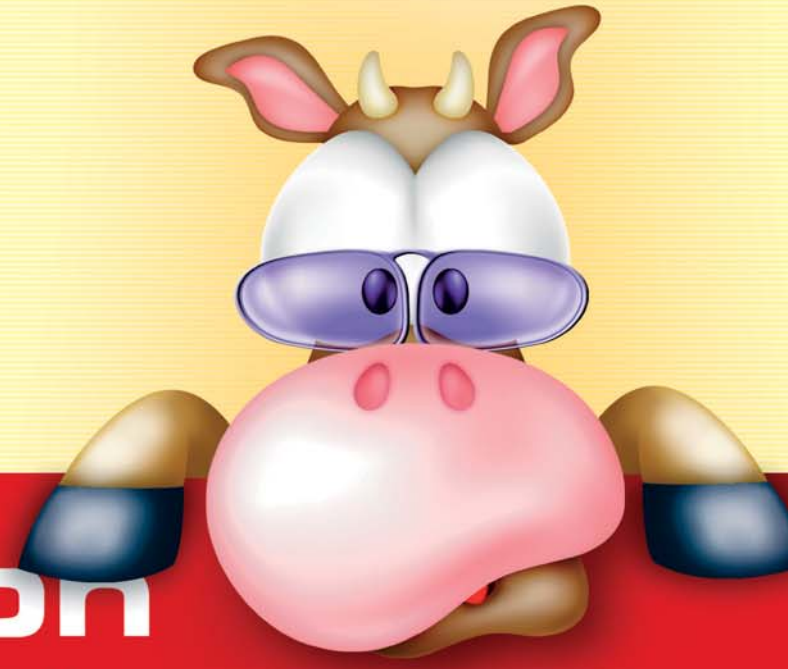
les Cahiers du Programmeur

Zope Plone

Kamon **Ayeva**
Olivier **Deckmyn**
Pierre-Julien **Grizel**
Maik **Röder**

Préface d'Alexander Limi et Alan Runyan, fondateurs du projet Plone



2e édition

EYROLLES

Préface

Quelques jours seulement après la sortie de Plone 2.0, nous sommes très heureux d'assister à la parution de ce livre, et de voir que Plone est en train de gagner une telle popularité en France. Depuis ses débuts, l'accent a été mis sur l'intégration de différentes langues afin de fournir aux utilisateurs et aux développeurs la meilleure expérience possible dans leur langue maternelle. La large diffusion de Plone en France nous montre que nous avons réussi à atteindre cet objectif, et nous sommes impatients de découvrir ce que le futur nous apportera de la part des développeurs français, qui jouent déjà un rôle important au sein de la communauté Plone.

Oui, la communauté Plone française a été l'un des tout premiers points d'entrée de Plone en Europe – depuis la sortie de la version 1.0 lors de la conférence *Solutions Linux* à Paris – et aura toujours une place à part dans l'histoire de ce logiciel. Les développeurs français ont contribué à de nombreuses parties importantes de l'infrastructure de Plone ; ils ont également réalisé un grand travail afin de faire connaître ce produit au-delà de la communauté Open Source et de prouver qu'il apportait une solution sérieuse, documentée, souple et innovatrice sur le marché de la gestion de contenu, bénéficiant d'une assistance technique de qualité.

Avec Plone 2.0, nouvelle version majeure de notre plate-forme, nous avons atteint une nouvelle étape dans de nombreux domaines. Utilisateurs, développeurs, graphistes ou administrateurs d'une solution Plone vont bénéficier d'avancées qui vous seront présentées dans cet ouvrage : l'internationalisation complète des interfaces bien sûr, mais aussi la gestion plus fine de la collaboration et du partage d'informations notamment grâce à l'ajout de la notion de groupes, la saisie *WYSIWYG* des documents issus des applications bureautiques, l'affichage sur tous types de terminaux, le découpage fin des feuilles de styles, le développement rapide de nouveaux types de documents (`ArcheTypes`), la montée en charge de la plate-forme, l'installation ou la configuration de nouveaux composants à travers des panneaux de configuration sont, entre autres, au rendez-vous !

Tout cela conserve la souplesse inédite de développement et de déploiement du serveur d'application sous-jacent, Zope. Bien sûr, la migration des sites existants Plone 1.0 est possible, afin de permettre aux utilisateurs de bénéficier des fonctionnalités de gestion de contenu, de gestion électronique de documents, de dématérialisation de procédures, de portails d'entreprise, de bases documentaires ou encore d'*e-learning*.

Bref, nous avons maintenant un socle technique solide sur lequel construire le futur de Plone. L'année écoulée a été agitée, productive, mais surtout récompensée, et la diffusion de Plone à travers le monde a dépassé nos espérances.

Cher lecteur, nous espèrons que vous apprécierez cet ouvrage, tout comme nous avons eu du plaisir à travailler en coopération avec ses auteurs afin d'amener Plone sur le marché français. Avec l'utilisation croissante de l'Open Source en Europe, il nous appartient de dessiner le futur ; nous espèrons que vous nous rejoindrez dans ce voyage.

Alexander Limi et Alan Runyan

Fondateurs du projet Plone.

Avant-propos

Plone est un outil de gestion de contenu : il sert à mettre en ligne (Internet, intranet) de l'information, pour pouvoir l'organiser et la traiter sans aucune connaissance technique liée au média manipulé.

Dans le monde des outils de gestion de contenu, Plone tient une place toute particulière, car il est très largement capable de rivaliser avec les meilleurs logiciels commerciaux du marché. Le marché de la gestion de contenu est particulièrement concurrentiel : début 2004, une étude recense pas moins de 870 offres. Sur ces offres, une grosse dizaine domine le marché, chaque produit étant ancré sur un segment de marché ou une spécificité. Les coûts de ces solutions ne sont pas du tout négligeables, avec notamment des licences à plusieurs dizaines voire centaines de milliers d'euros. Plone est atypique : c'est une solution Open Source qui vient déranger ce monde commercial, qui couvre le spectre fonctionnel de la plupart de ces outils, avec une ouverture sans précédent. Le fait que l'outil soit également gratuit commence à chambouler la donne.

Dans le monde Open Source, la gestion de contenu est un sujet qui suscite de l'intérêt. On retrouve deux grandes catégories d'outils : les outils conçus dans un garage pour répondre à un besoin particulier et dont on diffuse le code source, et les outils pour lesquels la phase amont de réflexion et de maturation a été plus aboutie, et qui tentent d'apporter des solutions génériques à des problématiques répandues. C'est dans la première catégorie que l'on trouve le plus d'outils, souvent réalisés avec PHP. On trouve parmi eux des logiciels d'excellente facture, particulièrement efficaces à résoudre en un temps très court et à un coût presque nul une problématique précise. La difficulté dont ces outils ne savent en général pas bien se sortir est leur capacité (si elle est nécessaire) à évoluer dans une direction pour laquelle ils n'ont pas été conçus : les extensions et évolutions de l'outil peuvent alors être extrêmement coûteuses, pour leur création, mais aussi pour les maintenir.

Parmi ces solutions, on retiendra notamment SPIP 1 et les familles d'outils dérivés de PHPNuke 2. Dans le monde de Zope, l'outil SquishDot 3 a eu son heure de gloire. Ces outils sont très répandus et généralement assez faciles à installer, paramétrer et utiliser.

Dans la seconde catégorie, on retrouve des solutions plus « lourdes », plus « réfléchies », pour lesquelles la courbe d'apprentissage est plus raide. La plupart de ces outils sont écrits en Java, quelques-uns avec PHP. Ces solutions sont plus utilisées sur des projets importants, mais réclament des phases de conception et de développement bien plus longues. Les coûts de réalisation (temps, machine, hommes) sont eux aussi plus grands. Nombre de ces outils fournissent une API ou un *framework*, qu'il convient de confier à une équipe de développeurs avertis. Les solutions obtenues sont généralement de très bonne facture, mais parfois un peu pénibles à faire évoluer. Parmi ces solutions, on retrouve notamment Cocoon 4, les solutions basées sur Tomcat 5 ou JBoss 6 mais aussi CMF 7. Ces outils sont bien moins répandus que les précédents, car bien plus difficiles à mettre en œuvre.

Plone appartient plutôt à la seconde catégorie, mais possède un avantage concurrentiel énorme : il est très simple à installer et est immédiatement utilisable, notamment grâce à son ergonomie hors-pair et aux très nombreux composants disponibles. La communauté qui anime Plone est très large (plusieurs milliers d'individus et/ou sociétés) et très active, présente sur tous les continents.

ALTERNATIVES Logiciels de gestion de contenu

1. http://www.spip.net
2. http://phpnuke.org/
3. http://www.squishdot.org/
4. http://cocoon.apache.org
5. http://jakarta.apache.org/tomcat/
6. http://www.jboss.org
7. http://cmf.zope.org

Plone : pour quelles applications et quels utilisateurs ?

Plone a été conçu pour être évolutif, depuis sa genèse. Son positionnement est double :

- Il est parfaitement utilisable par une PME pour mettre en place son site Internet et son intranet, avec de nombreux services : c'est l'utilisation « LEGO™ » de Plone, où l'on assemble des composants existants pour construire ses sites.
- Il convient également parfaitement comme base à des projets bien plus ambitieux, où l'on va réutiliser de nombreux composants existants, et en développer de nouveaux pour résoudre des problématiques non couvertes.

Les auteurs de ce livre utilisent quotidiennement l'outil dans ces deux configurations.

Structure de l'ouvrage

Dans cet ouvrage, nous allons nous attacher à vous faire découvrir ces deux aspects de Plone, au travers d'un exemple réel : nous allons construire ensemble un intranet d'entreprise. Le cheminement choisi est le suivant.

Le **chapitre 1** : seule partie théorique de l'ouvrage, elle s'attache à présenter la gestion de contenu du point de vue du besoin, en décrivant les avantages et inconvénients de ces solutions. Ce chapitre est destiné à tous les profils, mais sera particulièrement apprécié des décideurs.

Le **chapitre 2** : visite guidée du site à réaliser. Sur la base de l'étude de cas, le lecteur découvre un site Plone, avec les services vus du point de vue de l'utilisateur et du contributeur.

Les **chapitres 3 et 4** : installation et configuration de Plone. Plone peut être installé sur un serveur Unix, mais aussi sur un poste de travail MS-Windows™.

Le **chapitre 5** : mise en couleur du site Plone, adaptation visuelle et graphique du site.

Le **chapitre 6** : création et adaptation de nouveaux *workflows* pour gérer du contenu.

Le **chapitre 7** : adaptations fonctionnelles de Plone, création de nouveaux types de contenus (atomes d'information du système).

Les **chapitres 8 et 9** : mise en production, montée en charge. Ces chapitres sont plutôt destinés aux administrateurs système.

Le lecteur retrouvera en annexe des compléments d'informations et des références techniques (API).

À qui s'adresse ce livre ?

Ce livre s'adresse à plusieurs profils. Cette approche nous a semblé logique dans la mesure où un projet fait généralement intervenir toutes ces personnes (parfois représentées par le même individu ;)).

	Chapitres								
Profil	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Décideur	✓	✓							
Intégrateur	✓	✓	✓	✓	✓	✓	✓		✓
Développeur	✓	✓	✓	✓		✓	✓		✓
Graphiste	✓	✓			✓				
Administrateur système	✓	✓	✓					✓	✓

Bien entendu, les lecteurs curieux pourront s'essayer à lire les chapitres qui ne leur sont pas directement destinés !

ALLER PLUS LOIN **Zope**

Il est à noter que le présent cahier est un excellent complément au livre Zope, 2e édition. Le présent ouvrage est plus orienté « étude de cas » et se présente donc comme un grand tutoriel, sans vocation d'exhaustivité ; le livre *Zope* est quant à lui bien plus fourni, traitant de plus de sujets et plus en profondeur.

📖 O.Deckmyn, P.-J. Grizel, *Zope 2e édition*, Eyrolles 2003.

Pour plus d'informations sur le livre et pour contacter les auteurs :

- http://www.zopera.org/infos/zopebooks/plone
- livreplone@zopera.org

Remerciements

Nous tenons à remercier chaleureusement les personnes suivantes, sans lesquelles cet ouvrage n'aurait pas trouvé vie : Muriel, et toute la formidable équipe d'Eyrolles (Jean-Marie, Anne et Sophie...), les membres de la communauté Zope et Plone, notamment Alexander Limi, Alan Runyan, Judy, Ludo, Benoît, Rosette (mais avec les yeux), Anne-Marie, Tiyi Anh, Jean-René, Sandrine, Cyrille, Nicolas, Jérémy et tous les autres, qui nous supportent au quotidien.

Table des matières

La gestion de contenu avec Zope et Plone

1

Zope
Plone

CMS | Portail collaboratif | Intranet | Services | Publication

SOMMAIRE

MOTS-CLÉS

- CMS
- Portail collaboratif
- Intranet
- Services
- Publication

La gestion de contenu est l'ensemble des concepts et outils visant à résoudre les problèmes de production de contenu des sites web. Elle cherche à intégrer de manière intelligente les différents acteurs du site et les différents supports de diffusion de l'information.

La gestion de contenu

Pour mieux comprendre le rôle et les avantages de la gestion de contenu, il convient de s'intéresser à l'évolution du Web et plus particulièrement de la création des sites web.

Les premiers sites web ont été édités par des équipes scientifiques animées par le seul besoin de mettre en ligne des informations. Le style était austère, mais l'essentiel était présenté.

Puis, de plus en plus de passionnés se sont rués sur la manne du Web, y apportant leur touche de gaieté personnelle. Ils eurent rapidement besoin d'outils plus « visuels », d'où l'émergence de logiciels d'édition de pages HTML tels que Macromedia Dreamweaver, Microsoft FrontPage, etc.

Les entreprises ont à leur tour jeté leur dévolu sur les fantastiques possibilités du Web et ont confié à des spécialistes le soin de créer, maintenir et mettre à jour leur site. Le métier de « webmestre professionnel » était alors né !

Cependant, avec l'explosion du Web et la professionnalisation de l'activité de création de sites (*startups* et « dotcom », sites dynamiques, sites marchands, etc.), le webmestre est devenu victime de son succès. D'une part, la gestion des pages statiques pose rapidement de nombreux problèmes techniques dès que le site devient conséquent.

La gestion de contenu

La gestion de contenu dynamique et rédactionnel d'un site web doit se faire rapidement et facilement. Il faut pour cela mettre en place un système flexible qui permette une mise à jour facile : la correction, l'ajout de textes, photos ou fonctions multimédias doivent pouvoir être faits par les utilisateurs sans aide extérieure.

Système de gestion de contenu (CMS)

Un système de gestion de contenu (*Content Management System*) est l'ensemble des outils permettant de mettre en œuvre la gestion de contenu. Il se compose en général de nombreux modules fournissant des services : gestion des utilisateurs, création et édition de contenu, indexation et recherche, etc.

Un système de gestion de contenu est généralement composé de modules fournissant des fonctionnalités essentielles sur lesquelles l'utilisateur développe ses applications. Les fonctionnalités que l'on retrouve dans les produits du marché ayant atteint un certain niveau de maturité sont :

- la gestion des utilisateurs et de leurs droits ;
- la création et l'édition de contenu ;
- le stockage du contenu .
- les métadonnées (ensemble de propriétés décrivant le contenu) ;
- la gestion de la qualité de l'information ;
- l'indexation et la recherche ;
- la gestion de l'interface utilisateur ;
- la syndication (regroupement d'informations provenant de différents sites) ;
- la gestion des versions ;
- et tous les services autour de ce tronc commun de la gestion de contenu.

Quelques CMS Open Source

Pour le socle technique de votre projet, choisissez parmi les solutions Open Source les plus populaires du moment :

- Tiki CMS/Groupware - http://tikiwiki.org (solution PHP) ;
- Typo3 - http://typo3.org (solution PHP) ;
- PHPNuke - http://phpnuke.org/ (solution PHP) ;
- Drupal - http://drupal.org (solution Java) ;
- Open ACS – http://openacs.org (solution Java) ;
- Bricolage – http://bricolage.cc (solution HTML::Mason / Perl) ;

et bien sûr :

- Plone 2 – http://plone.org (solution Zope / Python).

