

# ZEN - Documentation technique et installation

**Author :** Guillaume Savary  
**Contact :** [guillaume@\\_nospam\\_pimentech.net](mailto:guillaume@_nospam_pimentech.net)  
**Revision :** 1.50  
**Date :** 2009-03-05  
**Copyright :** 2006 PimenTech SARL  
**Tags :** crm zope

## I Présentation

Zen est une application de CRM open source - ou gestion de la relation client - elle permet d'historiser, de prévoir et d'avoir une vision d'ensemble des actions réalisées pour chaque client. Zen n'est **pas** une application d'ERP - ou progiciel de gestion intégré - elle ne gère donc pas la partie commande et facturation.

Ce document ne présente que les aspects techniques de Zen. Pour une présentation fonctionnelle, lire plutôt la documentation utilisateur disponible sur le [site de PimenTech](#).

## II Conception technique

### 1 - Zope

Zope est un serveur d'applications édité par la société américaine *Zope Corporation*. Il est disponible sous une licence Open Source, la ZPL. Il est conçu en python et permet ainsi un développement « orienté objet » de plus haut niveau. Il dispose d'une base de données objet ainsi qu'un système d'authentification propre et robuste.

### 2 - PostgreSQL

[Postgresql](#) est un système de gestion de bases de données relationnelles objet fondé sur POSTGRES. Ce dernier est à l'origine de nombreux concepts qui ne seront rendus disponibles au sein des bases de données commerciales que bien plus tard.

Postgresql, avec de bonnes performances et une stabilité presque légendaire offre en plus de nombreuses fonctionnalités modernes :

- requêtes complexes ;
- clés étrangères ;
- déclencheurs (triggers) ;
- vues ;
- intégrité des transactions ;
- contrôle des accès concurrents

L'ensemble des développements dépend fortement de la pertinence de la modélisation de la base de données. *PimenTech* définit ses modèles relationnels en XML avec un outil développé en interne (DButils : librement utilisable et téléchargeable sur notre site internet) qui génère le code SQL de création de la base ainsi qu'un schéma EER (Enhanced Entity Relationship). Ce code XML est transmis avec les sources de Zen.

### 3 - Zope-libcommon

*PimenTech* a développé une bibliothèque nommée Zope-libcommon, la fonction principale de celle-ci est de lier de façon transparente des objets Zope à PostgreSQL. Ainsi nous avons l'assurance d'avoir une base de données SQL parfaitement synchrone avec le serveur d'application. Ceci permet une sauvegarde SQL mais surtout de se servir de PostgreSQL comme d'un pont vers d'autres applications.

Il est à noter que certaines parties de Zen, comme les actions ou les contacts, ne sont stockées que sous leur forme SQL pour des raisons de performances. Zen peut ainsi contenir des dizaines de milliers de contacts et d'actions.

## III Ergonomie

### 1 - Css

Zen fait un usage intensif du CSS (Cascading Style Sheet) dans le but de séparer le plus possible le design des pages et les données. Malheureusement le support des CSS par *Internet Explorer* est assez médiocre, même dans sa version 7. Nous avons développé le style des pages de Zen en fonction des standards du W3C, si votre navigateur n'est pas capable de les respecter, ce n'est pas la peine de nous envoyer un rapport de bug !

### 2 - Ajax

Ajax (Javascript Asynchrone et XML) permet de charger du code pour générer ou compléter les pages WEB et ceci alors même qu'une page est déjà chargée dans le navigateur. Cela a pour intérêt d'accélérer l'affichage des pages en allant chercher sur le serveur seulement ce dont nous avons besoin. En conséquence cela permet d'améliorer l'ergonomie en chargeant des pages plus riches en données. Zen fait un usage intensif d'Ajax, notamment dans l'édition des actions et dans les tris de toutes les listes.

### 3 - Gecko

Malheureusement, comme nous avons vu pour le CSS, le support des technologies WEB n'est pas égal selon les navigateurs. *PimenTech* a donc choisi de supporter les navigateurs basés sur Gecko (Firefox, Seamonkey, Camino) pour les raisons suivantes :

- multi-plateforme (Windows, MacOS, Linux, BSD, Solaris...)
- bon support des CSS version 1, 2 et 3 partiellement
- très bon support d'Ajax et Javascript (Mozilla, ex Netscape co-inventeur du Javascript)

Cela dit, d'autres navigateurs sont susceptibles de faire fonctionner Zen moyennant de petits ajustements, comme Opera ou les navigateurs basés sur Khtml (Safari, Konqueror).

