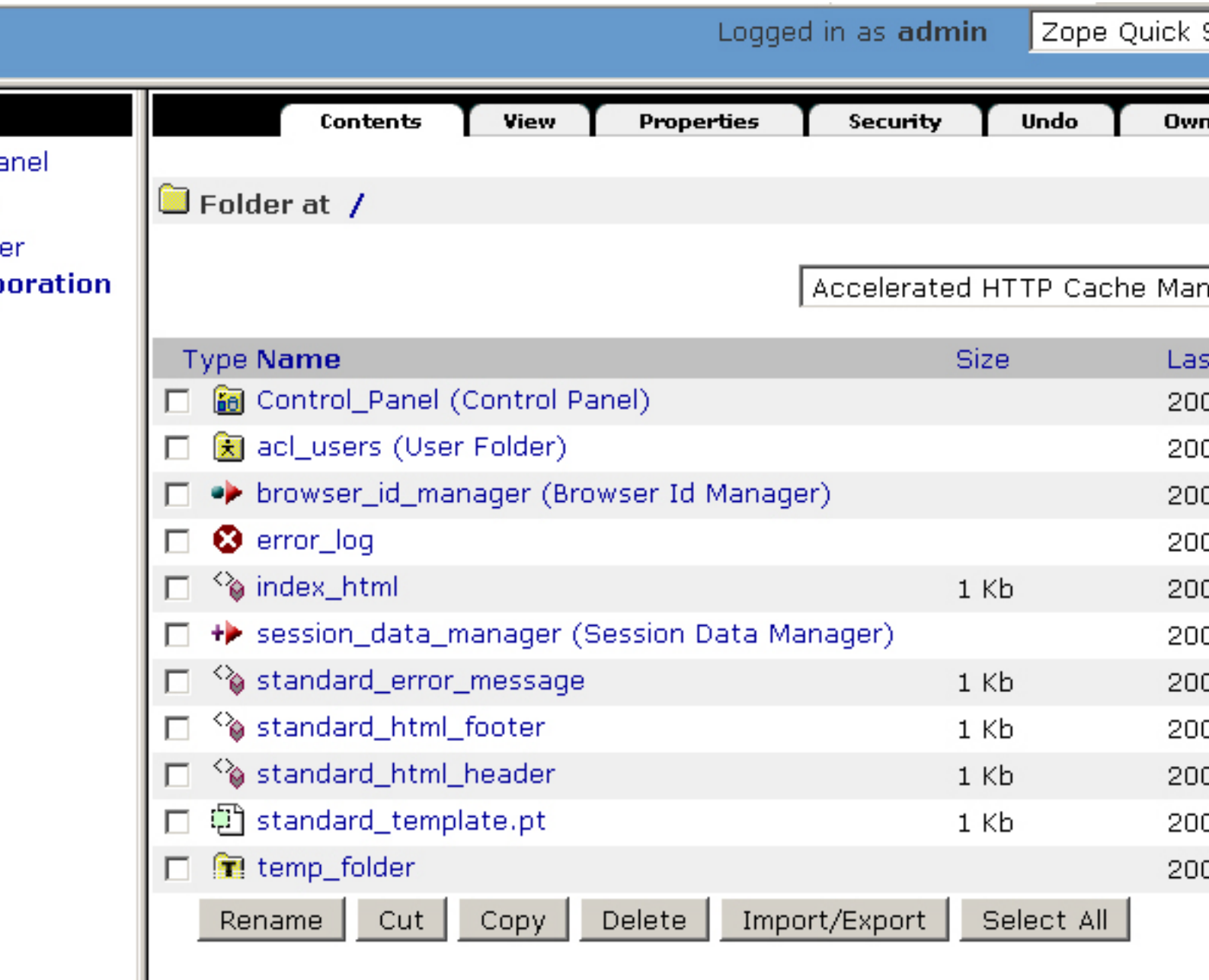
Par défaut Zope installera ses fichiers dans "c:\Program Files\Zope-2.7.0", ainsi que dans "c:\Zope-Instance" (fichiers nécessaires au lancement de Zope).

Lorsque le programme d'installation se termine, laissez-le démarrer Zope.

## 2.3 - Vérification de l'installation

Par défaut, le serveur web de Zope est mappé sur le port 8080 (au moins sous Windows).

Démarrez votre navigateur préféré. Sur <http://localhost:8080/> vous devriez obtenir une page de présentation intitulée "Zope Quick start". Dans une autre fenêtre de navigateur, allez sur <http://localhost:8080/manage> en donnant comme login "admin" et comme mot de passe, celui que vous venez de définir. Vous devriez obtenir quelque chose dans ce style :



Si c'est bien ce que vous obtenez, alors Zope est correctement installé sur votre machine.

### 3 - Premier contact avec Zope

#### 3.1 - Premier contact avec la ZMI, outils de base

L'accès à Zope par <http://localhost:8080/manage>, protégé par mot de passe, vous amène sur l'outil d'administration en ligne de Zope : la ZMI (*Zope Management Interface*). L'interface vous présente une arborescence, à la manière d'un explorateur classique. Mais soyez conscient que ce n'est pas une arborescence de fichiers : c'est une arborescence d'objets, contenus les uns dans les autres (tous descendants du *Root Folder*).