Une liste complète des outils de gestion de contenu Open Source est disponible sur le site de l'OSCOM :

- http://www.oscom.org/matrix/

D'autre part, le webmestre doit sans cesse se former aux nouvelles technologies, et il a de plus en plus affaire à des interlocuteurs, utilisateurs, clients ou partenaires financiers, qui ne maîtrisent pas le langage technique – et veulent encore moins en entendre parler. L'entreprise doit donc adopter une démarche rationnelle pour la gestion de son site web.

Face à cet environnement humain et technologique de plus en plus complexe, la nécessité d'organiser la gestion des sites s'imposait. Que ce soit la charte graphique, la création, la maintenance, la mise à jour, l'évolution, tout doit pouvoir être géré, décidé, mis en œuvre à différents niveaux et/ou par différentes personnes.

De là sont nés le concept de gestion de contenu et les systèmes de gestion de contenu.

Domaines d'application

La gestion de contenu permet d'industrialiser la mise en œuvre des sites web ayant des contraintes fortes : forte audience, mise à jour fréquente du contenu, différents groupes d'utilisateurs avec différents droits ou privilèges, temps de téléchargement des pages, contenu multimédia, transactions commerciales, etc.

Parmi les applications typiques d'un système de gestion de contenu, on peut citer le site éditorial, la communauté en ligne, les intranets et les bases de connaissances.

Journaux en ligne, webzines et weblogs

Le site éditorial est le genre le plus répandu sur le Web du fait de sa nature de média partagé. Il permet à un individu ou à un groupe d'individus de se positionner comme source d'information, « infomédiaire », ou veilleur sur des sujets spécifiques. Il se présente sous différentes formes selon le modèle économique, l'objectif visé par le créateur, et la tendance du moment.

Les sites éditoriaux les plus fréquemment rencontrés sont les portails d'information tels que News.com ou ZDNet, les journaux en ligne tels que le lemonde.fr, les « webzines » qui sont plus spécialisés sur un sujet donné, et les « weblogs » (« webillards » en français). Ces derniers, bien que soumis à des contraintes différentes de celles du monde industriel, sont actuellement très populaires dans le monde des sites personnels et du journalisme amateur.

CULTURE **« Blogosphère »**

Les « weblogs » constituent aujourd'hui l'un des domaines les plus créatifs du Web, intégrant régulièrement de nouvelles tendances ou technologies : K-Logs, MoBLogs, Wikilogs, etc. Cette « technosphère » particulière où se rejoignent producteurs de contenu indépendants, journalistes, leaders d'opinion, activistes, entrepreneurs et autres pionniers d'Internet, a été désignée outre-Atlantique sous le terme fédérateur de « Social software ».

Communautés en ligne

La communauté en ligne

Le site est ouvert à des contributeurs sans limitation en nombre, dès lors qu'ils ont du contenu à proposer à la communauté, qu'ils sont inscrits et qu'ils respectent la charte éditoriale du site. Des relecteurs sont chargés de valider le contenu contribué avant qu'il ne devienne public. Les autres membres peuvent également contribuer en apportant des commentaires sur chaque contenu publié. Ainsi, la communauté s'enrichit par la participation de tous.
Parmi les exemples de sites communautaires connus, on peut citer Slashdot.org (/.), kuro5yin.org, ciao.fr, ou encore Zopera.org.

Une communauté en ligne réunit des internautes qui partagent des centres d'intérêt d'ordre général ou professionnel, en leur offrant la possibilité de contribuer à l'information sous forme d'articles, et d'alerter la communauté sur des informations vues ailleurs sur le Web. Un espace de forum permet par exemple aux membres de la communauté de réagir pour donner leur avis sur les contributions ou compléter l'information.

La politique éditoriale choisie par les créateurs du site détermine s'il est « modéré » ou pas. S'il est modéré, un responsable appelé modérateur est averti lors de l'arrivée d'une nouvelle contribution ; après lecture, celui-ci décide de valider ou non la publication. Si le site n'est pas modéré, le contenu est accepté d'office et donc visible dès sa publication par le contributeur. Cette démarche induit tous les risques liés aux habituels débordements humains… Un système non modéré n'est pas pour autant un système anarchique : un responsable doit toujours pouvoir intervenir sur un contenu pour l'ôter de la vue du public si besoin est.

Bases de connaissances

La base de connaissances

Le contenu de la base de connaissances est le capital intellectuel d'une entreprise, d'une organisation ou d'un groupe d'individus. Comme la communauté en ligne, elle requiert l'implication des membres, principaux bénéficiaires de ce capital.

Il s'agit ici d'applications intranet ou Internet permettant de capitaliser l'information et le savoir-faire au sein de l'entreprise ou d'une communauté : idées, documentation, procédures, etc. Cette capitalisation doit se faire de manière structurée et cohérente. De fait, elle requiert des technologies capables de gérer des informations aussi bien structurées que non structurées. D'autre part, ces technologies doivent être au service des employés ou individus qui sont à l'origine de ce capital, en étant flexibles, simples d'utilisation et en aidant à valoriser leur travail.

Présentation de la solution Plone

Statut de Plone

Plone est actuellement disponible en version 2.0 et bénéficie d'une communauté d'utilisateurs et de développeurs très active.
À l'heure où nous mettons sous presse, la version 2.1 est déjà en cours de développement.

Plone est un système de gestion de contenu (en anglais *Content Management System* ou CMS) basé sur Zope et le CMF (*Content Management Framework*), la librairie de composants qui complète Zope en fournissant un ensemble de services aux concepteurs de sites, aux intégrateurs d'applications et aux gestionnaires de contenus.

Le CMF fournit une librairie de composants, chacun spécialisé dans le traitement d'une problématique précise. Chacun des composants se concentre sur son périmètre fonctionnel et collabore avec les autres via des interfaces. Le développeur d'applications ou l'intégrateur peut directement utiliser leurs services via le Web (typiquement, via l'appel d'un script Python ou autre objet exécutable).

Zope, un serveur d'applications avec un modèle de développement objet

Zope est une plate-forme de développement d'applications web basée sur Python. Zope intègre un grand nombre d'outils et de fonctionnalités, dont un gestionnaire de bases de données objet, un module de publication d'objets web et un langage de création dynamique de pages. Contrairement aux autres solutions du marché, la finalité de Zope n'est pas de publier des « pages » mais des « objets » pouvant être assemblés automatiquement à partir de composants dont le comportement, les données et l'apparence sont configurables par le concepteur du site. Cette approche rend Zope beaucoup plus apte que d'autres produits à faciliter la publication de contenu web.

📖 Zope, P.-J. Grizel et O. Deckmyn, Éditions Eyrolles (2003).

ARCHITECTURE **Architecture de composants**

Une évolution majeure de Zope introduite avec le *Content Management Framework* est la notion de « composants ». Cette architecture, qui sera généralisée avec Zope 3, permet de rationaliser le développement en déléguant les fonctionnalités à des composants objets spécialisés.

Le principal avantage est le gain de productivité pour les développeurs. Ils n'ont plus à réinventer la roue, et les différents modules applicatifs collaborent facilement ensemble puisqu'ils se basent sur un référentiel de composants avec des interfaces connues. Le développeur peut également fournir une nouvelle implémentation d'un composant existant. Ainsi, parce qu'il respecte les mêmes interfaces, son nouveau composant peut remplacer le composant défini par défaut au sein du CMF. Par exemple, le composant permettant l'authentification des utilisateurs de l'application aura différentes implémentations selon que la source des utilisateurs est embarquée au sein du serveur Zope (comptes stockés dans le conteneur `acl_users` ou *Standard User Folder*) ou que la source est un annuaire LDAP (via un *LDAP User Folder*).

LDAP

LDAP (*Lightweight Directory Access Protocol*) est un protocole d'échange de données relativement simples stockées dans une base arborescente. C'est un protocole très utilisé pour gérer les annuaires, tels que Active Directory de Microsoft, OpenLDAP ou Novell Directory.

📖 Zope, P.-J. Grizel et O. Deckmyn (Eyrolles 2003) pour une présentation de l'intégration Zope/LDAP.

Les services de Plone

Voici une présentation rapide des services que l'on retrouve dans Plone.

Gestion des membres

Par défaut, le système fournit à chaque membre un espace personnel pour l'organisation et l'édition du contenu auquel il contribue. Ainsi, si vous choisissez un modèle collaboratif décentralisé, le contenu du site n'est pas « stocké » à l'endroit où il sera affiché, mais réparti dans les espaces personnels des membres. Le système met également en œuvre la gestion des profils des membres avec leurs options personnelles.

Ce service est fourni par les composants `portal_membership` et `portal_memberdata`.

Gestion des groupes de membres

Plone 2 permet la gestion des groupes grâce au système d'authentification intégré *GroupUserFolder*. Chaque groupe peut alors disposer de son « espace de groupe » où les membres du groupe ont les droits nécessaires pour créer et gérer leur contenu commun. Le système met également en œuvre la gestion des profils des groupes.

Ce service est fourni par les composants `portal_groups et portal_groupdata`.

Gestion de l'interface utilisateur

Cette gestion est assurée grâce au composant `portal_skins`.

Avec un système d'interface utilisateur reposant sur le nouveau concept des *Skins*, Plone 2 met **réellement** en œuvre la séparation du contenu, de la logique applicative et de la présentation.

En voici une explication très rapide : le contenu est géré via les types de contenus, la logique l'est par l'application et le workflow, et la présentation par les *skins* (CSS2). Ce système permet de déléguer le rendu de tel ou tel objet à une méthode spécialisée pour cette tâche. Il permet de plus de proposer plusieurs présentations pour un même site. On peut aussi utiliser ce principe pour concevoir un site multicible (HTML, WAP, XML, etc.) ou multilingue. Plone 2 propose un système de *skin* qui permet de modifier tout ou partie de l'interface utilisateur sans jamais changer aucun des modèles HTML fournis, mais en agissant au niveau de la feuille de style (CSS2).

Gestion du processus de création et d'édition du contenu

Cette gestion est assurée grâce à plusieurs composants qui collaborent, principalement : `portal_types`, `portal_factory`, `portal_skins` et `portal_form_controller`.

Un type de contenu est une définition faite au sein d'une application Plone pour permettre la gestion du contenu. Le type de contenu s'appuie sur la classe d'objet, mais prend en compte d'autres paramètres. Les types de contenu par défaut (par exemple `Actualité`, `Document` ou `Lien`) peuvent être étendus ou modifiés pour devenir de nouveaux types. Le concepteur définit le comportement du nouveau type et Plone l'expose aux membres du site pour les assister dans leur publication.

Le type de contenu est l'atome de l'information d'un site Plone.

Exemples de types de contenu

- Actualité
- Document
- Article
- Dossier
- Offre d'emploi

Gestion du workflow de publication

Ce service est fourni par le composant `portal_workflow` en collaboration avec le composant `portal_types`.

Pour chaque type de contenu, le concepteur du site peut déterminer tous les états possibles, les transitions et les acteurs. Le niveau de paramétrage du *workflow* est maximal ; le produit d'extension DCWorkflow permet même de « dessiner » une machine à états via le Web.

Workflow

DCWorkflow est un outil de *workflow* « orienté document ». Il décrit bien les différentes étapes d'un document dans le site : ses états, les transitions pour passer d'un état à l'autre.

Il existe cependant un autre type de *workflow* : le *workflow* de processus, qui permet de décrire et d'assister un processus (et non plus un cycle de vie de document) dans le site. Cela peut par exemple concerner le processus d'acheminement du courrier dans une entreprise ou une administration (réception, ouverture, cycles de lecture, cycles de réponses, réponse, etc.). Un tel composant existe déjà pour Plone et se nomme CMFOpenFlow.

▸ http://www.reflab.it/community/Openprojects

Comme tous les composants de Plone, le composant qui gère les *workflows* peut se changer, et l'on peut utiliser à la fois DCWorkflow et CMFOpenFlow sur un même site, en fonction du besoin.

Workflow pas français ?

Non, le mot *workflow* n'est pas français… mais nous ne connaissons pas de (bonne) traduction en français. Certains disent « circuits de validation », mais le terme est en fait inexact et incomplet.

D'après le dictionnaire des expressions informatiques en ligne, le terme se traduit par « gestion électronique de processus » et trouve la définition suivante :

« Outil décisionnel coopératif qui s'attache à optimiser et rationaliser les flux d'information et les procédures de travail. Exemple : on pourra mettre en place un circuit automatique de circulation et de validation des documents sous format électronique, nécessaires à l'activité de l'entreprise. »

▸ http://www.dicofr.com

Gestion de l'indexation et du moteur de recherche

Toutes les instances de contenus créées sont indexées dans un ZCatalog (moteur de recherche objet, intégré à Zope), qui est très fortement sollicité pour le rendu des pages. Le choix des objets rendus sur une page se fait systématiquement via une requête sur le ZCatalog. Plone est une application qui est vraiment centrée sur l'utilisation du ZCatalog de Zope et en fait un usage intensif, lui permettant ainsi d'avoir d'excellentes performances.

Ce service est fourni par le composant `portal_catalog`.

B.A.-BA **Indexation**

L'indexation consiste à conserver dans une base de données des liens vers certains attributs d'un objet (par exemple l'auteur du contenu en question), et de fournir un service qui permet de retrouver très rapidement cette information. Ainsi, si l'on recherche la liste des contributions d'un auteur particulier sur le site, on utilise le moteur d'indexation du site, qui va retrouver l'information en quelques milli-secondes, sans avoir à parcourir toute la base de données pour retrouver cette information.

Gestion des métadonnées

Ce service est fourni par le composant CMF `portal_metadata`.

On peut décrire tous les contenus du site à l'aide d'un ensemble de métadonnées définies par le standard Dublin Core. Les principaux éléments de description actuellement pris en charge sont : le titre, la description, le sujet (liste de mots-clés), le format, la langue, l'auteur, date de publication et de validité, les contributeurs et les droits (copyright), etc.

JARGON **Meta-informations, meta-données ou meta-datas**

Ces trois termes désignent la même notion : un jeu de données décrivant le contenu auquel elles sont associées. Un jeu de méta-informations permet donc de décrire l'information en question (le contenu). Ainsi, on retrouve habituellement les méta-informations suivantes :

- auteur
- date de publication
- date de validité
- copyright
- etc.

Ces informations sont fondamentales dans un système de gestion de contenu, car elles participent énormément à la qualité de l'information contenue dans la base, et donc à sa réutilisabilité.

CULTURE **RDF**

RDF (*Resource Description Framework*) est une technologie développée par le W3C (*World Wide Web Consortium*) pour permettre la description de ressources sur le Web (sites, applications, contenus). RDF s'appuie sur la technologie XML (*eXtensible Markup Language*) pour la syntaxe d'échange des données.

CULTURE **Dublin Core (DC) Metadata**

Le Dublin Core Metadata Initiative est une organisation qui travaille à définir des standards d'utilisation des métadonnées pour les applications. Le *Dublin Core Metadata Element Set* définit les attributs de description des données en s'appuyant sur la technologie RDF.

▸ http://dublincore.org/

Autre standard de métadonnées

- *Learning Objects Metadata* (LOM)

Gestion des versions

Ce service requiert l'intégration d'une nouvelle brique logicielle du CMF qui n'est pas encore disponible dans la distribution standard : `CMFStaging`. Le lecteur curieux pourra chercher dans le référentiel de code source de Zope via un accès CVS (*Concurrent Versionning System*) en lecture :

▸ http://cvs.zope.org/CMF/CMFStaging/

La gestion des versions permet de manipuler différentes versions d'un contenu, ou d'un ensemble de contenus (par exemple, toute une partie du site). Typiquement, chaque contributeur peut revenir à une version précédente de son contenu, et le webmestre peut rendre publique une nouvelle version du site après l'avoir testée, tout en ayant la possibilité de rétablir la version précédente si cela s'avère nécessaire.