## IV Installation

### 1 - Pré-requis

#### a) Sur Debian Sarge

Ajouter dans le fichier `/etc/apt/sources.list` la ligne :

```
deb http://ftp.pimentech.net/debian/ sarge pimentech backport
deb http://www.backports.org/debian/ sarge-backports main
```

Pour ne pas polluer la sarge, ajouter dans le fichier `/etc/apt/preferences` :

```
Package: *
Pin: release a=sarge-backports
Pin-Priority: 200
```

puis, après l'update :

```
sh# apt-get install task-pimentech zope-libcommon zope2.7 python2.2-psycopg \
python2.2-egenix-mxdatetime python2.2-egenix-mxtools \
python2.3-psycopg zope-psycopgda zope-cookiecrunder \
python-pychart tetex-extra apache2 sysutils
```

```
sh# ln -s /usr/lib/zope/lib/python/Products/CookieCrunder \
/usr/lib/zope2.7/lib/python/Products
sh# ln -s /usr/lib/zope/lib/python/Products/ZPsycopgDA \
/usr/lib/zope2.7/lib/python/Products
```

```
sh# apt-get install postgresql-common postgresql-8.1 postgresql-contrib-8.1
```

La section backport ajoute la version postgresql-8.1, supprimer backport pour debian testing/unstable. Postgresql 8.0 fonctionne également.

Apache2 est recommandé mais Zen fonctionnera également avec Apache1.3. Les modules utilisés seront *mod\_proxy*, *mod\_rewrite* et *mod\_user*.

## b) Autres UNIX

Installer :

- Python2.3
- Apache (1.3 ou 2.0 recommandé) avec *mod\_proxy*, *mod\_rewrite* et *mod\_user*.
- Zope2.7
- Cookiecrunder
- PostgreSQL >= 8.0 (8.1 recommandé) et trgm - Trigram Matching (depuis postgresql-contrib)
- Zope-PsycopgDA (<http://initd.org/projects/psycopg1>)
- Python-Pychart (for SVG chats)
- Tetex-extra (for TeX => PDF generation)

Librairies pimentech nécessaires, impérativement en dernière version :

- [pimentech-scripts.tgz](#) ou packet debian pimentech-scripts
- [zope-libcommon.tgz](#) ou packet debian zope-libcommon
- [python-libcommon.tgz](#) ou packet debian python-libcommon
- [pimentech-dbutils.tgz](#) ou packet debian pimentech-dbutils

Normalement un simple *make install* suffit à installer ces outils.

## c) Services nécessaires

Pour fonctionner correctement *Zen* a besoin d'un serveur de mail opérationnel dont vous pouvez changer la configuration dans le fichier *globals.mak*. Il s'agit de la variable *SMTPHOST*, par défaut *localhost*.

Le service de *cron* doit également fonctionner pour la génération des rapports et l'envoi d'émailing.

## 2 - Postgresql

### a) Créer un utilisateur 'crm'

```
sh# adduser crm
sh# su - crm
```

```
sh$ mkdir src ; cd src
sh$ wget http://ftp.pimentech.net/src/crm.tar.gz
sh$ tar xzf crm.tar.gz
```

```
# zen_crm comme lien vers la version de Zen à installer
sh$ ln -s zen_crm_X.Y zen_crm
```

## b) Paramétrage SQL

Paramétrer son `pg_hba.conf` (droits d'accès à Postgresql) - si besoin -

Voici un fichier type, notez la nécessité de la ligne `host all 127.0.0.1 md5` si votre base Postgres est sur localhost et de l'avant dernière `host all 192.168.1.0 md5` si la base est sur le réseau local. Zen n'utilise pas les connexions en socket Unix, mais elles sont utiles si vous faites de la ligne de commande avec `psql`.

```
sh# cat /etc/postgresql/8.1/main/pg_hba.conf

# "local" is for Unix domain socket connections only
local    all         postgres,root       ident sameuser
local    crm         crm                  ident sameuser
local    all         all                      md5

# IPv4 local connections:
host     all         all          127.0.0.1/32       md5
host     all         all          192.168.1.0/24    md5
host     all         all          0.0.0.0/0         reject
```

Mot de passe d'administration de Postgresql (si besoin) :

```
sh# su - postgres
sh$ psql -U postgres template1
```

```
template1=# alter user postgres with encrypted password 'your_postgresql_ADMIN_password';
ALTER USER
template1=# \q
```