A la racine, quelques éléments intéressants à noter (leur utilisation sera détaillée plus tard) :

- Le *Control Panel*, qui nous servira à gérer les produits installés sur le serveur, qui nous donne des informations sur son fonctionnement et sa configuration
- Le composant *acl\_users*, qui sert à gérer les comptes utilisateurs (le compte admin doit être le seul configuré pour l'instant)
- Le *Browser Id Manager*, utilisé pour identifier les visiteurs de manière unique (souvent utilisé avec les sessions)
- L'*error\_log*, qui répertorie les erreurs (exceptions) survenues dans l'arborescence depuis le dernier redémarrage du serveur
- *index\_html*, la page par défaut (celle que l'on obtient actuellement sur <http://localhost:8080>)
- Le *session data manager*, qui comme on peut l'imaginer sert à gérer les sessions
- *temp\_folder*, le répertoire des fichiers temporaires (dont les fichiers de session)

Tous ces éléments (ces objets) ont chacun un type, qui apparaît notamment lorsqu'on place la souris au-dessus de leur icône, ou que l'on clique sur le composant. Ces types (ou *meta types*) correspondent aux éléments de la liste déroulante en haut à droite, qui sert à ajouter des objets dans l'arborescence. Vous pouvez donc théoriquement créer d'autres instances du même type à d'autres endroits de l'arborescence (sauf pour les objets *Control Panel* et le *Temporary Folder*). Nous verrons plus tard comment créer de nouveaux types d'objets.

Dans Zope, les objets sont identifiés par leur place dans l'arborescence, et par leur *Id* (c'est souvent le seul champ obligatoire à remplir lorsqu'on crée un nouvel objet). Les objets peuvent également avoir un titre (*Title*), qui apparaîtra entre parenthèse dans la ZMI, et qui pourra servir à l'identifier de manière plus "conviviale". Deux objets dans le même répertoire (ou autre conteneur) de la ZMI ne peuvent pas avoir le même *Id*. Ils peuvent par contre tout à fait avoir le même *Title*.

Dans chaque instance d'objet dans la ZMI, vous disposez d'onglets de contrôle. Ici pour le *Root Folder*, nous avons *Contents*, qui liste les objets fils, *View*, qui affiche l'objet si cela est possible, *Properties* qui permet d'ajuster certaines propriétés de l'objet comme son titre, ou d'en créer de nouvelles, *Security* qui permet de gérer les permissions suivant les profils d'utilisateurs, *Undo* qui permet d'annuler des actions passées, *Ownership* qui donne le propriétaire de l'objet, et *Find* qui est un outil de recherche d'objets dans l'arborescence. Vous retrouverez la majorité de ces onglets pour tous les objets Zope. Vous pourrez également en rencontrer d'autres : nous verrons qu'il est possible de créer des objets avec des onglets personnalisés.

Dans la liste des types d'objets disponibles, vous disposez entre autres du type *Folder* qui constituera vraisemblablement le squelette de notre future arborescence. Encore une fois, ce n'est pas un dossier ou un répertoire au sens où on l'entend sur le système de fichier : c'est un objet Zope, de type *Folder*, qui est contenu dans un autre objet, et qui peut contenir d'autres objets... Certains types d'objets sont des *conteneurs*, c'est-à-dire qu'ils peuvent avoir des objets fils (c'est le cas des objets *Folder*), et d'autres pas (c'est le cas par exemple des *DTML Methods*). Seuls les conteneurs ont un onglet *Content*.

Vous avez également à votre disposition un gestionnaire d'aide contextualisé ("Help !" en haut à droite), qui vous donne une aide adaptée à la page sur laquelle vous vous trouvez.

## 3.2 - La ZODB

Le "cœur" des informations contenues dans Zope est la ZODB, ou *Zope Object DataBase*. Cette base de données objet contient toutes les informations persistantes contenues dans la ZMI, les objets, les instances, les méthodes, les valeurs des propriétés. Dans le système de fichiers du serveur, la ZODB est dans un fichier nommé "Data.fs" (sous Windows, il est dans le répertoire "c:\Zope-instance\var"). C'est ce fichier que vous devez sauvegarder ou restaurer si vous souhaitez mettre en œuvre un système de backup. Si vous le supprimez, vous êtes bon pour réinstaller le serveur : en effet les informations de la gestion des utilisateurs sont dans ce fichier, et donc vous n'aurez plus d'accès configuré à l'interface d'administration (encore une fois, *been there, done that*).

Dans la ZMI, tout est objet. L'objet père est le *Root Folder*, qui contient d'autres objets, qui peuvent en contenir d'autres. Ces "objets contenus" sont considérés comme des *méthodes*, au sens POO, de l'objet père. Il faudra bien garder cela à l'esprit quand nous reparlerons de publication d'objets.

### 3.3 - Les types de composants programmables

Dans la ZMI, il y a des composants qui servent à la structure du site et de son administration, comme par exemple les *Folders*, et des composants programmables. Ces derniers peuvent l'être via la ZMI, ou alors en faisant référence à des programmes externes sur le système de fichier. Dans ce tutoriel, nous allons parler de ces différents types de composants programmables :

- Documents DTML
- Méthodes DTML
- Scripts Python
- Zope Page Templates (ZPT)
- Méthodes externes

Basiquement, les documents DTML, méthodes DTML et les ZPT servent à générer des pages web, les scripts Python et les méthodes externes sont plutôt destinés à assurer des traitements. Cependant, les uns et les autres sont capables des deux types d'opérations, traitements et vues (en fait, les documents DTML sont plutôt des composants de données, et les méthodes de composants de présentation). Dans cette liste seules les méthodes externes ne peuvent être éditées via la ZMI, elles doivent faire référence à des modules Python sur le système de fichiers.