Ainsi, en cas d'erreur faite à un moment de la vie du contenu, il est possible « d'annuler » les modifications les plus récentes pour revenir à une version plus ancienne.

Gestion de la syndication

Ce service est fourni par le composant CMF `portal_syndication`.

Le CMF (et donc Plone) vous permet d'exposer n'importe quel contenu de votre site sous forme de RSS XML. Ainsi, d'autres sites peuvent s'abonner aux informations de votre site pour les diffuser.

B.A.-BA **Syndication**

La syndication permet de partager de l'information entre différents sites web. À partir d'un site fournissant cette application, l'internaute peut consulter des contenus (généralement de type *news*) provenant de différents sites sans être obligé d'aller sur les sites en question.

RSS (*RDF Site Summary*) est le format de fichier de description de ressources le plus utilisé pour la syndication.Il permet de décrire un ensemble de contenus, offrant pour chacun un lien vers le texte complet.

Avantages de Plone pour les gestionnaires de contenu

La mise en ligne du contenu devient facile

D'une part, la publication du contenu ne nécessite qu'un simple navigateur web. Ainsi, le producteur de contenu peut publier ses informations sur le site de partout et à tout moment. Ceci est particulièrement indispensable pour des entreprises avec des équipes éclatées, qui travaillent à distance ou avec des horaires décalés.

D'autre part, l'outil de publication permet aux rédacteurs d'injecter des documents bureautiques, en prenant en charge les conversions de format nécessaires. La flexibilité de la publication vient du fait que le contenu mis en ligne est stocké dans une base de données. En effet, il est ainsi accessible et modifiable par les rédacteurs identifiés, et manipulable par tous les moyens de traitement informatique.

La publication du contenu peut être organisée

La complexité de la production de contenu non seulement crée des goulots d'étranglement qui handicapent l'entreprise, mais également décourage l'implication des employés, clients et partenaires, fournisseurs potentiels de contenu à valeur ajoutée. Grâce à la gestion de contenu, tout collaborateur de l'entreprise, détenteur d'information, peut, à l'intérieur de son périmètre de responsabilité, produire son contenu sur le site, sans empiéter sur le travail de son collègue, pair ou partenaire. Tous les maillons de l'organisation participent ainsi à la vie de « l'entreprise virtuelle » avec un certain degré d'autonomie. Il en découle une valorisation du travail des collaborateurs et, par conséquent, des gains en productivité et en opportunités commerciales pour l'entreprise.

La gestion de contenu permet d'assurer la qualité de l'information

La gestion de contenu permet en outre aux informations mises en ligne de suivre un circuit de validation qui réduit les risques d'erreur de publication. On peut ainsi rejeter un contenu afin qu'il soit corrigé puis soumis à nouveau

à validation. D'autre part, il est toujours possible de commenter un contenu afin d'y ajouter des informations complémentaires ou d'élargir son contexte.

D'autres fonctions importantes liées à la qualité de l'information sont la normalisation des gabarits de pages, le suivi de la validité des documents dans le temps, la possibilité de retour en arrière, et l'archivage automatique. Ces fonctions sont autant de garanties d'une meilleure expérience utilisateur, facteur bénéfique pour l'audience du site.

Le gestionnaire de contenu dispose d'une interface utilisateur adaptée à son besoin

Un aspect important des sites web modernes est la manière dont les fonctionnalités de présentation et de navigation sont mises en œuvre. À partir du moment où le site s'adresse à un public large et diversifié, souvent international, la mise en œuvre de la charte graphique et la gestion de l'interface utilisateur ne peuvent plus être prises à la légère. Un nouveau métier, celui de « spécialiste de l'ergonomie web », s'est même développé.

Un investissement pour l'avenir

La plate-forme technique de Plone est saine, avec une séparation « Présentation/Structure/Logique/Données » qui permet de changer le comportement visuel du site simplement avec les paramètres CSS (voir chapitre 5), et avec un outil intégré de migration qui gère les changements techniques nécessaires lors d'une mise à jour.

Avantages de Plone pour les développeurs

Un socle technique sur lequel bâtir des applications « métier »

Le développeur peut jouer sur plusieurs niveaux d'intégration de Zope, qui lui donnent une flexibilité importante. Il peut :

- intégrer ou étendre des types de contenu déjà disponibles,
- développer ses propres types de contenu. C'est généralement nécessaire dès que l'on doit fournir des « objets métier »,
- bâtir de la logique applicative au sein de la *skin* pour apporter des fonctionnalités supplémentaires au site,
- développer son propre composant ou *Tool*,
- intégrer des données externes (via des connecteurs aux SGBDR, des scripts de connexion à des serveurs externes, etc.).

Principaux apports de Plone 2

- Intégration complète des composants du CMF et des produits additionnels tels que GroupUserFolder avec pour objectif une infrastructure plus robuste et extensible.
- Couche de présentation et d'intégration graphique offrant des sites fortement ergonomiques et conformes aux standards d'accessibilité (W3C AAA, Section 508).
- Plone Control Panel : interface d'administration extensible grâce à la technologie des « configlets ».
- Nouveaux types de contenu basés sur la technologie Archetypes.
- Riche bibliothèque de composants maintenus dans le cadre du projet Sourceforge *Collective* (http://collective.sourceforge.net).
- Meilleurs outils I18N.
- Amélioration de la recherche grâce à l'utilisation d'index plus performants (ZCTextIndex, TextIndexNG, DateIndex, DateRangeIndex, TopicIndex, etc).

Un système d'authentification intégrant la gestion des groupes

Le composant GroupUserFolder, développé par Ingeniweb, a été intégré à Plone 2 pour permettre l'authentification à partir d'une source d'utilisateurs gérée dans Zope ou sur un serveur externe (LDAP, Active Directory, ou base SQL), et la gestion de groupes d'utilisateurs. Cette gestion des groupes est essentielle pour la mise en œuvre d'un portail collaboratif puisqu'elle facilite la distribution des droits et permet la gestion d'espaces collaboratifs.

Une librairie de composants « plug-ins »

La communauté CMF/Plone, via son projet *Collective* sur Sourceforge.net, fournit un nombre important et sans-cesse croissant de composants pour répondre à différents besoins. Le développeur n'a pas besoin de réinventer la roue. Il a toutes les chances de trouver au sein du *Collective* un composant qui réponde à son besoin (au moins en partie). Il peut rapidement le tester, l'intégrer et/ou contribuer à le faire évoluer vers une API assez robuste pour que toute la communauté puisse se reposer sur ce composant à l'avenir. C'est le cas des composants/outils tels que GroupUserFolder et Epoz, qui sont aujourd'hui des composants fournis en standard dans Plone 2.

Des composants paramétrables

Les composants sont également assez génériques pour couvrir plusieurs cas d'utilisation ou configurations de déploiement. Pour atteindre ce niveau flexibilité sans sacrifier la qualité logicielle, les composants proposent pour la plupart des options de paramétrage disponibles sur les objets qu'ils gèrent ou sur une fiche de propriétés (objet de type `PropertyManager`) au sein du composant `portal_properties`. Certaines options s'appuient sur le langage TALES pour permettre un paramétrage dynamique (exemple des actions de `portal_actions`).

CULTURE Plone, c'est aussi un projet bien organisé

Des équipes :

- Développement
- Gestion des versions publiées et des « installeurs » (outils d'installation)
- i18n (traductions)
- Documentation

Des processus et des pratiques :

- *Plone Improvement Proposal* (PLIP)
- Obligation d'écrire des tests unitaires
- Sessions de *sprint*

Des outils :

- Collecteur de bogues
- Site web
- Référentiel de code basé sur CVS (Sourceforge)
- ZopeTestCase et CMFTestCase

Et aussi…

La solution Plone intègre progressivement les autres outils qui font la force de Zope, tels que :

- le moteur d'indexation, le ZCatalog, et les nouveaux types d'index introduits par les développeurs de la communauté (TopicIndex, TextIndexNG, FieldedTextIndex, etc.) ;
- les solutions de gestion de la mémoire cache (Cache Managers) ;
- l'outil de *clustering* applicatif, Zope Enterprise Objects (ZEO), intégré en standard depuis Zope 2.7 ;
- la technologie *Adaptable Persistent Engine* (APE), qui complète la base de données objet de Zope pour permettre le *mapping* relationnel-objet.

En résumé…

Plone s'avère la solution idéale dans le monde Zope pour des projets de portail collaboratif ou d'intranet/extranet. Cette technologie amène avec elle la puissance de Python et Zope, et est complètement adaptée aux approches de développement pragmatique. Les responsables d'un projet peuvent rapidement mettre en œuvre le socle technique du site, puis y intégrer la charte graphique, le contenu, et les services complémentaires développés sur mesure, ceci de manière incrémentale.

Nous allons vous montrer dans les chapitres qui suivent comment élaborer et mettre en œuvre votre projet de portail ou de système de gestion de contenu avec Plone.

Présentation de l'étude de cas

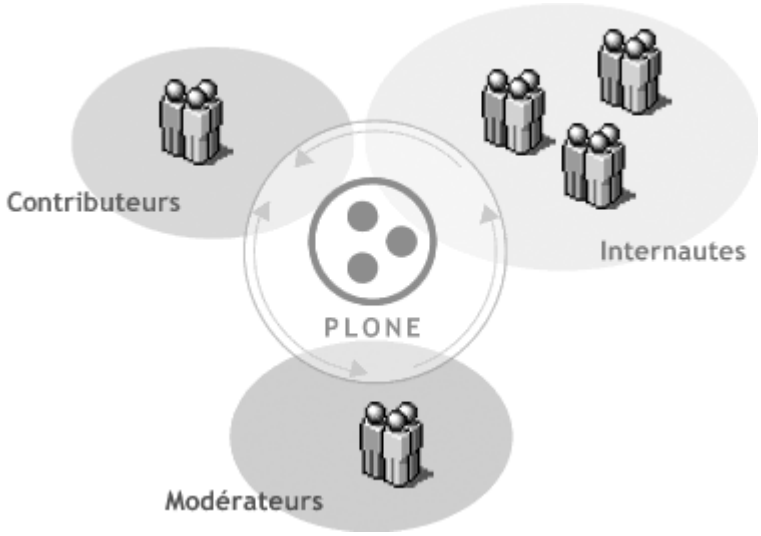
2

Zope
Plone

SOMMAIRE

MOTS-CLÉS

- Intranet
- Cas d'utilisation
- Utilisateur final
- Méthodologie



Nous allons rentrer dans le vif du sujet en abordant le site qui va servir de support à l'étude de cas. Nous verrons que cette étude est articulée autour de la réalisation d'un site intranet d'entreprise, mettant en œuvre les fonctionnalités de gestion de contenu de Plone.

B.A.-BA **Intranet/extranet**

Un intranet est un site web dont l'accès est réservé aux actifs d'une collectivité (entreprise, administration, communauté, etc.). En termes de réseau, un intranet est généralement visible uniquement depuis l'intérieur de l'établissement.
Un extranet est un intranet auquel on accède depuis le réseau Internet, après s'être identifié par un identifiant et un mot de passe. Il n'est pas rare qu'un extranet soit également ouvert aux partenaires, clients ou fournisseurs de l'entreprise, selon les cas ou les besoins.

Un projet d'intranet d'entreprise

Cet ouvrage est conçu autour de l'analyse d'un cas précis : l'intranet de l'entreprise fictive « Société Géniale ». La « Société Géniale » est une PME spécialisée dans la conception, la production, la commercialisation et le service après-vente de farces et attrapes. Cet intranet est notamment disponible pour les différents services qui composent la société :

- Administration,
- Marketing,
- Commercial,
- Production,
- Service Après-Vente (SAV),
- Direction Générale.

Pour simplifier l'étude, nous limiterons notre analyse à trois populations :

- Marketing,
- Service Après-Vente,
- Direction Générale.

Les besoins

Gestion des utilisateurs

QUALITÉ **Les services d'un intranet moderne**

- Gestion des utilisateurs
- Sécurité basée sur les rôles
- Base documentaire (Diffusion de documents, d'informations et d'actualités de la société à l'ensemble des salariés)
- Rubriques pour l'accès aisé au contenu
- Accès restreint à certains contenus en fonction du profil
- *Workflow* : circuits de validation de publication
- Agenda partagé
- Moteur de recherche
- Intégration de fichiers bureautiques (Word, Excel, PDF, etc.)
- Espace de contenu personnel pour chaque salarié
- Espaces de travail collaboratif

Les utilisateurs s'authentifient auprès du site intranet grâce à leur identifiant et leur mot de passe. Ils sont réunis en groupes qui peuvent être calqués sur l'organisation de l'entreprise, chaque personne étant rattachée à un ou plusieurs service(s) donné(s).

Les comptes utilisateurs peuvent être créés par l'administrateur du site ou fournis par l'intégration d'une source externe de comptes utilisateurs comme un annuaire LDAP.

Sécurité basée sur les rôles

Les rôles regroupent des permissions techniques qui autorisent l'accès au contenu de l'intranet ainsi que des actions au sein de l'intranet. Les exemples de permissions nécessaires aux acteurs ou membres du site sont :

- ajouter du contenu ;
- modifier le contenu ;
- ajouter des dossiers ;
- modifier les métadonnées des éléments.

Au départ, les membres de l'équipe qui s'occupe du paramétrage et de la maintenance du site possèdent le rôle d'administrateur. Ils ont ensuite à

s'assurer que les rôles correspondant aux différents profils d'utilisateurs sont paramétrés et affectés aux utilisateurs.

Rôles, utilisateurs, groupes

Quelle différence faire entre rôle, utilisateur et groupe ? La réponse est parfois subtile, essayons d'apporter quelques éclaircissements.

Un utilisateur est la représentation informatique d'une personne physique. On utilise généralement un dérivé de son nom pour construire son identifiant (ex : `jdupont` pour Jean Dupont).

Un rôle est généralement la représentation informatique d'une fonction dans l'entreprise. Par exemple, le rôle « Photographe » sur un site éditorial identifie la fonction de photographe (si, si, promis). Les utilisateurs sont porteurs de zéro, un ou plusieurs rôle(s). Ainsi, notre utilisateur `jdupont`, porteur du rôle « Photographe » aura la possibilité d'ajouter des photographies sur le site.

Un groupe est la représentation informatique d'un groupement d'utilisateurs (ou de groupes). Ces utilisateurs sont réunis dans un groupe parce qu'ils partagent un point commun, par exemple l'appartenance à un même service de l'entreprise. Ainsi, le groupe « Marketing » va regrouper tous les membres du service éponyme. Un utilisateur peut appartenir à zéro, un ou plusieurs groupe(s). Ainsi, `jdupont` peut par exemple être membre des groupes « Marketing », « Direction » et « Équipe de Foot N°1 ». Utiliser les groupes est donc une façon pratique de distribuer des droits sur le site ou simplement de classer les utilisateurs.

Base documentaire

L'objectif principal du site est de permettre la diffusion d'informations aux salariés de la société. La production de cette information n'est pas réservée à une élite informatique, mais ouverte à tous, avec, pour certaines informations, un circuit de validation avant publication.

Les types d'informations à gérer sont :

- actualités ;
- articles ;
- événements ;
- liens.

Rubriques d'accès au contenu

Le site doit fournir des rubriques qui permettent un accès rapide à l'information. Une rubrique est un espace où sont agrégées toutes les informations publiées répondant à un critère donné. On retrouve généralement des rubriques telles que « Actualités », « Produits » et « Liens ».

PLONE **Les rubriques**

Dans Plone, une rubrique est un objet nommé `Topic` qui agrège les résultats d'une recherche sur le ZCatalog à partir d'un critère que le concepteur du site définit. Ce type de rubrique est une solution efficace pour organiser l'accès à l'information dans un site à haut volume. Du fait de l'utilisation du ZCatalog, l'affichage de la page d'une rubrique est rapide. D'autre part, un même contenu peut apparaître dans des rubriques différentes.

Circuits de validation de publication (workflow)

Tous les salariés peuvent, une fois authentifiés grâce à leur identifiant et leur mot de passe, proposer du contenu dans le site, dans les rubriques pour lesquelles ils auront été habilités à le faire. Un circuit de validation permet aux responsables de rubriques de valider ou de refuser la publication d'un contenu proposé.