Avant de créer la base, il est possible (et fortement recommandé) de paramétrer certains éléments de la base selon ses besoins spécifiques. Il faut pour cela modifier 3 fichiers de chargement de PostgreSQL suivant :

- `zen_crm/src/1st/fonction.lst` : pour les fonctions des employes et contacts
- `zen_crm/src/1st/type_origine.lst` : pour l'origine des cibles
- `zen_crm/src/1st/activite_societe.lst` : pour le type d'activité des sociétés
- `zen_crm/src/1st/type_cible.lst` : pour les types de cible

Si vous en ajoutez après avoir terminé l'installation, **il faudra instancier Zen à nouveau !** Ne touchez **PAS** aux autres fichiers, Zen ne fonctionnerait plus ! Ces fichiers sont du type :

UID | CODE | TITRE

Ils sont codés en **UTF-8**. Ne **PAS** mettre d'UID à Zéro !

## c) Installation de la base

Éditer le fichier `zen_crm/config/globals.mak` et indiquer les paramètres de la base & apache. Mettre au moins un mot de passe dans **DBPWD** qui sera votre mot de passe pour la base de Zen (différent de celui pour l'administration de Postgresql).

Création de la base :

```
sh$ cd ~/src/zen_crm/base/generation
sh$ make install

doing createuser.ok
password (postgres) : your_posstgresql_ADMIN_password
CREATE USER
doing create.ok
CREATE DATABASE
loading crm.ok
psql:crm.sql:11: NOTICE: CREATE TABLE créera des séquences implicites «statut_uid
psql:crm.sql:11: NOTICE: CREATE TABLE / PRIMARY KEY créera un index implicite «st
psql:crm.sql:24: NOTICE: CREATE TABLE créera des séquences implicites «object_uid
psql:crm.sql:24: NOTICE: CREATE TABLE / PRIMARY KEY créera un index implicite «ob
```

```
psql:crm.sql:191: NOTICE: CREATE TABLE / UNIQUE créera un index implicite «useri
loading vital_generation.ok
```

Maintenant que la base est créée, il faut la remplir avec quelques éléments par défaut. Cela va charger les fichiers que vous avez peut-être modifiés précédemment.

Chargement de la base initiale :

```
sh$ cd ~/src/zen_crm/base/chargement
sh$ make install
```

Certains moteurs de recherche de Zen ont besoin de faire des recherches approchées, il faut donc ajouter les fonctions de similarité de postgresql-contrib :

```
sh# cd /usr/share/postgresql/8.X/contrib/
sh# psql -U postgres crm -h localhost
```

```
crm=# \i pg_trgm.sql
crm=# \q
```

Nous en avons fini avec la base de données. Nous n'avons plus à modifier la base de données désormais, Zen va devenir maître et **elle seule** devra modifier la base. Dans le cas contraire, il faudra mettre à jour Zen pour prendre en compte les modifications manuelles faites en base. Pour cela, il faudra lancer la fonction d'update via l'url <http://url-zen/update> puis redémarrer Zope.

### 3 - Installation de Zen - le produit Zope

#### a) Petit paramétrage Python

Il faut paramétrer le fichier sitecustomize.py situé de préférence dans /usr/lib/zope2.7/lib/python/ sinon dans /usr/lib/python2.3/site-packages/dbutils/, ajouter ou modifier :

```
import sys
sys.setdefaultencoding('latin-1')
```

#### b) Installation des sources

Installation du produit Zope "ZEN" :

```
sh$ cd ~/src/zen_crm
sh$ make install
# vérification :
sh$ ls ~crm/bin/ZEN/

sh$ su
sh# ln -s ~crm/bin/ZEN /usr/lib/zope2.7/lib/python/Products
```

Droits UNIX et sécurité (Important!) :

```
sh$ su
sh# cd ~crm/src/zen_crm
sh# make security
```

#### c) Installation de Zope

Si besoin uniquement, création d'une instance de **Zope** :

Bien lire les commentaires, un nom d'instance **ZOPE** (et non ZEN !) est demandé, mettre *default* par exemple et un répertoire 'default' sera créé. Un login & pass **Manager ZOPE** sont également demandés.

```
sh# mkzope2.7instance
sh# /etc/init.d/zope restart
# vérification :
sh# ls /var/lib/zope2.7/instances/default/*
```

#### d) Installation de Zen

Zope permet d'instancier différents produits. À ne pas confondre avec l'instance de Zope créé par `mkzope2.7instance`.