En ce qui concerne les quatre premiers types d'objets, lorsque vous cliquez sur l'objet vous allez par défaut sur un onglet *edit*, qui est un petit éditeur de texte qui vous permet de modifier le composant. Vous n'aurez sans doute aucun mal à l'utiliser.

ZOPE Logged in as **admin** Zope Quick Start

Root Folder

- Control\_Panel
- acl\_users
- bibli
- test
- machin
- temp\_folder

ope Corporation  
resh

**Edit View Proxy History Security Undo Owners**

**DTML Method at /test**

You may edit the source for this document using the form below. You may also upload the source for this document from a local file. Click the *browse* button to select a local file to upload.

**Title**

```
<dtml-var standard_html_header>
<h2><dtml-var title_or_id> <dtml-var document_title></h2>
<p>
This is the <dtml-var document_id> Document
in the <dtml-var title_and_id> Folder.
</p>
<dtml-var standard_html_footer>
```

Save Changes Taller Shorter Wider Narrower

File  Parcourir...  
Upload File

### 3.4 - Le principe d'acquisition

Le principe d'acquisition est un "ajout" de Zope au langage Python. C'est un concept extrêmement utile dans le cadre d'une application internet, mais qui peut s'avérer déroutant, et qui peut rendre délicate la détection de certaines erreurs.

L'acquisition se résume ainsi : lorsqu'on demande à Zope une méthode "m" d'un objet "A", et qu'il ne la trouve pas (ni dans "A" ni dans un de ses ancêtres), il va aller la chercher dans l'objet conteneur de "A", puis dans le conteneur du conteneur, et ainsi de suite jusqu'à la racine. Si en définitive il ne trouve pas la méthode, Zope lancera une exception.

#### Exemple d'acquisition sous Zope

```
class A:
    def __init__(self):
        self.donnee = "A"
```

### Exemple d'acquisition sous Zope

```
def aff(self):
    print self.donnee

class B:
    def __init__(self):
        self.donnee = "B"

a = A()
b = B()
a.contenu = b
```

Avec l'exemple précédent, dans un environnement Zope, si on appelle `b.aff()`, on a une erreur. Si on appelle `a.aff()`, on obtient "A", et si on appelle `a.contenu.aff()`, on obtient "B", alors même que "b" et "a.contenu" désignent la même instance... Étonnant, non ?

C'est un principe qui joue beaucoup par exemple dans la gestion des exceptions. Lorsqu'un composant génère une exception, Zope va la logger dans un objet "error\_log" de type *Site Error Log*. Et il va chercher ce composant par acquisition : c'est donc l'objet "error\_log" le plus proche (toujours en remontant dans l'arborescence, jamais en descendant !) qui va référencer l'exception. Cela permet de créer plusieurs "error\_log" pour plusieurs sections du site, afin de le "partitionner" du point de vue des exceptions.

## 3.5 - Les objets et leur publication

Comme pour tout serveur web, le client va demander l'accès à une ressource, via une requête HTTP qui comprendra une URL très classique, par exemple "http://www.site.com/machin/truc". Dans le cas d'un serveur classique, type apache, "machin" est un répertoire dans le *DocumentRoot* du serveur, "truc" est un sous-répertoire de "machin", et apache va y chercher un fichier par défaut nommé par exemple "index.html", qu'il va envoyer par HTTP. Dans le cas de Zope, c'est très différent : dans le cas le plus simple, "machin" est un objet fils du *Root Folder*, et "truc" est une méthode (un objet fils) de "machin". C'est cet objet "truc" qui est publié par Zope (plus exactement par le composant ZPublisher)... Ce peut être un objet qui a un fils nommé *index\_html*, qui est un nom de méthode par défaut pour Zope, ou encore une méthode de "machin" qui renvoie du code HTML, du texte, ou qui fait une toute autre action...

Lorsque nous allons sur <http://localhost:8080>, Zope publie la méthode DTML de *Root Folder* nommée *index\_html*. Si nous essayons d'aller sur <http://localhost:8080/machin/truc>, nous aurons bien évidemment une erreur... Mais il suffit de créer un objet "Folder" dans *Root Folder*, auquel nous donnons l'id "machin", puis un autre objet "Folder" à l'intérieur du premier, avec l'id "truc". Réessayons : <http://localhost:8080/machin/truc>, nous obtenons la page d'accueil "Zope Quick start" ! Pourquoi ? A cause de l'acquisition ! Zope n'a pas trouvé de méthode par défaut nommée *index\_html* dans "truc", donc par acquisition il est allé en chercher dans "machin", puis dans "Root Folder". C'est donc le *index\_html* de *Root Folder* qui a été publié.

Si nous voulons changer cette page, il suffit de créer un objet du type "DTML Method", intitulé *index\_html*, dans le folder *truc* (laissez son contenu tel quel pour le moment). Si nous retournons sur <http://localhost:8080/machin/truc>, nous obtenons :

# truc

This is the index\_html Document in the truc Folder.