Accès restreint à certains contenus en fonction du profil

Certaines informations doivent être à accès restreint. Dans notre cas, les dossiers contenant les informations stratégiques ou sensibles de la Direction Générale seront privés et leur accès ne sera autorisé qu'aux utilisateurs membres de ce service.

PLONE Les rubriques à accès restreint

Pour parvenir à restreindre l'accès à une rubrique, l'utilisation d'un état de *workflow* (par exemple `publishedRestricted`) avec une politique de sécurité locale est une excellente stratégie avec Plone.

Agenda partagé

Les événements de la société sont consignés dans un agenda partagé par tous. La publication d'un événement dans l'agenda est également soumise à un circuit de validation.

Moteur de recherche

Le moteur de recherche permet de retrouver les informations disponibles sur l'intranet, en ne présentant à l'utilisateur que les contenus qu'il est habilité à voir. Les résultats de la recherche sont affichés sur une page dynamique dont l'ergonomie est très travaillée. Ces résultats doivent pouvoir être facilement parcourus et reclassés.

Ce même moteur de recherche sera également utilisé pour afficher dynamiquement (dans une boîte latérale, ou « portlet ») les actualités récentes du site et les informations relatives à la rubrique ou au document que l'internaute est en train de consulter.

Portlets

Le terme *portlet*, ou « boîte latérale », désigne les conteneurs d'informations que l'utilisateur (ou l'administrateur) peut disposer dans les colonnes gauche et droite du site. Ce terme nous vient tout droit du monde Java. Alors que les termes *applet* (« petite application ») et *servlet* (« petit serveur ») étaient très utilisés pour désigner respectivement une mini-application embarquée dans un navigateur web et un mini-serveur java responsable d'une tâche atomique, le terme *portlet* (« petit portail ») s'est vite imposé pour désigner ces boîtes qui composent les portails…

Intégration des fichiers bureautiques (Word, Excel, PDF, etc.)

Il est de notoriété publique que 90 % de l'information d'une entreprise est stockée dans des fichiers bureautiques, utilisant généralement la suite bureautique Microsoft Office™ (et de plus en plus souvent Open Office, son concurrent issu du logiciel libre). Les utilisateurs devront donc pouvoir ajouter facilement ces contenus dans le site. Le contenu de ces fichiers devra être accessible par le moteur de recherche. Si un utilisateur sollicite le moteur de recherche avec un mot contenu dans un fichier PDF publié sur le site, ce fichier sera retrouvé immédiatement, quel que soit l'endroit où ce mot est écrit dans le fichier PDF.

Espace personnel pour chaque salarié

Chaque utilisateur dispose d'un espace personnel dans lequel il peut mettre en place ses propres fichiers, ou bien travailler ses contributions avant demande de publication.

Espaces de travail collaboratif

Chaque groupe de travail dispose d'un espace commun à tous ses membres. Ceci permet à des utilisateurs de travailler ensemble à l'élaboration d'un document, avant de décider d'en publier une version définitive.

Présentation du site

Nous avons assez bavardé ! Parcourons ensemble ce bel intranet.

Page d'accueil

Pour se connecter au site, l'utilisateur pointe son navigateur sur : http://intranet.

On distingue cinq grandes parties sur cette première page qui expose toutes les fonctionnalités et interactions importantes aux utilisateurs arrivant sur le site. Nous les détaillons ci-dessous.

La barre de haut de page

On y retrouve principalement le logo de l'entreprise, des onglets représentant les liens vers les rubriques importantes du site, et les liens d'actions personnelles (spécifiques au profil de l'utilisateur).

Figure 2–1 Haut de page

La barre de gauche

On y retrouve par exemple l'arbre de navigation et la boîte permettant de s'identifier. En particulier, la boîte de navigation permet de savoir où l'on se trouve dans le site et de naviguer rapidement en cliquant directement sur les sections voulues.

Figure 2–2 Barre de gauche

Figure 2–3 Barre de droite

La partie centrale

C'est la partie qui diffuse le contenu à proprement parler de la page d'accueil. Elle combine du contenu statique, fourni par un document que le webmestre peut facilement modifier, et du contenu dynamique pour plus d'efficacité. La zone de contenu dynamique agrège les dix documents et articles récemment publiés au sein du site en les classant par ordre chronologique inverse.

La barre de droite

On y retrouve des éléments apportant des services supplémentaires aux membres de l'intranet, typiquement la boîte de recherche utilisant des moteurs de recherche Internet (Google, Pages Jaunes, Societes.com, etc.), la boîte des dernières actualités, et l'agenda donnant accès aux événements publiés sur le site.

La barre de bas de page

Elle est réservée à des informations de *copyright* et à d'autres interactions importantes que l'on veut exposer à l'utilisateur (lien de retour en début de page, accès au plan du site, etc.).

Figure 2–4 Bas de page

Principales rubriques et espaces collaboratifs

Rubriques

Les rubriques constituent le principal mode d'accès à l'information au sein du site.

La rubrique *Actualités* regroupe les articles d'actualité et de revue de presse publiés par l'équipe Marketing.

Documentation regroupe les documents sur les processus et procédés techniques relatifs à l'activité de l'entreprise.

On trouve sur *Produits* des articles publiés par l'équipe Marketing, diffusant notamment les fiches « produits » de la société.

Événements regroupe les annonces d'événements relatifs à l'activité de l'entreprise (conférences, ateliers, journées d'action commerciale, etc.)

Partenaires regroupe des liens annotés vers les sites partenaires ou affiliés, ou les sites de référence.

Espaces collaboratifs

En dehors des rubriques d'information, des espaces collaboratifs peuvent être créés pour permettre à un groupe de collaborateurs de l'entreprise de travailler ensemble sur une étude ou un projet donné, à caractère confidentiel (ou non).

Ces espaces sont à accès restreint au groupe. Ils peuvent être permanents ou temporaires, l'administrateur étant chargé de les paramétrer selon le besoin, et éventuellement d'archiver leur contenu lorsque cela est nécessaire.

Workflows de publication

Le *workflow* de publication mis en œuvre par le système de gestion de contenu définit les états par lesquels le contenu transite, selon la politique éditoriale et/ou de collaboration du site, depuis sa création jusqu'à sa destruction.

Le *workflow* de publication de documents (et d'articles) est principalement composé des états suivants :

- *Privé* – C'est l'état initial d'un document en cours de rédaction.
- *En attente de modération* – C'est l'état intermédiaire par lequel doit passer le contenu devant être publié. Le contenu attend dans cet état d'être validé par un modérateur (concrètement, d'être « publié »).
- *Publié* – C'est l'état dans lequel le contenu est publiquement accessible par toutes les personnes de l'entreprise accédant à l'intranet.

Il existe un *workflow* simplifié, pour les dossiers (utilisés comme conteneurs des documents). Le *workflow* des dossiers est composé uniquement des états *privé* et *publié*, l'état *privé* étant l'état initial.

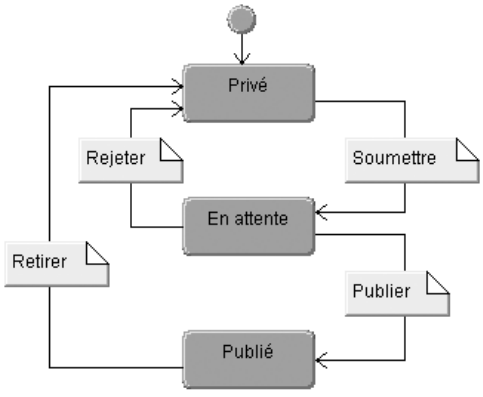


Figure 2–5 Schéma d'un exemple de workflow pour les documents

Contribution de contenu

Le contenu d'un dossier (ou d'un espace) peut être rédigé et maintenu par les membres du site ayant des droits de contribution dans ce dossier. Ces droits sont donc attribués localement via l'onglet `Partage` accessible dans le contexte de chaque dossier (pour les utilisateurs ayant les bons droits). C'est une solution efficace pour la gestion déléguée du contenu.

Pour contribuer, vous devez d'abord accéder à un dossier où vous avez l'autorisation de proposer du contenu. Dans un tel dossier, vous voyez apparaître les onglets de gestion avancée tels que `Contenu`, `Partage`, ainsi qu'un lien permettant d'ajouter votre élément de contenu, et un autre permettant de changer l'état du dossier si besoin.

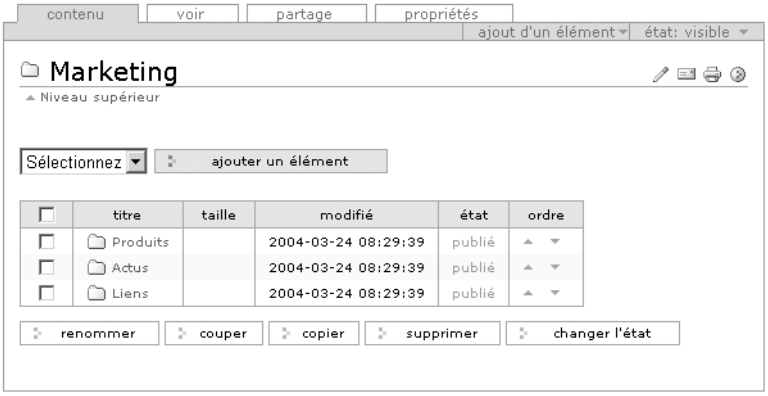


Figure 2–6 Vue du contenu d'un dossier

B.A.-BA Délégation de droits

La délégation de droits permet à un utilisateur, comme dans la vraie vie (enfin, souvent), de déléguer (prêter) ses droits à un autre utilisateur. Ainsi, en cas d'absence de l'indispensable Directeur Marketing, son jeune bras droit pourra le remplacer pendant cette période. Plutôt que de donner son identifiant et son mot de passe à cet utilisateur, le directeur va simplement lui déléguer les droits de modération du contenu qu'il aura dans telle ou telle rubrique.
Il est intéressant de noter qu'un utilisateur ne peut donner (ou déléguer) plus de droits qu'il n'en a lui-même.

OUTILS Gérer un projet Plone

Collecte des spécifications : Wiki
Tests unitaires : ZopeTestCase et CMFTestCase
Modélisation : UML
Gestion d'un référentiel de code source : CVS

Points de méthodologie

Identification des acteurs

Un acteur représente un rôle joué par une entité externe (utilisateur, autre système informatique) qui interagit directement avec le système à réaliser. Dans notre cas de figure et pour des raisons de simplification, il n'y a que des acteurs humains.

Les acteurs du site intranet sont les suivants :

- Le **membre** : toute personne enregistrée dans la base d'utilisateurs du site, et qui peut consulter tout le contenu publié. De plus, le membre dispose d'un dossier personnel où il peut produire et éventuellement publier du contenu personnel. Pour faire simple, on pourra dire que les membres sont les salariés de la société.

QUALITÉ **Pratiquer l'« eXtreme Programming »**

Cette méthodologie de développement (terme inexact, mais pratique pour situer le débat), préconise de nouvelles dispositions de travail sur un projet de développement logiciel. Comme toute méthode de développement, l'eXtreme Programming (XP) propose un cadre pour l'ensemble des aspects du projet logiciel, depuis l'analyse des besoins jusqu'aux tests, en passant par la conception. Mais à la différence de processus prédictifs, recourant généralement à UML, XP ne se fonde pas sur la définition exhaustive et précoce des besoins ; elle parie plutôt, à partir d'un ensemble de règles strictes, sur la souplesse et la mise en valeur du « capital humain ».

La programmation objet et la gestion de projet comme disciplines collectives : tout en mettant l'accent sur les bonnes pratiques de programmation, XP préconise un déroulement par itérations courtes et géré collectivement. Il en découle une redéfinition de la relation entre clients et fournisseurs, avec de surprenants résultats en termes de qualité de code et de délais.

Plus d'informations :

📖 *L'Extreme Programming, avec deux études de cas*, de Jean-Louis Bénard, Laurent Bossavit, Régis Médina, Dominic Williams aux Editions Eyrolles.

- Le **contributeur** : personne qui fait partie de l'équipe rédactionnelle de l'intranet ; il est autorisé à ajouter et modifier du contenu dans une (ou plusieurs parties) du site à caractère public (hors de son dossier personnel).
- Le **modérateur** : personne qui relit et valide la publication de contenus qui lui sont soumis, ou les retourne aux auteurs pour correction avant d'accepter de les publier.
- Le **webmestre** : personne qui a en charge la maintenance du site, notamment la création des dossiers qui structurent le site et la gestion des comptes utilisateurs et de leurs habilitations.

Identification des cas d'utilisation

Un cas d'utilisation (ou *use case*) représente un ensemble de séquences d'actions qui sont réalisées par le système informatique (ici le système de gestion de contenu du site) et qui produisent un résultat observable et utile pour un acteur donné.

À titre d'exemple, nous présentons ici les deux principaux cas d'utilisation de la contribution d'articles au sein du site.

Création et édition d'un article

Acteur(s) : Contributeur

Préconditions :

- Le contributeur est authentifié par le système.
- Le contributeur se trouve dans un dossier ou espace où il a les droits requis.

Scénario :

- Via un bouton ou lien d'action, le contributeur accède au formulaire de création de l'article.
- Le contributeur complète les champs du formulaire avec le contenu textuel de son article et valide.
- L'article est créé dans son état initial, l'état `privé`.
- Le contributeur peut continuer à le modifier. En passant par les onglets d'extension de l'article prévus par le système, il peut lui associer des mots-clés, des photos, et/ou des fichiers bureautiques en pièce jointe.

Publication d'un article

Acteur(s) : Contributeur, Modérateur

Préconditions :

- Les acteurs sont authentifiés par le système.
- L'article se trouve dans l'état « privé ».

Scénario :

- Via un bouton ou lien d'action disponible dans le contexte de l'article, le contributeur soumet celui-ci pour la publication.
- Lorsqu'il se connecte, le modérateur concerné par l'article, voit apparaître ce dernier dans la liste des éléments en attente de modération. Il consulte l'article et, après relecture, s'il est d'accord pour le publier, il actionne le bouton (ou le lien) permettant de passer l'article à l'état « publié ».

B.A.-BA Métadonnées

Le titre et la description sont essentiels pour la découverte (via le moteur de recherche par exemple) et la consultation de votre article. Ce sont les premières métadonnées de votre contenu. On y ajoutera par exemple l'auteur ou la date de publication du contenu.

Alternative

Si le modérateur ne souhaite pas valider l'article, il choisit l'action de le rejeter ; l'article revient dans l'état « privé » et le contributeur peut le remanier.

Stratégie de stockage du contenu

Nous devons trouver la meilleure solution de stockage des articles et des documents. La base de données objet de Zope fournit une solution performante pour stocker chaque article et y accéder soit directement via son chemin, soit, plus efficacement, via le moteur d'indexation du site. Les informations peuvent ensuite être agrégées sur les pages des rubriques. Même pour des gros volumes (plusieurs giga-octets), la base objet de Zope est parfaitement adaptée, ses performances restant excellentes.

Il est conseillé d'utiliser une arborescence de stockage calquée sur l'organisation des gestionnaires de contenu, de manière à permettre la gestion déléguée du contenu.

Typiquement, dans notre cas, on aurait l'organisation suivante :

```
[Contenu]          ←Dossier racine de la base documentaire
        Marketing
                actus
                produits
                liens
        SAV
                actus
                procédures
                solutions
                liens
        Direction
                procédures
                mémos
                gestion
```

En résumé…

Maintenant que nous avons fait le tour des besoins d'un projet d'intranet « type », lançons-nous dans la mise en œuvre. Après l'installation du serveur de développement, nous allons voir, au fil des chapitres suivants, comment Plone apporte une solution à la fois simple et élégante aux besoins fonctionnels de ce type de projet.