Zope utilise une base de données interne, dans laquelle nous pouvons "instancier" des objets. Zen est un object instanciable, on peut même en mettre plusieurs ! Quand vous êtes dans [http://mon\\_serveur:9673/manage](http://mon_serveur:9673/manage), Zope vous propose un bouton "Add" avec une boite de sélection. Cela permet d'ajouter des instances de différents objets.

Pour que Zen fonctionne, il faudra sélectionner "Virtual Host Monster" (s'il n'en existe pas déjà un) et faites "add" (id quelconque). Pour (enfin !) créer Zen, il faudra faire la même chose en sélectionnant "ZEN" et faire "add". L'ID demandé pour Zen (par défaut la chaîne "ZEN") correspond au fichier `globals_ZEN.py` dans `~/config`.

```
sh$ cp ~/config/globals.py ~/config/globals_ZEN.py
```

#### Note

Pour créer une seconde instance de Zen `INSTANCE_2`, il faut un fichier `globals_INSTANCE_2.py` avec les paramètres de cette instance, par exemple, dans lequel nous avons changé `DBNAME` ou `DBHOST` ainsi que `TEXTWORKDIR` (= `tex_2`) et `PDFDIR` (= `pdf_zen_2`). Il faut bien sûr avoir créé une autre base dans postgresql ! On a alors 2 produits Zen [http://mon\\_serveur/ZEN](http://mon_serveur/ZEN) et [http://mon\\_serveur/INSTANCE\\_2](http://mon_serveur/INSTANCE_2) sur 2 bases différentes.

Relancer Zope, y créer une instance de Zen avec d'ID "ZEN". L'instance "ZEN" sera joignable à l'adresse `http://mon_serveur:9673/ZEN` Ajoutez également à la racine une instance de 'Virtual Host Monster' (id sans importance) s'il n'existe pas déjà.

## 4 - Installation d'Apache

Pour avoir accès aux fichiers statiques, nous avons besoin d'Apache (v1.3 ou 2.0). Modules nécessaires :

- mod\_proxy
- mod\_userdir
- mod\_rewrite

Dans votre configuration de mod\_proxy, il se peut que vous ayiez à libérer des droits « Allow » :

```
<Proxy *>
    Order deny,allow
    Deny from all
    #Allow from .your_domain.com
</Proxy>
```

Nous allons créer un Virtual Host décrivant notre configuration Zen :

```
<VirtualHost ip.ad.re.ss:80>
    ServerName zen.monserveur.com
    AddDefaultCharset UTF-8
    RewriteEngine on
    RewriteRule ^/static/(.*) http://localhost:80/$1 [P,l]
    RewriteRule ^/misc_(.*) http://localhost:9673/VirtualHostBase/http/{SERVER_NAME}
    RewriteRule ^/p_(.*) http://localhost:9673/VirtualHostBase/http/{SERVER_NAME}
    RewriteRule ^/ZEN(.*) http://localhost:9673/VirtualHostBase/http/{SERVER_NAME}
</VirtualHost>
```

encore une autre solution :

```
<VirtualHost ip.ad.re.ss:80>
    ServerAlias zen.monserveur.com
    AddDefaultCharset UTF-8
    RewriteEngine on
    RewriteCond %{HTTP:Authorization} ^(.*)
    RewriteRule ^/(.*) http://zen.monserveur.com:9673/VirtualHostBase/http/{SERVER
```

```

ProxyPass /static http://localhost
ProxyPassReverse /static http://localhost
</VirtualHost>

```

Lien pour avoir pimentech-scripts (javascript, images, css) disponible :

```

sh# ln -s /usr/local/share/pimentech /var/www/
# vérification :
sh# ls /var/www/pimentech/*

```

```
sh# /etc/init.d/apache reload
```

Pour tester Apache :

```

sh$ lynx http://localhost:80/~crm/
[DIR] Parent Directory -
[DIR] css/ 03-Oct-2006 15:11 -
[DIR] img/ 03-Oct-2006 15:11 -
[DIR] js/ 02-Sep-2005 10:02 -
[DIR] kupu/ 09-Aug-2005 10:11 -
[DIR] static/ 03-Oct-2006 15:11 -