Terminé

## 3.6 - Gestion des transactions

Zope est un serveur totalement transactionnel et journalisé. C'est à dire que toutes les opérations élémentaires sont enregistrées, journalisées, de manière à pouvoir "revenir en arrière" en cas d'erreur ou de problème. C'est à cela que sert l'onglet *Undo* des composants de la ZMI. Si vous cliquez sur cet onglet dans un des objets de la ZMI (un répertoire par exemple), vous verrez la liste de toutes les transactions (actions indivisibles) effectuées dans cet objet, ainsi que dans tous ses enfants (ainsi l'onglet *Undo* du *Root Folder* répertorie les transactions de la totalité du serveur). La gestion des transactions est persistante, car maintenue dans la ZODB : un redémarrage du serveur ne les efface pas (à la différence, par exemple, des exceptions répertoriées dans les objets *Site Error Log*).



Folder at / [Help!](#)

This application's transactional feature allows you to easily undo changes made to the application's settings or data. You can revert the application to a "snapshot" of its state at a previous point in time.

Select one or more transactions below and then click on the "Undo" button to undo those transactions. Note that even though a transaction is shown below, you may not be able to undo it if later transactions modified objects that were modified by a selected transaction.

	<a href="#">Earlier Transactions &gt;</a>
<input type="checkbox"/> /bibli/test/manage_delObjects by <b>admin</b>	2004-06-21 11:19:21
<input type="checkbox"/> /bibli/test/addDTMLDocument by <b>admin</b>	2004-06-21 11:02:18
<input type="checkbox"/> /bibli/manage_addFolder by <b>admin</b>	2004-06-21 11:02:12
<input type="checkbox"/> /bibli/manage_addPageTemplate by <b>admin</b>	2004-06-21 09:16:05
<input type="checkbox"/> /bibli/manage_addPythonScript by <b>admin</b>	2004-06-21 09:15:50
<input type="checkbox"/> /bibli/addDTMLMethod by <b>admin</b>	2004-06-21 09:15:40
<input type="checkbox"/> /bibli/addDTMLDocument by <b>admin</b>	2004-06-21 09:15:36
<input type="checkbox"/> /bibli/manage_delObjects by <b>admin</b>	2004-06-21 09:10:26
<input type="checkbox"/> /bibli/addDTMLMethod by <b>admin</b>	2004-06-21 09:10:23
<input type="checkbox"/> /bibli/manage_delObjects by <b>admin</b>	2004-06-18 16:58:55

Comme on peut s'y attendre, la principale utilité de l'onglet *Undo* est de "défaire" des transactions. Ce n'est pas toujours possible si la transaction n'est pas la dernière effectuée : par exemple si on crée un objet, puis qu'on le supprime, on ne peut évidemment pas ensuite défaire la transaction de création. On aurait une exception du genre :

# Site Error

An error was encountered while publishing this resource.

**Error Type: MultipleUndoErrors**

**Error Value: Undo error 000000000000ab4: Undo error 000000000000ab4:  
Some data were modified by a later transaction**

---

## Troubleshooting Suggestions

- ◆ The URL may be incorrect.
- ◆ The parameters passed to this resource may be incorrect.
- ◆ A resource that this resource relies on may be encountering an error.

For more detailed information about the error, please refer to the error log.

If the error persists please contact the site maintainer. Thank you for your patience.

Pour défaire une telle action, il faudrait également défaire toutes les actions postérieures ayant modifié l'objet. Mais dans beaucoup de cas, il reste possible de défaire une transaction ancienne, ce qui est d'un confort certain !

Il est conseillé, si l'on veut défaire des transactions, d'utiliser l'onglet *Undo* de l'objet le plus proche (dans l'arborescence) des actions considérées : si on veut défaire une transaction sur une méthode DTML nommée "test" dans le répertoire "/rep1/rep2/", il vaut mieux utiliser l'onglet de "test" ou de "rep2". Cela permet de ne pas être noyé sous une liste de transactions qui ne nous concerne pas, et d'éviter de défaire par erreur une de ces transactions.

A noter également : les actions de type "undo" sont aussi des transactions que l'on peut annuler, et qui apparaissent dans la liste...

## 4 - Documents et méthodes DTML

La distinction entre documents et méthodes DTML est assez subtile. Les documents DTML sont plutôt des objets de données, ils ont des propriétés propres et peuvent réaliser un affichage. Les méthodes DTML, elles, n'ont pas de propriétés et sont plutôt des composants de présentation (d'autres objets par exemple). Le *Zope Book* nous précise que les documents DTML sont quelque peu "démodés", et qu'ils ne sont là que pour assurer une compatibilité avec les anciennes versions de Zope. Il nous assure que se limiter aux méthodes DTML n'est pas une mauvaise pratique, et que l'on ne manque rien d'exceptionnel en n'utilisant pas les documents DTML. Donc, soyons sages, restons simples, et tenons-nous-en aux méthodes !

## 4.1 - Une méthode DTML de base : page d'accueil de la bibliothèque

Donc voilà, vous êtes (nous sommes ?) le nouveau webmaster de la bibliothèque municipale du petit village de Tavachypas-sur-l'Ebouse, charmante bourgade de la Creuse, qui a décidé de promouvoir la technologie Zope.