Installation de Zope et de Plone

3

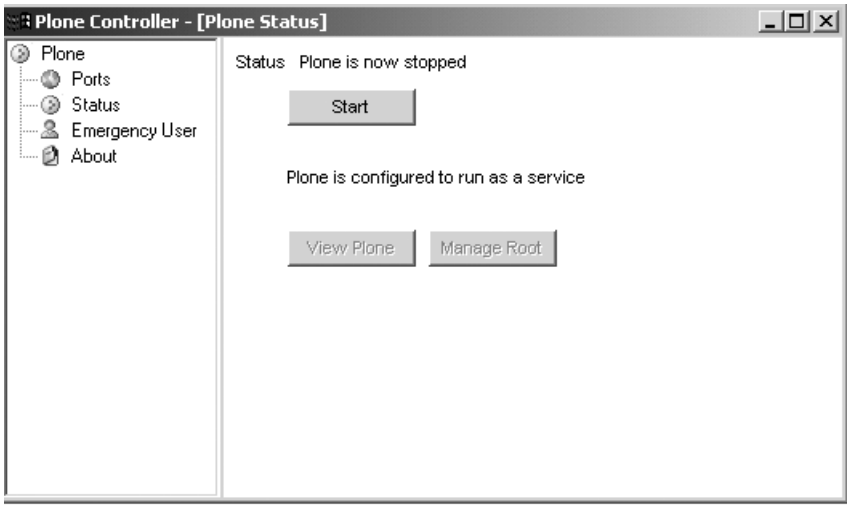
Zope
Plone

Plate-forme | Instance Zope | Modules | Composants | Produits | Services |

SOMMAIRE

MOTS-CLÉS

- Plate-forme
- Instance Zope
- Modules
- Composants
- Produits
- Services



Dans ce chapitre, nous allons installer les différents composants logiciels nécessaires à l'infrastructure et à la mise en œuvre du site intranet, non sans rappeler quelle est l'infrastructure de Plone.

Composantes d'une plate-forme Plone

Rappelons que Plone est une plate-forme de gestion de contenu qui s'appuie sur le serveur d'application Zope. Le cœur de Zope est un système de publication d'objets nommé ZPublisher. D'autres services de « bas niveau » sont offerts par le serveur : des objets de contenu simples (`Folder`, `Image`, `File`, etc.), des objets qui exécutent du code (`DTML Method`, `Page Template`, `Script Python`), le système de sécurité, le moteur d'indexation, etc. Plone n'est en fait que la « surcouche » qui facilite la mise en œuvre d'applications de gestion de contenu. Nous présentons ci-dessous les composantes de cette surcouche logicielle.

RAPPEL **Publicateur d'objets de Zope**

Chaque fois que vous invoquez une URL, Zope vous sert un objet grâce à un module nommé `ZPublisher`, qui joue le rôle d'ORB (*Object Request Broker*).

L'infrastructure de Plone

L'infrastructure de Plone est fournie par les produits CMF auxquels contribuent Zope Corporation et les principaux acteurs de la communauté CMF/Plone. Découvrons-les ci-dessous.

Le produit CMFPlone

C'est le produit central qui apporte les fonctionnalités du système de gestion de contenu Plone. Tous les autres produits sont soit des dépendances, soit des options complémentaires.

La base de l'infrastructure

CMFCore

C'est le produit qui fournit le cœur de l'architecture de CMF et ses classes d'objets techniques.

CMFDefault

C'est le produit qui fournit la première implémentation par défaut d'un portail basé sur le CMF. Plone s'inspire des choix techniques de CMFDefault et utilise certains de ses composants via l'héritage.

Archive des produits de la couche CMF

Le CMF est disponible (indépendamment de Plone) comme un ensemble de produits qui fournissent la base de l'infrastructure : CMFCore, CMFDefault, CMFCalendar, CMFTopic, et DCWorkflow.

▸ http://www.zope.org/Products

OUTILS **Archetypes**

Ce paquetage est présenté plus loin dans ce chapitre. Concernant son utilisation par le développeur, reportez-vous au chapitre 6.
La référence des champs et des widgets Archetypes est donnée dans l'Annexe 3 de cet ouvrage.

DÉPENDANCE **CMFPlone et CMFDefault**

Il est prévu qu'une future version de Plone ne dépende plus de CMFDefault. Typiquement, le produit tierce partie CMFPloneTypes, actuellement en version *beta*, devrait permettre de remplacer les types de contenu de CMFDefault par de nouveaux types (`Plone Document`, `Plone News Item`, etc.) construits sur une base CMFPlone/Archetypes. Des outils seraient alors fournis pour permettre aux utilisateurs de migrer leur contenu existant.

CMFTopic

C'est le produit qui fournit le type de contenu `Topic` permettant, dans les sites à haut volume de contenu, de créer des vues transversales (ou « filtres » ou « thèmes ») basées sur le moteur de recherche. Il s'agit d'un puissant composant CMF.

SYNTAXE Topic (ou Rubrique en français)

Les `Topics` sont typiquement utilisés pour construire des rubriques qui agrègent des articles ou documents correspondant à un critère donné.

CMFCalendar

C'est le produit qui fournit le type de contenu `Event` permettant de créer des fiches « Événement ».

SYNTAXE Event (ou Événement en français)

Le type `Event` permet de mettre en œuvre une rubrique Agenda simple, sans aucun développement supplémentaire.

DCWorkflow

Développé par Shane Hathaway de Zope Corp., DCWorkflow est le produit Zope qui fournit le moteur de *workflow* des applications CMF. Pour en savoir plus sur le paramétrage des *workflows* avec DCWorkflow, reportez-vous au chapitre 6 « Personnalisation fonctionnelle ».

RAPPEL Qu'appelle-t-on workflow ?

Le *workflow* est la discipline ou le concept d'organisation des circuits de communication et de validation de l'information.

Les autres dépendances de Plone

CMFQuickInstallerTool

C'est le produit qui facilite la configuration des nouveaux composants *plug-ins* CMF que l'on veut intégrer dans un site Plone. Ce sont ses services qui sont exposés par le `Plone Control Panel`, accessible via l'interface `Installation de nouveaux produits`.

CMFActionIcons

Ce nouveau produit, proposé par Tres Seaver, permet d'associer des icônes aux actions du site. L'interface utilisateur s'appuie sur cet outil pour exposer des barres d'actions à base d'icônes (les fameuses *toolbars*).

JARGON Qu'est–ce qu'une action ?

Une action est l'abstraction dans l'architecture CMF qui permet de représenter une interaction web (entre l'utilisateur et le site) en lui associant de la logique applicative. Les actions sont gérées grâce au composant `CMF Actions Tool`. Pour en savoir plus, reportez-vous au chapitre 5.

CMFFormController

Ce nouveau produit fournit les mécanismes sur lesquels s'appuie la gestion de la validation et des actions associées aux formulaires au sein du site.

HISTORIQUE Composants portal_form et portal_form_controller

CMFFormController fournit un nouveau composant nommé `portal_form_controller`, qui remplace le composant `portal_form` de Plone 1.

GroupUserFolder

Développé par Ingeniweb, GroupUserFolder est un système d'authentification de Zope qui permet la gestion des groupes d'utilisateurs. Les comptes utilisateurs peuvent provenir de plusieurs sources (le serveur Zope lui-même, un annuaire LDAP, ou une base SQL). GroupUserFolder est intégré en standard dans Plone depuis la version 2.0 et est complété par des services permettant la mise en œuvre d'applications de *groupware*.

▸ http://sourceforge.net/projects/collective

Formulator

Développé par Maartijn Faassen d'Infrae, Formulator est le produit Zope le plus complet pour la problématique de gestion technique des formulaires web.

▸ http://sourceforge.net/projects/formulator/

En ce qui concerne Plone 2.0, les services de Formulator sont maintenant remplacés par ceux de CMFFormController, mais le produit est encore inclus dans la distribution de Plone.

BTreeFolder2

Il s'agit du produit qui ajoute à Zope la classe `BTree Folder`, base technique des types de dossiers capables d'avoir des milliers de contenus, comme le nouveau type `Large Plone Folder` introduit dans Plone 2. Les `BTree Folders` stockent leurs sous-objets dans une structure Python de type `BTree`, pour plus de performance.

Les produits d'extension de Plone

Les produits de développement

Archetypes

Il s'agit d'une suite logicielle ou *framework* d'aide au développement de types de contenu CMF. Concrètement, Archetypes permet au développeur de définir un « schéma » et, à partir de celui-ci, produit le code nécessaire à la logique du type de contenu. Beaucoup de composants de la communauté Plone s'appuient sur cette technologie et elle devrait devenir la base de tous les types d'objets « métiers » à l'avenir.

DÉPENDANCES **Archetypes**

Archetypes est ce qu'on appelle un mini-framework ; un framework spécialisé à l'intérieur du framework CMF.
Archetypes s'appuie sur deux modules Python additionnels inclus dans sa distribution : les modules `generation` et `validation`. Vous devez les installer dans le répertoire des modules Python de votre serveur, typiquement `[Zope]/lib/python`, ou dans le répertoire des produits Zope, auquel cas ils sont pris en compte en tant que produits Zope.

PortalTransforms

Complémentaire de l'outil Archetypes, ce composant permet d'effectuer des transformations de formats pour les attributs textuels (`Texte structuré`, `HTML`, `RTF`, `Word`, etc.) définis pour un type de contenu.

Les extensions pour le support multilingue

PlacelessTranslationService

PlacelessTranslationService (PTS) fournit le moteur de l'internationalisation (i18n) des pages d'un site construit avec Plone 2.

HISTORIQUE **PlacelessTranslationService *vs* TranslationService et Localizer**

La solution PlacelessTranslationService (PTS) remplace le couple TranslationService et Localizer utilisé pour internationaliser l'interface des sites avec Plone 1.

PloneLanguageTool

Complémentaire de PTS, ce produit rend paramétrable la négociation que doit faire le navigateur avec le serveur pour afficher les pages d'un site multilingue dans la langue que souhaite l'utilisateur. Les paramètres utilisés dans cette négociation sont par exemple :

- la langue par défaut du navigateur,
- les langues utilisées pour le site, et celle par défaut,
- la langue explicitement demandée par l'internaute via un paramètre présent dans l'URL ou dans un cookie.

I18NLayer

I18NLayer est le produit CMF/Plone/Archetypes permettant de mettre en œuvre du contenu multilingue de manière aisée. Son principe est d'encapsuler plusieurs versions du contenu (par exemple, un document traduit en plusieurs langues) au sein d'un objet conteneur.

CULTURE **i18n ?**

Le monde de l'informatique utilise bien souvent des barbarismes : i18n en est un parfait exemple. I18n signifie « internationalisation », et se note i18n car il y a 18 lettres entre le « i » et le « n »… La localisation se note l10n
C'est sûr : les informaticiens sont des poètes !

Les produits de la gestion éditoriale

PloneArticle

PloneArticle est un composant Plone qui fournit un type de contenu pour la publication d'articles et/ou de documents intégrant des images et des pièces jointes. C'est la solution idéale pour les sites fortement orientés « Contenu éditorial » ou pour les intranets avec un besoin d'intégration des documents bureautiques de l'entreprise. La recherche *full text* s'applique au contenu des fichiers bureautiques intégrés en pièces jointes.

PloneExFile

Ce composant fournit un type de contenu pour la gestion de fichiers bureautiques dans un site Plone. Il vous permet de stocker le contenu d'un fichier de votre machine en tant qu'objet de contenu au sein du site Plone. L'objet est stocké dans la base de données de Zope et est publié selon le *workflow*. Les utilisateurs peuvent télécharger le fichier en pièce jointe à tout moment. Ainsi, vous pouvez bénéficier de la recherche *full text*.

DÉPENDANCES **ZAttachmentAttribute et ZAAPlugins**

ZAttachmentAttribute est une extension Zope qui fournit la couche technique à la fonctionnalité d'association de pièces jointes aux documents d'un site fait en Zope. Ce produit est complété par le composant ZAAPlugins qui fournit à un site Plone des éléments d'interface graphique et des outils de gestion de formats textuels ou multimedias pour les pièces jointes.

Ces deux produits servent de base aux composants PloneExFile et PloneArticle pour permettre d'intégrer des fichiers bureautiques aux documents ou articles.

ALTERNATIVES

Autres options que vous pouvez installer

Éditeurs de texte :

- CMFVisualEditor
- Epoz

Outils d'aide au développement :

- External Editor (pour l'édition de scripts et templates)
- ZopeTestCase & CMFTestCase

Installation du serveur Zope

Depuis la version 2.7, Zope dispose d'une nouvelle procédure d'installation. Elle est à la fois plus simple et plus flexible. Un exécutable est fourni pour une installation rapide sur plate-forme Linux ou MS-Windows.

PLATE-FORME MS-Windows, Linux ou Unix ?

Zope fonctionne indifféremment sous MS-Windows, Unix (Solaris, freeBSD, MacOS X) ou Linux (Red Hat, Debian…).

POUR LES PRESSÉS Utiliser un installeur « tout en un »

Si vous êtes pressé, vous pouvez télécharger un installeur permettant d'installer Zope et tous les modules additionnels pour Plone (y compris Archetypes) à partir de Plone.org. Un installeur de ce type est fourni pour les plates-formes MS-Windows et MacOS X.

TÉLÉCHARGEMENTS

Zope est téléchargeable à partir de :
- http://www.zope.org/Products/Zope

Toutes les dépendances de Plone 2 (y compris Archetypes) sont livrées dans la même archive téléchargeable à partir de :
- http://www.plone.org/download
- http://sf.net/projects/plone

Sous MS-Windows

Installation du logiciel et de l'instance Zope

Créez votre dossier de travail (`C:\zope` par exemple), puis lancez le programme d'installation via l'exécutable fourni sur Zope.org (par exemple, `Zope-2.7-win32-x86.exe` pour la version 2.7). L'installeur vous demande de choisir le chemin sous lequel vous voulez installer le logiciel Zope. Vous choisissez par exemple le répertoire `C:\zope\zope27`.

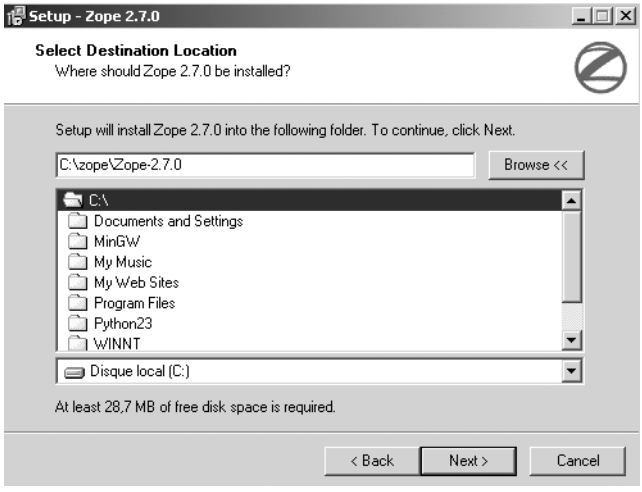


Figure 3–1 Choix du chemin d'installation du logiciel

L'installeur vous demande ensuite de choisir le répertoire dans lequel seront installés les programmes et données de la première instance de serveur Zope. Vous pouvez choisir par exemple `C:\zope\instances\PloneDev`.

Le dernier écran de configuration vous demande de choisir le mot de passe du compte d'administration initial nommé `admin`. Après la validation de cet écran, le programme d'installation copie les différents fichiers à l'emplace-

Installation multi-instances

L'installation dite « multi-instances » permet de mutualiser plusieurs instances de serveur « Zope/Plone » sur le même logiciel Zope (celui-ci installé dans le dossier `C:\zope\zope27` par exemple, représenté par la variable d'environnement `SOFTWARE_HOME`). En pratique, cela facilite la maintenance de plusieurs environnements de développement Zope sur la même machine. Typiquement, vous pouvez avoir plusieurs instances de serveur (`PloneDev`, `PloneTest`, etc.) et cela vous permet d'être plus réactif et productif.