```

## 5 - Première Utilisation

- 1) Allez sur <http://votreserveurweb/ZEN> et identifiez-vous en **Manager Zope**
- 2) Créer un utilisateur/employé en **Administrateur Zen** (les champs jaunes sont obligatoires)
- 3) Fermer le navigateur et le relancer puis se connecter avec ce nouvel employé
- 4) Créer les autres utilisateurs/employés Zen avec un profil (*Commercial, Assistant, ...*) A noter qu'un directeur de département ou l'administrateur pourront aussi être affecté comme commercial, l'inverse n'est pas vrai.
- 5) Créer un département (ou service, division). Puis affecter un utilisateur/employé comme *Assistant* dans ce département.
- 6) Enfin affecter un employé *Commercial* dans l'affectation de l'assistant.
- 7) Vous pourrez ensuite ajouter des cartouches (templates) pour l'emailing, etc...
- 8) Lisez la <http://ftp.pimentech.net/src/crm/doc/> documentation de l'utilisateur de l'utilisateur pour comprendre le fonctionnement de Zen.

## V Maintenance

### 1 - admin

Backup & crontab pour le module d'envoi des emails et génération des rapports PDF :

```

sh$ cd ~crm/admin
sh$ make start

# vérification :
sh$ crontab -l
* 7-23 * * * mailer.sh
59 1 * * * chargement.sh
35 2 * * 1 make_rapport.sh -t all -p hebdo
35 3 1 * * make_rapport.sh -t all -p mensuel
35 4 1 1 * make_rapport.sh -t dg -p annuel
45 4 1 1 * make_rapport.sh -t manager -p annuel

```

Cette crontab s'occupe de d'envoyer vos emailings avec mailer.sh. Les magnifiques *make\_rapport* s'occupent d'envoyer des rapports à différentes périodes. Utilisez "*make\_rapport -h*" pour plus de détails sur son fonctionnement. Par défaut les rapports sont envoyés par email aux différents utilisateurs concernés. *chargement.sh* quant à lui s'occupe de diverses tâches quotidiennes :

- création d'un dump de sauvegarde SQL avec dump-db.sh
- nettoyage des vieux dumps
- vérification de l'intégrité de la base Postgres avec clean-db
- nettoyage des fichiers temporaires

## 2 - faire des cartouches pour l'emailing

Les cartouches (modèles) des emails sont créés grâce aux **Zope Page Templates** incluses dans Zope. Ils doivent être écrit en **XHTML** strict. Vous pouvez lire la doc officielle ici : [Zope Page Template](#).

L'intérêt est de préparer des modèles d'email en HTML car :

- vous voulez que vos emails respectent votre charte graphique
- vos commerciaux ne savent pas forcément faire du HTML, et quand bien même, ils ont autre chose à faire
- vous ne voulez que des emails standards : il suffit de ne faire que des cartouches statiques
- tout simplement, vous gagnez du temps !

Les cartouches sont soit dans le département : ils sont alors partagés par tous les commerciaux du département, soit dans l'affectation d'un commercial : ils ne sont alors accessibles que par lui (pour la signature par exemple).

Rien ne vaut un bon exemple :

### Cartouche Statique :

Ceci est un cartouche non éditable par le commercial qui va l'envoyer :

```
<html>
  <p tal:content="structure here/HEADER"></p>
  <body>
    <p tal:content="structure here/CONTENT"></p>
  </body>
</html>
```

### Cartouche Éditable :

Le même que le statique mais avec l'ajout du texte du commercial :

```
<html>
  <p tal:content="structure here/HEADER"></p>
  <body>
    <!-- cartouche spécial "text" va inclure le message du commercial -->
    <p tal:content="structure here/text"></p>
    <p tal:content="structure here/CONTENT"></p>
  </body>
</html>
```

### Cartouches d'Inclusion :

Ces cartouches sont invisibles pour les commerciaux, ils ne servent qu'à faire des blocs à inclure dans d'autres cartouches pour faire des parties communes réutilisables.

- HEADER :

```
<head>
  ... some header stuff
</head>
```

- CONTENT :

```
<p>
    ... some xhtml stuff
</p>
<!-- un appel a un autre cartouche d'inclusion -->
<p tal:content="structure here/SIGNATURE"></p>
- SIGNATURE :
<p>

Ma signature
</p>
```

**Résultat obtenu avec le cartouche éditable :**

```
<html>
  <head>
    ... some header stuff
  </head>
  <body>
    ... some xhtml stuff

    <!-- cartouche spécial "text" va inclure le message du commercial -->
    <p>texte du commercial</p>

    <!-- un appel a un autre cartouche d'inclusion -->
    <p>
    
    Ma signature
    </p>
  </body>
</html>
```