Nous allons commencer par créer un répertoire "bibli" dans le *Root Folder*, histoire de pouvoir isoler notre travail. Vous devriez y arriver tout seul, mais si ce n'est pas le cas : placez-vous dans le *Root Folder*, dans le menu en haut à gauche choisissez *Folder*, et donnez-lui comme Id "bibli". Donnez-lui donc également le titre "Bibliothèque municipale". Laissez les deux autres cases du formulaire décochées.

Dans ce nouveau répertoire, créons également un composant *error\_log*, de type *Site Error Log* (Si cette fois vous n'y arrivez pas, laissez tomber Zope ;-). C'est lui qui enregistrera les exceptions que notre site pourra générer.

Vous allez également créer une méthode DTML, avec l'Id "index\_html" (l'Id par défaut que va chercher ZPublisher pour publier "bibli"), en laissant le contenu par défaut proposé. Votre interface devrait maintenant ressembler à ça

ZOPE Logged in as admin Zope Quick Start

Root Folder

- Control\_Panel
- acl\_users
- bibli
- temp\_folder

Accelerated HTTP Cache Manager

Type	Name	Size	Last Modified
<input type="checkbox"/>	error_log		2004-06-21 14:50
<input type="checkbox"/>	index_html	1 Kb	2004-06-21 14:50

Rename Cut Copy Delete Import/Export Select All

Et si vous allez sur <http://localhost:8080/bibli> :

## Bibliothèque municipale

This is the `index_html` Document in the Bibliothèque municipale (`bibli`) Folder.

Vous vous demandez sûrement : "Comment Zope a-t-il pu fournir un résultat si beau, si grand, si impressionnant alors que je n'ai rien mis de particulier dans `index_html` ?" Allons jeter un coup d'oeil dans cette méthode DTML.

#### index\_html

```
<dtml-var standard_html_header>
<h2><dtml-var title_or_id> <dtml-var document_title></h2>
<p>
  This is the <dtml-var document_id> Document
  in the <dtml-var title_and_id> Folder.
</p>
<dtml-var standard_html_footer>
```

Le DTML est un langage à balises, comme le HTML. D'ailleurs le DTML comprend le HTML : vous pourriez mettre uniquement du HTML dans une méthode DTML, ça se passerait très bien (par contre, si vous essayez d'afficher du DTML dans un navigateur ou un éditeur WYSIWYG, vous aurez quelques problèmes...). Nous voyons bien dans ce document quelques balises HTML, et d'autres qui n'en sont pas : toutes les balises `<dtml-xxx>`, spécifiques au DTML. On remarquera que les balises DTML uniques (sans balise fermante) ne se terminent pas par `"/>"`. Dans ce document nous ne voyons que la balise la plus courante : `<dtml-var xxx>`, qui permet d'appeler "xxx" dans l'objet dans lequel on se trouve. "xxx" peut être une méthode de cet objet, une méthode héritée ou obtenue par acquisition, ou une méthode interne à Zope.

Voyons en détail les méthodes que nous appelons ici.

- *standard\_html\_header, standard\_html\_footer* : ce sont des méthodes DTML se trouvant dans le *Root Folder*, elles sont obtenues par acquisition. Elles définissent pour l'une les en-têtes HTML et l'ouverture de la balise `<body>`, et pour l'autre la fermeture des balises. Dans la même famille, nous avons *standard\_error\_message*, chargé d'afficher les exceptions générées.
- *title\_or\_id, title\_and\_id* : ce sont des méthodes Zope qui s'appliquent au conteneur de l'objet en cours de publication. Elles affichent, pour la première, le titre du conteneur, ou à défaut son Id, et pour la deuxième, le titre, et l'Id entre parenthèses. On dispose également de *title* et de *id*, qui sont les noms des propriétés correspondantes (ne pas oublier que les propriétés sont des méthodes !)
- *document\_title, document\_id* : ces méthodes s'appliquent à l'objet lui-même. Elles affichent respectivement son titre (même vide) et son Id. Ici *document\_title* n'affiche rien, sauf si vous donnez un titre à "index\_html".

Nous allons maintenant personnaliser un peu notre page d'accueil. Nous avons vu que "index\_html" utilisait *standard\_html\_header* et *standard\_html\_footer*. Nous allons créer, dans notre répertoire "bibli", deux méthodes DTML avec les mêmes id, de manière à avoir une "charte graphique" homogène sur l'ensemble du site : en effet, tous les documents situés en aval de "bibli" récupéreront nos nouveaux footer et header par acquisition (à moins qu'un répertoire les redéfinisse). Voilà ce que nous allons mettre dans ces méthodes DTML :

#### standard\_html\_header

```
<html>
  <head>
    <title><dtml-var title_or_id></title>
    <link rel="stylesheet" href="style.css" type="text/css" />
  </head>
  <body>
```

#### standard\_html\_footer

```
<hr width="90%" />
<center>
  <font size="2" color="maroon">
    <b>
      Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact :
      <a href="mailto:webmaster@tavachypas.org">webmaster@tavachypas.org</a>
    </b>
  </font>
</center>
</body>
</html>
```