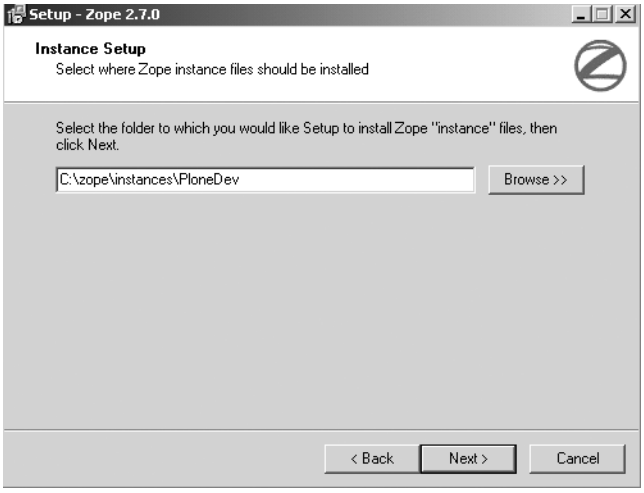


Figure 3–2 Choix du chemin d'installation de l'instance

ment prévu, compile les programmes Python qui doivent l'être et fait le paramétrage initial de l'instance du serveur Zope (initialisation de la base de données objet stockée dans le fichier `C:\zope\instances\PloneDev\var\Data.fs`, création du compte d'administration `admin`, etc.).

Démarrage du serveur

Pour démarrer votre instance de serveur Zope, il suffit d'exécuter le programme DOS `runzope.bat` qui se trouve dans le répertoire `C:\zope\instances\PloneDev\`.

Sous Linux ou Unix

Préparation

Créez un compte utilisateur spécifique, que vous pouvez nommer `zope`. C'est cet utilisateur qui exécutera le serveur Zope. Créez un répertoire spécifique pour héberger l'installation, typiquement `/usr/local/zope`, et paramétrez l'utilisateur `zope` comme propriétaire de ce répertoire. Téléchargez l'archive du code source de Zope dans un sous-répertoire temporaire, par exemple `/usr/local/zope/tmp`, décompressez-la puis déplacez le répertoire obtenu sous `/usr/local/zope`.

Vous êtes maintenant prêt à lancer la compilation du logiciel.

Attention

Il est recommandé de ne pas installer Zope en tant qu'utilisateur `root`.

Pré-requis

Les logiciels suivants doivent être installés sur la machine :

- Python 2.3.3 ou plus récent ;
- un compilateur C tel que gcc (GNU C Compiler) ;
- GNU make.

B.A.-BA La commande tar

La commande `tar` permet de décompresser un fichier d'archive. Pour information, « tar » signifie *Tape Archive*. Il s'agit du programme historique sous Unix permettant de faire des sauvegardes sur bande magnétique.

Compilation et installation du logiciel

Pour compiler le code source et lancer l'installation de Zope, exécutez les trois commandes suivantes l'une à la suite de l'autre :

```
./configure --prefix=/usr/local/zope/zope27
make
make install
```

AIDE

La commande `./configure -help` permet de trouver toutes les options fournies par le script de configuration.

ATTENTION **Plusieurs versions de Python installées**

Si vous avez plusieurs Python sur la machine, le programme de configuration utilise la version 2.3.3 par défaut. Si vous le désirez, il est possible de spécifier une version plus récente, en utilisant l'option `-with--python`.

Création de l'instance Zope

Pour créer votre instance de serveur, exécutez le script Python `mkzopeinstance.py` disponible dans le répertoire `bin` du logiciel Zope :

```
cd /usr/local/zope/zope27/bin
mkzopeinstance.py
```

AIDE

La commande `mkzopeinstance.py --help` permet de trouver toutes les options avec lesquelles vous pouvez spécifier la création de l'instance.

Le script vous demande de spécifier le nom d'utilisateur et le mot de passe à utiliser pour la création du compte d'administration de l'instance (compte initial).

Démarrage du serveur

Pour démarrer votre instance Zope, vous pouvez simplement utiliser le script `runzope` fourni par l'instance :

```
cd /usr/local/zope/PloneDev/bin
./runzope
```

Installation de produits d'extension

Vous devez ensuite installer les produits d'extension du serveur Zope (présentés au début de ce chapitre). Pour installer un produit d'extension, il suffit de décompresser l'archive téléchargée et de copier le(s) dossier(s) obtenu(s) dans le répertoire des produits, typiquement `/PloneDev/Products`. Ensuite, il faut démarrer (ou re-démarrer) le serveur pour que les nouveaux produits soient pris en compte.

ASTUCE

Le dossier correspondant à un produit Zope est reconnaissable entre autres par un module spécial qu'il contient, nommé `__init__.py`.

Prise en main du serveur

Pointez votre navigateur sur une URL du type http://localhost:8080/manage (ou http://127.0.0.1:8080/manage), afin d'accéder à l'interface de management de Zope.

Authentification

Pour accéder à l'interface de management de Zope (ou ZMI), vous devez vous authentifier grâce au compte d'administration (`admin` par défaut) créé lors de l'installation de Zope.

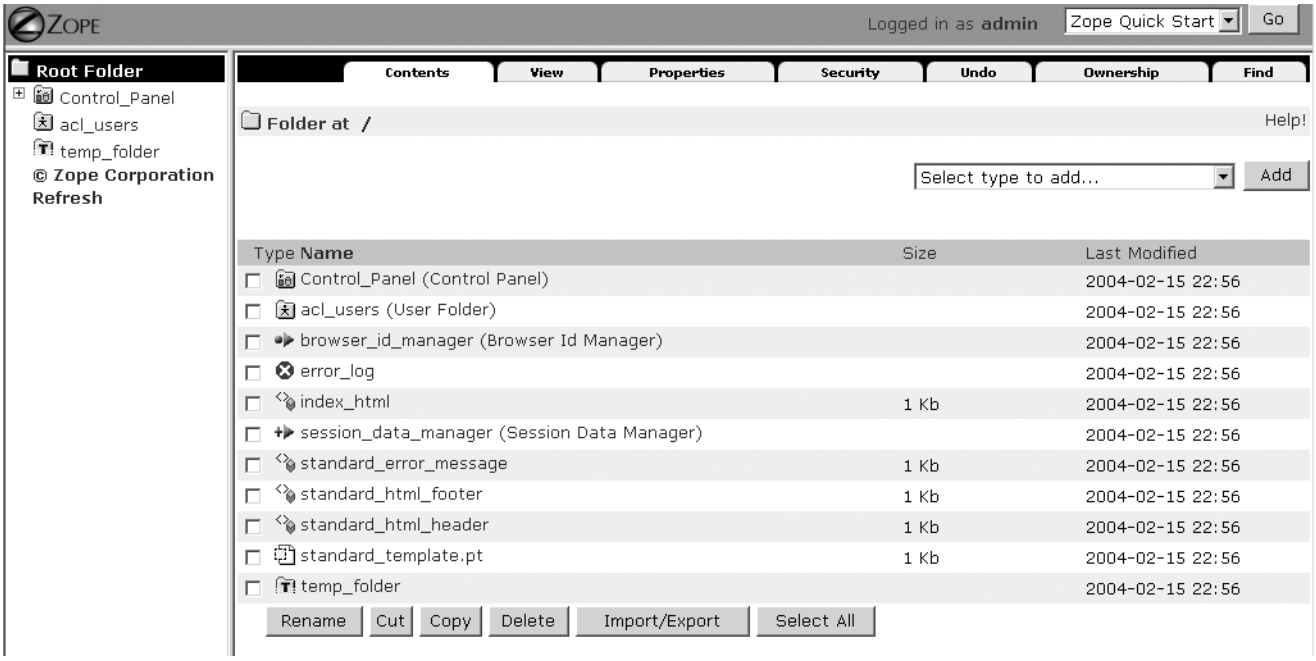


Figure 3–3 Interface d'administration de Zope

Produits installés

En cliquant sur l'objet nommé `Control_Panel`, puis sur l'objet `Products`, vous pouvez voir les objets représentant tous les produits installés : les produits de la distribution Zope par défaut, et tous les produits et options de la plate-forme Plone.

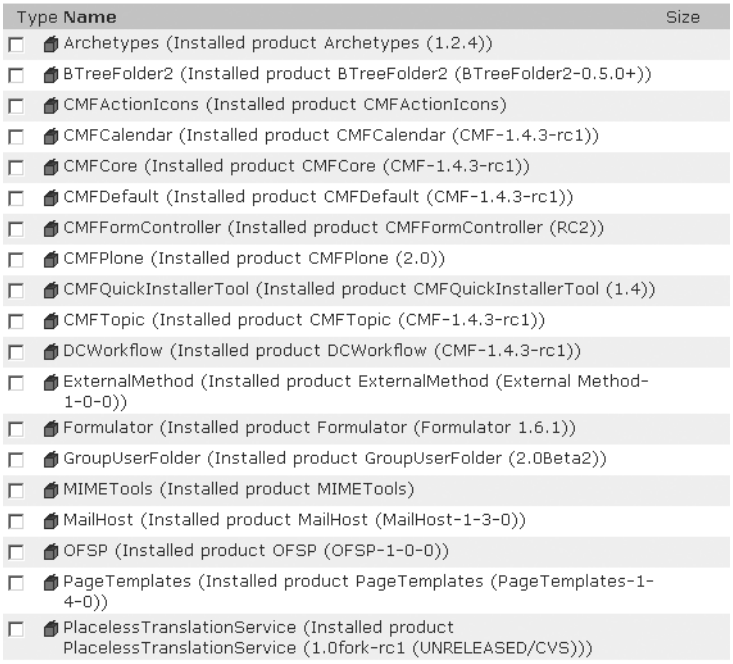


Figure 3–4 Produits enregistrés sur le serveur Zope

En résumé...

Nous avons installé toutes les briques logicielles pour la mise en œuvre de notre système de gestion de contenu Zope/CMF/Plone.

Configuration de Plone

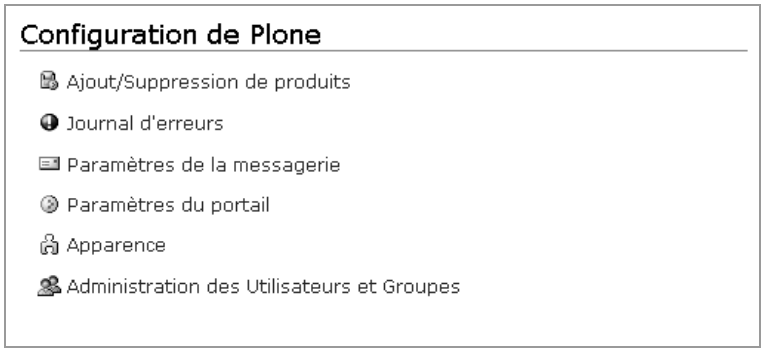
4

Zope
Plone

SOMMAIRE

MOTS-CLÉS

- Architecture
- Control Panel
- Administration
- Utilisateurs
- Groupes
- Configlets
- Multilingue



Le chapitre précédent a permis d'installer notre serveur Zope/Plone de développement ou d'intégration. Nous allons maintenant, via l'interface d'administration de Zope, puis à l'aide des panneaux de configuration fournis par Plone, aborder la configuration des objets d'infrastructure du site Plone qui servira à intégrer les services de l'intranet. Nous verrons notamment les éléments d'administration des utilisateurs et la stratégie à adopter pour gérer de manière déléguée les espaces de contenu.

Instanciation du site

Pour démarrer la configuration du site Plone, vous devez accéder au serveur Zope via l'interface d'administration et ajouter un objet de type `Plone Site`. Vous obtenez alors un formulaire de saisie des premiers paramètres de l'objet qui va constituer le site.

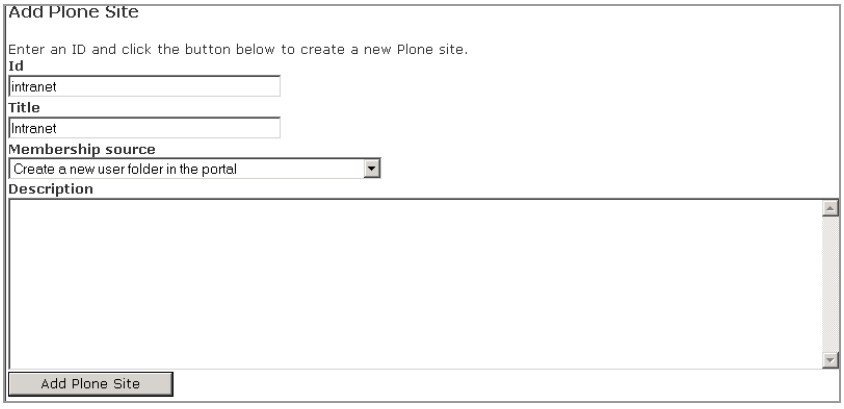


Figure 4–1 Paramétrage de création d'une instance de site Plone

Dans ce formulaire, vous êtes invité à fournir les informations suivantes :

- `Id` : c'est l'identifiant de l'objet, comparable au nom d'un fichier sur un disque dur. Par exemple, vous pouvez choisir le nom `website`.
- `Title` : c'est le titre donné au site. Ce paramètre est très réutilisé après dans le site. Il est modifiable à tout moment.
- `Membership source` : identifie l'endroit de stockage de la base des utilisateurs du site. Il existe deux valeurs : `Create a new user folder in the portal` ou `I have an existing user folder and want to use it instead`. La première option précise qu'on veut que le portail héberge en son sein la liste des utilisateurs. C'est le choix par défaut, que nous ferons ici. La seconde alternative permet de réutiliser le `User Folder` (conteneur des comptes utilisateurs nommé `acl_users`) existant à la racine (ou à un niveau supérieur de l'arborescence) de Zope.
- `Description` : description sommaire du site et des principes qui l'animent. Cette information sera réutilisée par le site à tous les moments où il donne une description de lui-même (notamment pour la page d'accueil, les moteurs de recherche, ou la syndication).

Une fois ces informations renseignées, cliquez sur le bouton `Add Plone Site`. Le serveur déclenche le processus de création et de configuration initiale du site. Cela prend quelques secondes… et à la fin, la page d'accueil du nouveau site s'affiche dans le cadre droit de l'interface d'administration.

B.A.-BA Interface d'administration Zope (Zope Management Interface)

C'est l'interface via laquelle l'on peut faire des actions de configuration poussées sur les objets techniques d'un site Zope. Pour y accéder, il suffit d'ajouter `/manage` à l'URL d'accès au site ; par exemple sur votre machine de développement, vous utilisez :

- http://127.0.0.1:8080/intranet/manage.

JARGON User Folder

L'objet de type `User Folder` nommé `acl_users` est un conteneur technique permettant l'authentification des utilisateurs du serveur Zope. En effet, c'est dans ce dossier que sont stockés les comptes utilisateurs du système, avec leur mot de passe et leurs droits (rôles). Le `User Folder` qui sera utilisé pour le site Plone est du type `Group User Folder`.

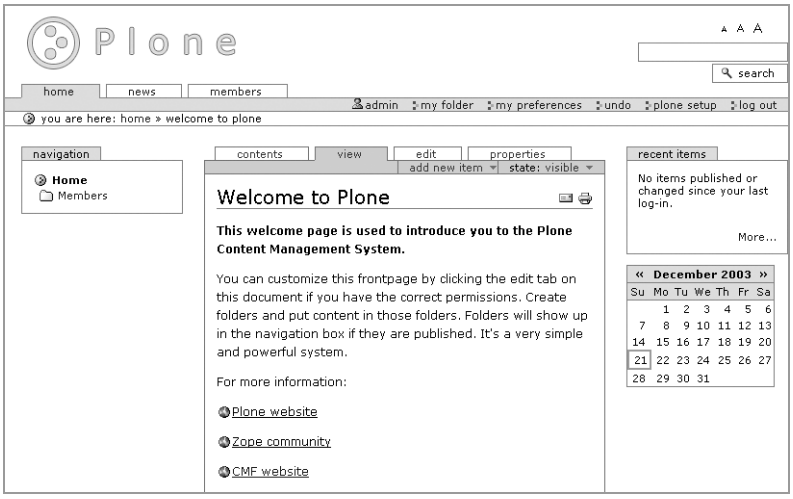


Figure 4–2 Le nouveau site Plone

Le lien `plone setup` ou `configurer plone`, en français, apparaît parce que vous avez des droits d'administration. Toutes les fonctionnalités exposées sont protégées par des permissions. Le rôle `Manager`, principal rôle de l'administrateur du site, vous permet d'effectuer toutes les actions de modification et d'administration.

Configuration et administration

Pour aller plus loin dans le paramétrage du site, vous pouvez utiliser l'interface d'administration de Plone nommée `Plone Control Panel`. Pour accéder à cette interface, il suffit de cliquer sur le lien `plone setup` (correspondant à l'URL http://127.0.0.1:8080/intranet/plone_control_panel).

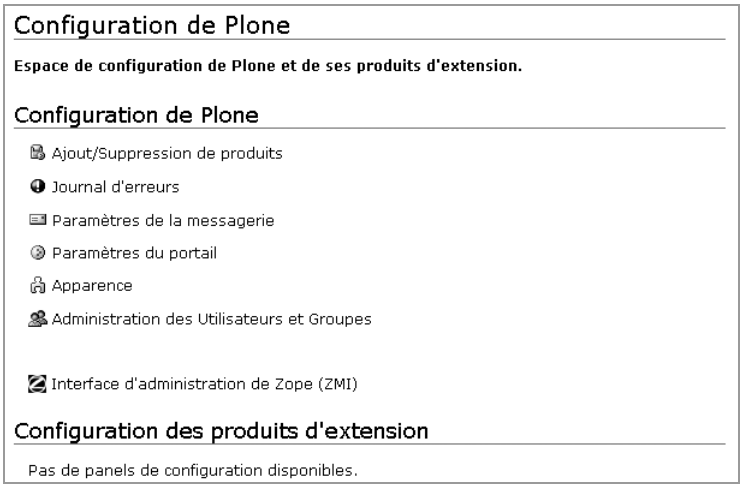


Figure 4–3 Plone Control Panel

Vous pouvez alors visiter la page de configuration ou d'administration qui vous intéresse parmi les options disponibles.

ASTUCE **Interface multilingue**

Sans que vous n'ayez eu à paramétrer quoi que ce soit, l'interface multilingue de Plone apparaît dans une langue précise. On remarque qu'il s'agit de la langue paramétrée sur le navigateur de l'utilisateur.

ADMINISTRATION

Principales pages d'administration

- Ajout/Suppression de produits
- Journal d'erreurs
- Paramètres du portail
- Paramètres de la messagerie
- Apparence (ou Habillage)
- Administration des Utilisateurs et Groupes
- Interface d'administration de Zope

ARCHITECTURE **Composants de services**

- Moteur d'indexation du contenu (`portal_catalog`) – Ce composant est une version évoluée du moteur d'indexation de Zope, le `ZCatalog`. Techniquement, il s'agit d'un *wrapper* (en français enveloppe) autour du `ZCatalog`. C'est un élément central de Plone. Il permet d'indexer tous les contenus ajoutés au site, et de construire les pages qui ont besoin d'agréger dynamiquement du contenu (page d'accueil, page correspondant à une rubrique, etc.) rapidement et simplement.
- Gestionnaire des actions (`portal_actions`) – Ce composant permet de paramétrer les différentes actions définies pour le contexte global du site. Certaines actions sont déjà paramétrées au niveau de la logique de configuration du CMF. Les autres sont ajoutées par le concepteur du site ou le développeur, soit directement via l'interface de ce composant, soit via le paramétrage des actions au sein des autres composants qui remplissent le contrat `Action Provider` (en français, Fournisseur d'actions) vis-à-vis du `portal_actions`. La liste des fournisseurs d'actions est paramétrée, et vérifiable en accédant à l'onglet `Action Providers` du composant.
- Gestionnaire d'icônes pour l'interface utilisateur (`portal_actionicons`).
- Support des commentaires sur le contenu (portal_discussion) : Grâce à ce composant, si l'administrateur le paramètre au niveau d'un type de contenu donné, les contenus basés sur ce type de contenu peuvent être commentés par les utilisateurs qui les consultent.
- Gestionnaire des membres (`portal_membership`) – Ce composant gère le mécanisme d'authentification des membres. Il permet de définir une interface stable, indépendante des implémentations des `acl_users` utilisées.
- Gestionnaire des profils des membres (`portal_memberdata`) – Ce composant permet de définir les propriétés ou attributs que le concepteur de site CMF désire conserver pour les membres. Il est très simple d'ajouter de nouveaux attributs au profil des membres.
- Gestionnaire des groupes (`portal_groups`).
- Gestionnaire des profils des groupes (`portal_groupdata`).
- Gestionnaire des métadonnées du contenu (`portal_metadata`) : Ce composant fournit l'interface de gestion des métadonnées conformément au standard *Dublin Core* (principalement, les attributs `Title`, `Description`, `Subject`, `Format`, `Contributors`, `Language`, `Rights`). Notez que les composants `portal_metadata` et `portal_catalog` collaborent étroitement pour permettre d'avoir un outil efficace pour la recherche et la découverte d'informations : pour chaque attribut de meta-donnée, il existe au sein du portal_catalog un index permettant d'indexer l'information correspondante.
- Moteur de l'interface utilisateur (`portal_skins`) : Ce composant fournit les services de l'interface graphique du site. Il intègre un mécanisme permettant de proposer plusieurs *skins*. Une *skin* est un ensemble de mise en page graphique cohérent. Cela va dans le sens de la sacro-sainte séparation contenu/design/logique que tout gestionnaire de contenu se doit de respecter. L'objectif premier est de permettre au concepteur de personnaliser l'interface par défaut en modifiant les différents éléments de présentation.
- Gestionnaire des types de contenu (`portal_types`) – Ce composant fournit l'interface de paramétrage de tous les types de contenu disponibles pour les gestionnaires de contenu. Par exemple, vous définissez, pour chaque type de contenu, les actions par lesquelles l'utilisateur interagit avec le contenu, en fonction de ses droits.
- Moteur du *workflow* (`portal_workflow`) – Ce composant aide à définir le cycle de vie du contenu. En effet, le contenu passe par plusieurs phases pendant sa durée de vie, de sa création à son archivage ou sa destruction. Le composant de *workflow* fournit une machine à états correspondant à un diagramme dans lequel les transitions sont les actions permettant de manipuler le contenu. Il collabore avec le composant `portal_types` puisque le *workflow* est défini pour chaque type de contenu disponible sur le site.
- Support de la syndication (`portal_syndication`) – Ce composant gère la syndication du contenu à travers le site. Pour activer la syndication, le concepteur ou l'administrateur du site doit utiliser le bouton `Enable Syndication` présent sous l'onglet `Properties` du composant. À partir de là, le lien `Syndication`, présent dans la boîte d'actions gauche, permet à l'administrateur ou au membre « propriétaire » du dossier courant, d'activer et de paramétrer la syndication du contenu de ce dossier.
- Support de l'annulation des modifications (`portal_undo`) – Ce composant permet l'annulation des actions effectuées sur le contenu, pour l'utilisateur autorisé.

- Gestionnaire de l'URL du site (`portal_url`) – Ce composant offre un mécanisme permettant d'avoir accès à l'URL absolue du site, en d'autres termes l'URL du « toit » du site. Sa fonction est utile chaque fois que l'on a besoin de générer l'URL d'un endroit précis au sein du site et qu'on n'est pas dans le contexte de l'objet représentant la page en question ; une fois qu'on a obtenu l'URL du site, on peut la compléter pour obtenir l'URL voulue.
- Composant de contrôle et validation des formulaires (`portal_form_controller`) – Ce composant est très puissant et facilite grandement la tâche du développeur et de l'intégrateur Plone. Il fournit un mini-*framework* de gestion de la validation et des actions qui interviennent sur un formulaire du site ou tout simplement sur un script qui doit effectuer un traitement et redonner la main à l'utilisateur.
- Gestionnaire de la création des objets de contenu (`portal_factory`) – Ce nouveau composant intervient pendant la phase de création des objets de contenu, après le passage par le formulaire d'édition et les mécanismes de validation.
- Gestionnaire d'un calendrier d'événements (`portal_calendar`) – Ce composant permet la gestion de contenus de type `Event`, et l'accès au contenu du site par des critères de date.

ARCHITECTURE **Composants de configuration ou composants utilitaires**

- Référentiel des propriétés du site (`portal_properties`) – Ce composant fournit au site un service de stockage et de gestion de propriétés. Ainsi, les développeurs et les concepteurs de sites peuvent y définir des fiches de propriétés (`PropertySheets`) dont ils se servent pour rendre paramétrables les fonctionnalités au sein du site.
- Moteur du `Plone Control Panel` (`portal_controlpanel`) – Ce composant gère les éléments (formulaires, scripts, images) qui fournissent les pages d'administration de Plone. Ces éléments sont nommés *configlets*.
- Fonctions utilitaires de Plone (`plone_utils`) – Ce composant fournit des fonctions utilitaires communes dont on se sert fréquemment dans la logique applicative d'un site Plone, par exemple, la fonction d'édition des métadonnées d'un contenu.
- Outil de gestion de mémoire cache spécifique à chaque type de contenu (`caching_policy_manager`).
- Outil de gestion de la migration (`portal_migration`)…
- Outil de configuration de produits d'extension Plone (`portal_quickinstaller`)…

ARCHITECTURE **Composants « tierce partie »**

Les composants spécialisés suivants sont ajoutés à la racine du site lorsque vous configurez le produit correspondant via la page d'administration `Add/Remove Products`.

- Les composants fournis par `Archetypes` : `archetype tool`…
- Les composants fournis par `Portal Transforms` : `portal_transforms`, `mimetypes_registry`…

ARCHITECTURE **Autres objets de l'infrastructure**

- Le `Group User Folder`, le système d'authentification et de gestion des groupes du site.
- Les conteneurs `Members` et `GroupWorkspaces` permettent (par défaut) de stocker le contenu respectivement des membres et des groupes de travail (pour les besoins du *Groupware*).
- L'objet `MailHost` permet d'envoyer des messages électroniques à partir du site. On retrouve la valeur qu'on a saisie pour le champ `SMTP Server` lors de l'installation. Ce composant est automatiquement installé à la racine du site.
- L'objet `content_type_registry` contient une définition de correspondance entre les types MIME et les types de contenu, qui eux sont définis au sein du composant `portal_types`.
- L'objet `Cookie Crumbler` (`cookie_authentication`) est responsable de maintenir des paramètres d'authentification, au sein d'un *cookie* stocké dans le navigateur de l'utilisateur connecté. On peut notamment y changer les noms des variables sérialisées dans le cookie. Grâce à l'objet `Cookie Crumbler`, il est possible de demander le login et le mot de passe de l'utilisateur au sein d'un formulaire HTML classique.
- Les objets HTTP `Cache Manager` (HTTPCache) et `RAM Cache Manager` (RAMCache).
- L'objet qui expose le journal d'erreurs (`error_log`).

Paramètres généraux du site

Les paramètres du fonctionnement général du site Plone sont regroupés dans un formulaire accessible via le lien `Portal Settings`. Les paramètres essentiels sont présentés ci-dessous :

- `Portal title` : vous retrouvez ici le titre du site, que vous aviez entré via le formulaire de création.
- `Portal description` : vous retrouvez ici la description du site.
- `Portal 'From' name` : nom utilisé pour l'en-tête `From` des messages envoyés par le site (notification par courrier électronique), typiquement le message de confirmation à envoyer au membre du portail lors d'une nouvelle inscription ou lorsqu'il veut retrouver son mot de passe.
- `Portal 'From' address` : adresse utilisée pour l'en-tête `From` des messages envoyés par le site.
- `Default language` : langue par défaut utilisée dans les propriétés (attributs) des objets ; sélectionnons donc `french`. Ce paramètre est important, étant donné que le site peut être multilingue.
- `Password policy` : définit la politique de sélection du mot de passe initial lors de l'inscription d'un membre. Pour une gestion simplifiée, il est probablement préférable de conserver le choix par défaut : `Allow members to select their initial password`.

Une fois ces choix effectués, cliquez sur `save` pour valider et mettre à jour le site avec vos changements.

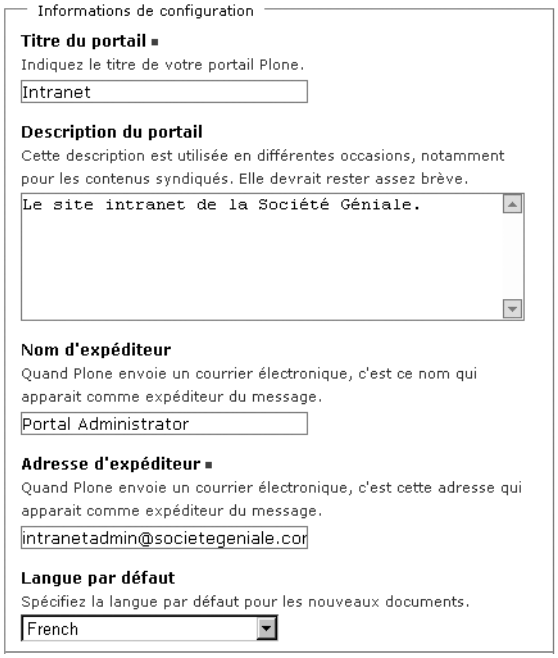


Figure 4–4
Paramètres du portail

Connexion au serveur de courrier électronique

Pour renseigner les paramètres de connexion au serveur de messagerie (serveur SMTP), visitez la page correspondant au lien `Mail Settings`. Les paramètres en question sont :

- `SMTP server` : définit le nom DNS (ou l'adresse IP) du serveur SMTP qui sera utilisé pour l'envoi de courriers électroniques à partir du site. Si le serveur SMTP (Sendmail, Postfix, Qmail ou autre) est installé sur la même machine, il suffit de garder la valeur par défaut, `localhost`.
- `SMTP port` : définit le port d'écoute du serveur SMTP, dont la valeur par défaut est `25`.

Comptes utilisateurs et groupes

Comptes utilisateurs

En général, les comptes des membres du site sont automatiquement créés à leur inscription. Néanmoins, l'administrateur peut explicitement créer de nouveaux comptes. Pour créer un nouveau compte, il suffit d'accéder à la page de gestion des utilisateurs (onglet Utilisateurs) à partir du panneau Administration des Utilisateurs et Groupes, et d'utiliser le bouton ajouter... réservé à cet effet.

Figure 4–5
Ajout d'un membre

B.A.-BA Le DNS

DNS ou *Domain Name System* est un service fondamental des réseaux TCP/IP en général, et d'Internet en particulier. C'est le service chargé de faire la correspondance entre le nom d'une machine connectée au réseau (ex. server1.acme.com) et l'adresse IP unique qui identifie cette machine.

CULTURE Serveur SMTP

Le serveur SMTP est le serveur responsable de l'émission des courriers électroniques. C'est à un serveur SMTP que votre client de messagerie confie vos précieux messages pour les envoyer. Si vous utilisez un service d'hébergement professionnel, la machine héberge sûrement déjà un tel serveur ; dans ce cas, c'est `localhost` qu'il convient de mettre. Dans le cas où vous testez le serveur depuis votre domicile, il faut spécifier le serveur SMTP de votre fournisseur d'accès, par exemple smtp.wanadoo.fr.

Groupes

Vous pouvez maintenant ajouter les groupes de l'intranet. Pour ajouter un groupe, cliquez sur l'onglet **Groupes** à partir du panneau de gestion des utilisateurs et des groupes, puis utilisez le bouton **ajouter...** Vous pouvez définir un groupe de membres par son nom, ses métadonnées (par défaut, `titre` et `description`), et son adresse e-mail.

Pour éditer les informations d'un groupe, il suffit de cliquer sur le lien du groupe (colonne **nom du groupe**) dans la matrice **Groupes/Rôles**. Vous arrivez sur la vue du groupe choisi, offrant deux onglets de paramétrage : celui des propriétés et celui des membres.

La méthode consiste à créer des groupes correspondant à des équipes ou à des services au sein de l'entreprise (par exemple Marketing), afin de mettre en œuvre une gestion de contenu déléguée aux services qui en ont besoin.