Créons également un objet de type *File*, d'id "style.css". Zope va automatiquement lui donner le type MIME "text/css".

```
style.css
body{background: #FFE4C4; color: maroon;}
p{text-align: justify; text-indent: 1cm;}
```

De cette manière, le fait d'appeler, au début et à la fin d'une méthode DTML (ou d'un autre composant), les méthodes *standard\_html\_header* et *standard\_html\_footer* permet de fixer, de manière simple et sans redondance de code, le titre de la fenêtre, du style CSS, un pied de page, et tout ce que vous ne manquerez pas d'imaginer... Associée éventuellement aux feuilles de style, c'est une méthode simple et puissante de gestion de la charte graphique d'un site. Vous pouvez évidemment étendre cette pratique en créant d'autres méthodes, appelées dans le corps de la page, pour mettre en forme tel ou tel composant, ou afficher un logo... Notre page d'accueil ressemble maintenant à ceci :

## Bibliothèque municipale

This is the `index_html` Document in the Bibliothèque municipale (bibli) Folder.

**Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact : [webmaster@tavachypas.org](mailto:webmaster@tavachypas.org)**

Nous allons rajouter un logo dans l'en-tête. Pour cela, dans le répertoire "bibli", créons un objet de type *Image*, avec l'id "logo". Dans le champ *File* du menu de création, choisissez un fichier image sur votre ordinateur. Il sera uploadé et intégré dans la ZODB. Nous allons également rajouter la ligne suivante à la fin de *standard\_html\_header* :

```
standard_html_header (extrait)
<center><dtml-var logo></center>
```

Notre page d'accueil devrait maintenant ressembler à ça :



## Bibliothèque municipale

This is the `index_html` Document in the Bibliothèque municipale (bibli) Folder.

**Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact : [webmaster@tavachypas.org](mailto:webmaster@tavachypas.org)**

De manière très simple, nous avons appelé par une balise `<dtml-var>` une méthode de l'objet "bibli", nommée "logo", qui s'est avérée être une image et qui a été gérée comme telle par ZPublisher...

## 4.2 - Un peu plus de DTML !

Nous allons améliorer un peu notre page d'accueil, à l'aide du DTML. D'abord, nous allons modifier `standard_html_footer`, de manière à afficher l'heure de dernière modification du document.

### standard\_html\_footer

```
<hr width="90%" />
<center>
  Dernière modification de la page&nbsp;: <dtml-var bobobase_modification_time fmt="rfc822">
  <br />
  <font size="2" color="maroon">
    <b>
      Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact&nbsp;:
      <a href="mailto:webmaster@tavachypas.org">webmaster@tavachypas.org</a>
    </b>
  </font>
</center>
</body>
</html>
```

Le résultat est le suivant :



# ibliothèque municipale

This is the `index_html` Document in the Bibliothèque municipale (bibli) Folder.

Dernière modification de la page : Wed, 28 Jul 2004 14:45:23 +0200

**Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact : [webmaster@tavachypas.org](mailto:webmaster@tavachypas.org)**

Pour arriver à ce résultat, nous avons utilisé la méthode `bobobase_modification_time`, native à Zope, qui retourne justement la date de dernière modification d'un objet Zope. Le nom de "bobo" vient tout droit de la préhistoire de Zope, c'est le nom d'un des module python sur lesquels s'est basé à l'origine le mécanisme de publication web de Zope. L'attribut "fmt" contient des information sur le formatage de la date. Ici nous utilisons le format défini dans la RFC 822. D'autres formats de nombres possibles :

- whole-dollars : ajoute le symbole \$ à un entier
- dollars-and-cents : nombre en \$ avec deux décimales
- aCommon : format de date standard US
- aCommonZ : (Z pour *zulu*) format de date standard US avec fuseau horaire

- Day : nom du jour en anglais
- day : numéro du jour dans le mois
- Month : nom du mois en anglais
- month : numéro du mois
- year : année (quatre chiffres)
- HTML4 : date au format W3C

Si vous êtes observateur vous constaterez que la date de "dernière modification" est en fait la date de dernière modification du conteneur, le répertoire bibli, et pas de l'objet *index\_html*. Les subtilités de l'acquisition, encore une fois... Pour afficher la date de dernière modification du fichier, il aurait fallu inclure dans *index\_html* la ligne suivante :

index\_html (extrait)

```
<dtml-var expr="_.getitem(document_id).bobobase_modification_time()" fmt="rfc822">
```

L'expression de cette balise dtml-var mérite qu'on s'y attarde un peu. Tout d'abord la notation <dtml-var method> utilisée depuis le début de ce tutoriel n'est qu'un raccourci pour spécifier un attribut (au sens XML) de la balise <dtml-var>. Les notations "standard" sont les suivantes :

dtml-var

```
<dtml-var name="nom d'objet">
<dtml-var expr="expression python/Zope">
```

Lorsque l'on écrit <dtml-var toto>, on sous-entend *name="toto"*. Lorsque l'on écrit <dtml-var "toto">, on sous-entend *expr="toto"*. Dans l'attribut name, c'est le nom d'un objet qui est mis, et cet objet est appelé. Dans l'attribut expr, c'est une expression python, qui va être interprétée.

Maintenant regardons de plus près cette expression Python. Le premier caractère est un soulignement (\_) : c'est la variable qui désigne l'espace de noms dans Zope. Dans Zope, les noms de variables **ne peuvent pas** commencer par un soulignement, à l'exception de celui-ci. Il permet en quelque sorte d'accéder aux "variables globales" de Zope. L'espace de nom contient des méthodes Zope prédéfinies (comme *getitem*), deux objets REQUEST et RESPONSE contenant les informations de la requête et de la réponse HTTP, et quelques modules python (string, math, random, whrandom). La méthode *getitem()* permet de récupérer une référence sur un objet, à partir de son identifiant. Ici on récupère le document courant (*index\_html*), et une fois que nous avons une référence nous pouvons appeler *bobobase\_modification\_time*.

Maintenant, je vous propose d'ajouter dans notre page d'accueil une mention de la provenance du visiteur. Restons basique, nous allons nous limiter à son adresse IP et à la page internet qui l'a mené jusqu'à notre site (le *referer*).

index\_html

```
<dtml-var standard_html_header>
<h2><dtml-var title_or_id> <dtml-var document_title></h2>
<p>
  Bienvenue ! Votre adresse internet est
  <dtml-var "REQUEST.envIRON['REMOTE_ADDR']">
  <dtml-if "REQUEST.envIRON.has_key('HTTP_REFERER')">
    <dtml-let ref="REQUEST.envIRON['HTTP_REFERER']">
      et vous venez de
      <dtml-var "REQUEST.envIRON['HTTP_REFERER']">,
    <dtml-if "ref.find(REQUEST.envIRON['HTTP_HOST']) >= 0">
      qui est sur notre site.
    <dtml-else>
      qui n'est pas sur notre site.
    </dtml-if>
  </dtml-let>
</dtml-if>
</p>
<p>
  This is the <dtml-var document_id> Document
  in the <dtml-var title_and_id> Folder.
</p>
<dtml-var standard_html_footer>
```

Ici, on accède à l'objet REQUEST, qui contient des informations sur la requête HTTP et l'environnement du serveur. REQUEST a entre autres pour membres un dictionnaire (Python) nommé *environ*, qui contient les variables d'environnement du serveur. REMOTE\_ADDR est l'IP du visiteur, HTTP\_HOST est l'adresse du serveur, HTTP\_REFERER est la page d'où vient le visiteur. Cette dernière entrée n'est définie que si la requête ne se fait pas directement (i.e. si c'est un lien depuis une autre page qui l'a mené ici). Donc il faut vérifier que l'entrée existe (grâce à la méthode *has\_key* des dictionnaires) avant de l'utiliser, sinon Zope va se fâcher, et lancer une exception *Key Error*. Pour plus d'informations sur l'objet REQUEST, voyez l'annexe A.

Nous avons utilisé deux nouvelles balises : dtml-let et dtml-if. dtml-let permet tout simplement de définir une variable. Ici la variable *ref* est un objet *string* qui contient le *referer*. La balise dtml-if est un if/then/else standard, organisé comme suit :

**dtml-if**

```
<dtml-if condition1>
  code à interpréter si condition1 est vraie
<dtml-elif condition2>
  code à interpréter si condition1 est fausse, et condition2 est vraie
<dtml-else>
  code à interpréter si condition1 et condition2 sont fausses
</dtml-if>
```

On peut évidemment omettre les balises <dtml-elif> et <dtml-else>, ou rajouter des balises <dtml-elif>. Dans chacune des zones de code définies, le code n'est interprété que si la condition correspondante est vérifiée. Pas de risque donc d'avoir une erreur de type *Key Error*.

On a également effectué un test pour savoir si le *referer* est sur notre site ou pas : on cherche une occurrence de HTTP\_HOST dans HTTP\_REFERER. Si Le visiteur vient d'une autre page de notre site, on aura quelque chose du genre :



## Bibliothèque municipale

Bienvenue ! Votre adresse internet est 127.0.0.1 et vous venez de <http://localhost:9080/bibli/test> sur notre site.

This is the `index_html` Document in the Bibliothèque municipale (bibli) Folder.

Dernière modification de la page : Thu, 29 Jul 2004 09:27:56 +0200

**Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact : [webmaster@tavachypas.org](mailto:webmaster@tavachypas.org)**

Et si la requête a été faite directement, on voit bien que le code de la partie <dtml-if> n'a pas été interprété :



# ibliothèque municipale

Bienvenue ! Votre adresse internet est 127.0.0.1

This is the index\_html Document in the Bibliothèque municipale (bibli) Folder.

Dernière modification de la page : Thu, 29 Jul 2004 09:27:56 +0200

**Bibliothèque municipale de Tavachypas-sur-l'Ebouse - contact : [webmaster@tavachypas.org](mailto:webmaster@tavachypas.org)**

## 5 - Remerciements

Je remercie les personnes suivantes pour leurs encouragements, leurs conseils et/ou leur travail de relecture : **Guigui\_, Zopeur.**

## Annexes -

### A - L'objet REQUEST

REQUEST est un objet de l'espace de noms DTML, qui contient les informations relatives à la requête du client et aux variables d'environnement du serveur. Il contient entre autres quatre dictionnaires : *form*, *cookies*, *other* et *environ*. Les tableaux suivants détaillent les entrées les plus courantes dans ces dictionnaires.

REQUEST.form		clé	valeur
<i>nom_champ</i>	valeur du champ <i>nom_champ</i>		

REQUEST.cookies		clé	valeur
tree-s	Variable interne à Zope : état de l'arborescence consultée (correspond à une liste d'ids des objets consultés)		
dtpref_rows	Variable interne à Zope : préférence utilisateur quant au nombre de lignes dans		

	les zones d'édition de la ZMI
<i>dtpref_cols</i>	Variable interne à Zope : préférence utilisateur quant au nombre de colonnes dans les zones d'édition de la ZMI
<i>_Zopeld</i>	Variable interne à Zope : ?? (peut-être le browser Id, qui identifie le navigateur de manière unique pour Zope)
<i>cookie_utilisateur</i>	Valeur de la variable de cookie <i>cookie_utilisateur</i>

REQUEST.other		clé	valeur
URL	URL de l'objet publié		
URL1 .. URLn	URL des conteneurs de l'objet publié, jusqu'à la racine (URL1 est l'URL du conteneur de l'objet, URL2 celle du conteneur de ce conteneur, etc.)		
BASE0 .. BASEn	Identique à URL0 .. URLn ?		
SERVER_URL	URL du serveur (un peu différent de REQUEST.envIRON['HTTP_HOST'], auquel on rajoute le protocole)		
AUTHENTICATED_USER	Nom de l'utilisateur authentifié par HTTP ('admin', 'Anonymous User'...)		
AUTHENTICATION_PATH	URI d'authentification de l'utilisateur		
PUBLISHED	Référence sur l'objet publié		

REQUEST.environ		clé	valeur
HTTP_COOKIE	Cookies en cours		
SERVER_SOFTWARE	Chaîne d'identification de la version du serveur Zope		
SCRIPT_NAME	toujours une chaîne vide		
REQUEST_METHOD	Type de requête HTTP (POST ou GET)		
HTTP_KEEP_ALIVE	Temps de Timeout du canal HTTP		
SERVER_PROTOCOL	Version de HTTP utilisée par le serveur		
QUERY_STRING	Chaîne de requête GET		
channel.creation_time	Date de création du canal HTTP		
CONNECTION_TYPE	Type de connexion HTTP		
HTTP_USER_AGENT	Chaîne d'identification de la version du navigateur		
HTTP_REFERER	URL de la ressource internet pointant vers l'objet, utilisée par le client pour y accéder		
SERVER_NAME	Nom du serveur (DNS, ou NetBios)		
REMOTE_ADDR	Adresse IP du client		
PATH_TRANSLATED	URI après réécriture		
SERVER_PORT	Port TCP du serveur Zope		
CONTENT_LENGTH	Lors de la soumission d'un formulaire par POST, nombre de signes (octets) à lire dans la requête		
HTTP_HOST	Adresse du serveur (précise le port au besoin)		
HTTP_ACCEPT	Types MIME acceptés par le navigateur		
GATEWAY_INTERFACE	Version de CGI supportée		
HTTP_ACCEPT_LANGUAGE	Liste de langues acceptées par le navigateur		
CONTENT_TYPE	Type MIME des données envoyées par le client		
HTTP_ACCEPT_ENCODING	Algorithmes de compression à la volée acceptés par le client		
PATH_INFO	URI de l'objet publié (arborescence après l'adresse du serveur, dans l'URL)		

Il y a d'autres méthodes et membres à l'objet REQUEST. Notons notamment l'objet SESSION, qui contient comme on peut l'imaginer les informations d'identification de la session en cours, et la liste PARENTS, qui est la liste des conteneurs traversés par acquisition.

## B - L'objet RESPONSE

RESPONSE est un objet de l'espace de noms DTML, qui contient les informations qui seront renvoyées au client à la fin du traitement (données HTTP). Zope y mettra bien entendu si besoin est le résultat de la publication de l'objet. L'objet implémente plusieurs méthodes que l'utilisateur peut appeler :

- `setStatus(codeStatut)` : Définit le statut de la réponse HTTP, et notamment le code numérique.
- `getStatus()` : Renvoie le code de statut HTTP courant.
- `getHeader(nomHeader)` : Renvoie la valeur courante d'un champ d'en-tête HTTP.
- `setHeader(nomHeader, valeurHeader)` : Modifie la valeur d'un champ d'en-tête HTTP.
- `addHeader(nomHeader, valeurHeader)` : Ajoute un champ d'en-tête HTTP, sans supprimer le ou les champs existants du même nom.
- `write(donnees)` : Écrit dans le champ de données à destination de l'utilisateur.
- `appendHeader(nomHeader, valeurHeader)` : Ajoute une valeur à un champ d'en-tête HTTP pouvant en contenir plusieurs.
- `setBase(URLbase)` : Définit l'URL de base à transmettre au navigateur (ce n'est pas une redirection !)
- `setCookie(nomCookie, valeurCookie, ...)` : Crée ou modifie un cookie. Les autres paramètres possibles sont *expires*, *domain*, *path*, *max\_age*, *comment*, *secure*.
- `appendCookie(nomCookie, valeurCookie)` : Ajoute une valeur à un cookie existant (les valeurs sont séparées par des ";").
- `expireCookie(nomCookie)` : Force l'expiration d'un cookie.
- `redirect(URL)` : Redirige le navigateur sur l'URL spécifiée.