Concrètement, chaque fois qu'il est utile d'avoir une gestion déléguée au niveau d'une rubrique ou d'un dossier du site, nous attribuerons au groupe concerné le rôle requis (par exemple, le rôle `Propriétaire`) de manière locale.

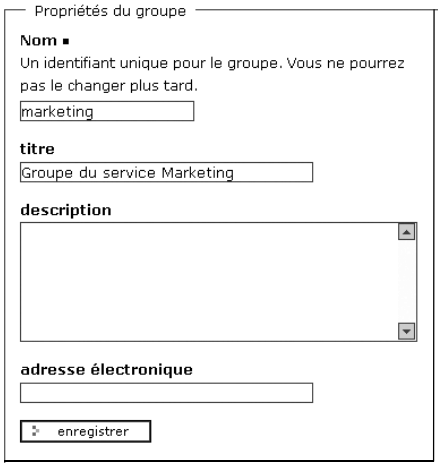


Figure 4–6 Ajout d'un groupe

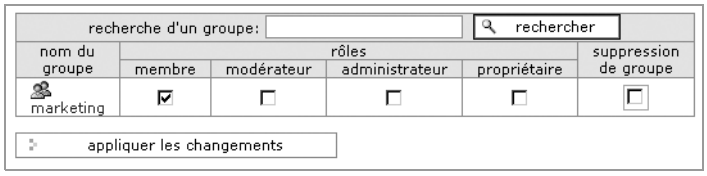


Figure 4–7 Matrice Groupes / Roles

Affecter des utilisateurs à un groupe

Pour affecter des membres à un groupe, cliquez sur l'onglet **membres du groupe** à partir de la vue du groupe. Cette interface, complètement intuitive, vous permet de sélectionner des membres du site et de les associer au groupe.

Figure 4–8
Membres d'un groupe

Politique de gestion déléguée

La création des dossiers qui serviront à stocker le contenu centralisé (parfois appelé « base documentaire »), fait également partie du travail de configuration initial. L'administrateur du site peut créer chaque dossier, puis y paramétrer les droits locaux de création et modification de contenu.

Comme nous l'avons vu au chapitre 2, l'arborescence conseillée pour la gestion du contenu est du type :

```
[Contenu]
  Marketing
     - actus
     - produits
     - liens
  SAV
     - actus
     - procedures
     - solutions
     - liens
  Direction
     - procedures
     - memos
     - gestion
```

Ainsi, nous pouvons accorder aux membres du groupe `marketing` les droits de gestion du contenu au sein du dossier `Marketing`. Cela consiste à leur accorder le rôle local `Propriétaire`, en passant par l'onglet **Partage** du dossier (figure 4-9).

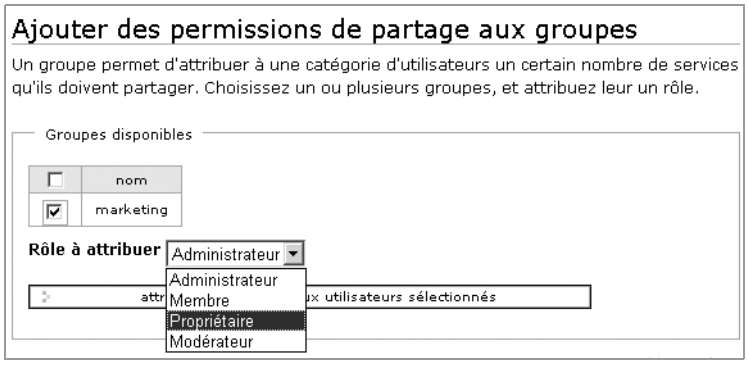


Figure 4–9 Paramétrage de rôles locaux (Partage)

Configuration avancée

Produits d'extension

Tous les produits qui apportent de nouvelles fonctionnalités pour étendre Plone peuvent être configurés via la page d'administration Ajout/Suppression de produits.

Il est conseillé de configurer les produits PloneLanguageTool, Archetypes, et PortalTransforms, afin d'intégrer leurs fonctionnalités au site.

Exemples de produits d'extension

Ces produits doivent avoir été installés sur le système de fichiers du serveur (dans le dossier `/Products`). Quelques produits d'extension à configurer :

- PloneLanguageTool
- Archetypes
- PortalTransforms
- PloneArticle
- PloneExFile
- ZAAPlugins
- I18NLayer
- Epoz

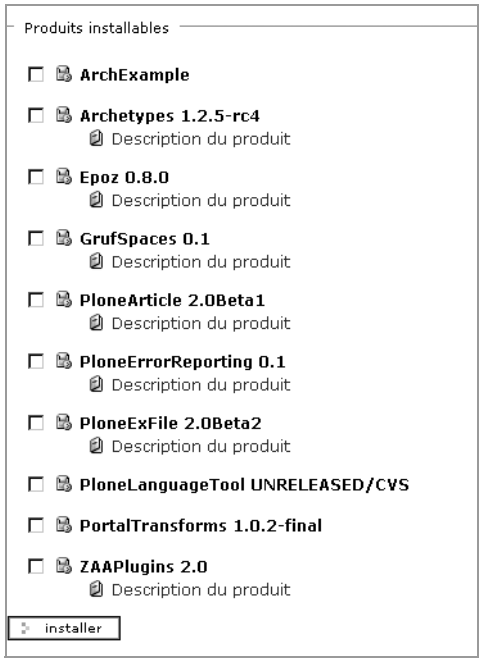


Figure 4–10 Configuration de produits

Multilinguisme

Le site Plone peut fonctionner en mode multilingue. Pour l'instant, grâce au composant `PlacelessTranslationService`, la langue de l'interface utilisateur est la langue paramétrée au niveau de votre navigateur.

Lorsque vous avez configuré le produit `PloneLanguageTool`, un nouveau composant nommé `portal_languages` a été ajouté au sein du site. Ce composant offre une interface de contrôle de la négociation du multilingue entre le navigateur et le serveur. En visitant le composant `portal_languages` via la ZMI, vous pouvez ainsi paramétrer différentes langues pour la gestion de l'interface multilingue, et choisir le « français » comme langue par défaut.

Interface graphique

Pour tous les détails sur le paramétrage de l'interface graphique du site, reportez-vous au chapitre suivant : Personnalisation graphique.

Rappel

ZMI : *Zope Management Interface*

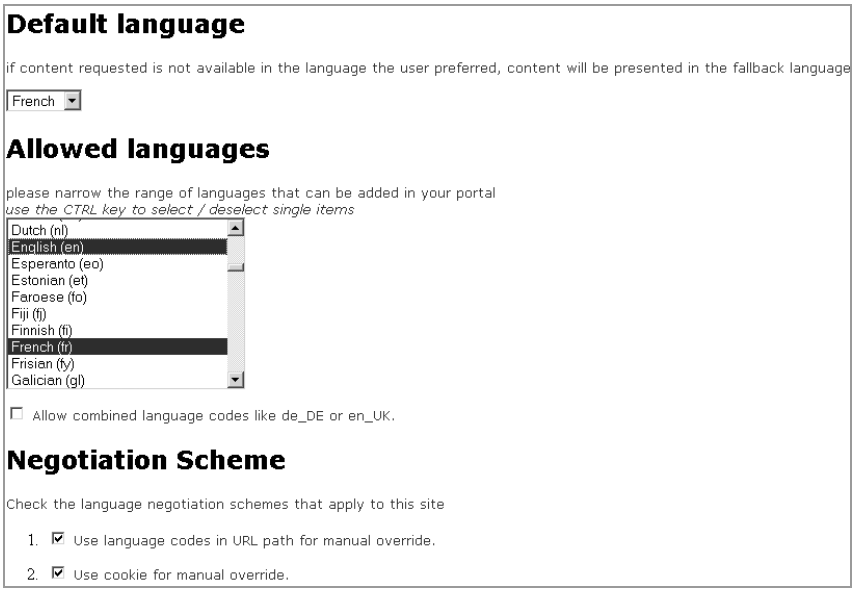


Figure 4–11 Paramétrage du composant `portal_languages`

Figure 4–12 Boîte permettant de changer de langue

ATTENTION **Jeu de caractères**

Si vous remarquez que les caractères accentués dans les premiers contenus que vous créez ne s'affichent pas correctement, vous devez ajuster le jeu de caractères paramétré pour le site. Via la ZMI, visitez l'objet `site_properties` situé au sein du composant `portal_properties`. Modifiez le paramètre `default_charset` de cette fiche de propriétés avec la valeur `ISO-8859-1`. Il s'agit du jeu de caractères par défaut pour un site en français.

Autres outils d'administration

- le journal d'erreurs (`error_log`),
- l'assistant de migration (`portal_migration`).

En résumé...

Nous avons donc une suite de composants qui collaborent pour fournir les services de la gestion de contenu. Nous allons maintenant nous lancer dans la mise en œuvre de l'étude de cas. Pour chacun des aspects de la réalisation du site (graphisme, fonctionnalités, déploiement), nous ferons appel à un certain nombre de ces composants.

Personnalisation graphique

5

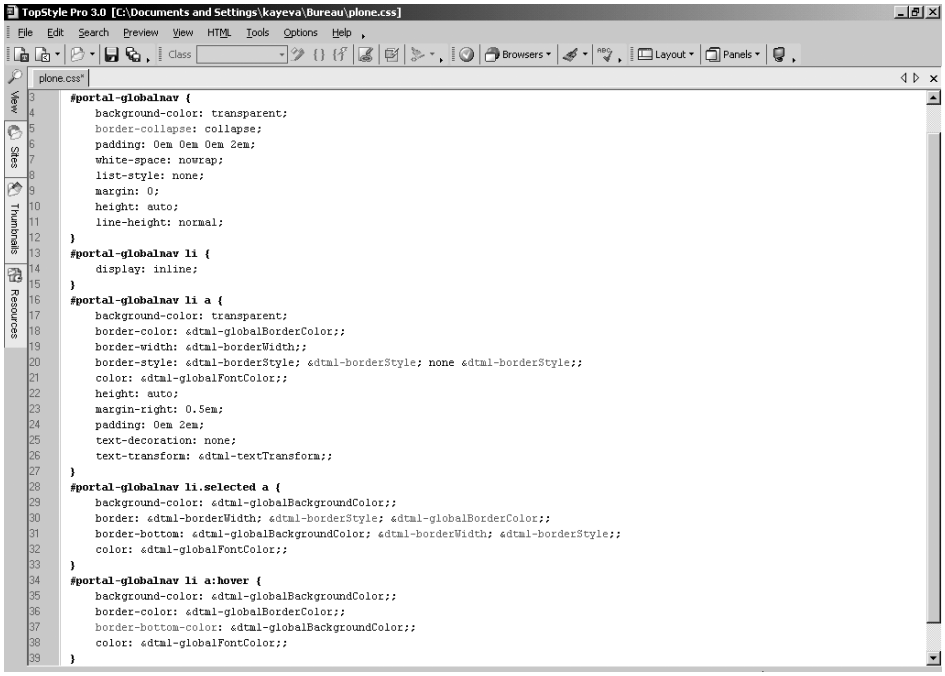
Zope Plone

CSS | XHTML | templates | ZPT | ergonomie | accessibilité

SOMMAIRE

MOTS-CLÉS

- CSS
- XHTML
- Templates
- ZPT
- Ergonomie
- Accessibilité



Vous avez travaillé avec votre graphiste pour réaliser la maquette de votre site intranet. Vous avez ensuite extrait les éléments graphiques (principalement des images) et produit la version HTML de cette maquette.

Méthodologie et bases théoriques

Même la phase de personnalisation graphique nécessite de la méthode. La démarche méthodologique est facilitée par le fait que Plone fournit une interface utilisateur structurée. Ainsi, vous pouvez organiser votre travail en différentes phases, chaque phase faisant appel à une technique ou une technologie spécifique. Typiquement, vous devez travailler sur les éléments suivants :

- Le gabarit des pages, structure de base de toutes les pages du site. On y retrouve l'ensemble des éléments de navigation (liens internes, liens externes, menus contextuels, *breadcrumbs*, *portlets*, etc.) et la zone centrale réservée au contenu.
- La charte graphique – C'est l'ensemble des éléments visuels utilisés pour la communication de l'entreprise : logos, couleurs, polices de caractères, icônes et pictogrammes, etc.
- La page d'accueil – C'est une page fondamentale pour le succès d'un site, et d'un intranet en particulier. En effet, elle doit inciter l'utilisateur à revenir fréquemment sur le site en lui proposant des informations pertinentes (et actualisées !) et un aspect attractif.

Dans la suite du chapitre, vous allez découvrir, à travers des exemples, les outils proposés par Plone pour personnaliser l'apparence et l'ergonomie de votre site intranet. Vous découvrirez notamment que l'apparence d'un site Plone peut être changée sans toucher aux *templates* (les objets de présentation de Zope), mais simplement en modifiant des règles dans la feuille de style de Plone grâce au langage CSS.

En effet, et c'est un point fondamental de Plone 2, la séparation entre le contenu et la présentation est totale : le changement de tous les éléments graphiques peut se faire sans même toucher aux pages HTML de Plone. 95 % de la personnalisation graphique peut se faire sur la base du changement des simples feuilles de style CSS2. Dit autrement, cela veut dire que deux sites Plone 2, graphiquement totalement différents, peuvent partager le même code HTML…

Références et sites incontournables

« Jeffrey Zeldman presents… » :
▸ http://www.zeldman.com

Éric Meyer :
▸ http://www.meyerweb.com/

A List Apart :
▸ http://www.alistapart.com

The Web Standards Project :
▸ http://www.webstandards.org/

CSS Zen Garden :
▸ http://www.csszengarden.com/

Tantek Çelik :
▸ http://tantek.com

Feuilles de style CSS

Les feuilles de style ou CSS (*Cascading Style Sheets*) permettent de contrôler finement la mise en page des différents éléments HTML d'une page web. Avec CSS, vous pouvez définir la couleur d'une police, sa taille, le positionnement absolu ou relatif, l'espacement entre les paragraphes, les effets de transition affectant des liens, ou encore adapter la mise en page au média de sortie (écran ou imprimante, par exemple).

Les normes techniques du W3C (World Wide Web Consortium)

- XHTML – C'est la nouvelle génération du langage HTML (après HTML 4). La grosse différence est que XHTML respecte le standard XML. http://www.w3.org/TR/xhtml1.
- CSS – Il s'agit de la spécification des *Cascaded Style Sheets*. Il y a deux versions : CSS Level 1 et CSS Level 2 (1998). http://www.w3.org/Style/CSS.
- DOM – La spécification DOM ou *Document Object Model* du W3C fournit une représentation d'un document (sous la forme d'un arbre), de manière à permettre de manipuler la présentation du document de manière flexible au niveau du navigateur. Techniquement, le DOM permet d'interpréter chaque élément d'une page HTML comme un objet accessible en parcourant l'arbre représentant le document. Par exemple, vous pouvez grâce à la méthode `getElementById` définie dans le langage JavaScript, accéder aux éléments de la page HTML et les manipuler. http://www.w3.org/DOM.

Les recommandations pour l'accessibilité

- Section 508 – Les recommandations d'accessibilité du gouvernement américain. http://www.section508.gov.
- WAI-WCAG – La WAI (*Web Accessibility Initiative*) du W3C a publié une série de directives dénommées *Web Content Accessibility Guidelines*. Ces directives expliquent comment rendre les contenus web accessibles aux personnes handicapées. En les suivant, le contenu web pourra être plus accessible non seulement aux personnes handicapées, mais également à tous les utilisateurs, indépendamment du programme utilisateur (navigateur classique, logiciel vocal, téléphone mobile, etc.) http://www.w3.org/WAI.

Et Plone dans tout ça ?

Plone respecte les normes XHTML et CSS pour son architecture de présentation, et les recommandations *Section 508* et *Level Triple-A Conformance to Web Content Accessibility Guidelines* du WAI (WAI-AAA).

CULTURE JavaScript

Développé à l'origine par la société Netscape Communications pour son navigateur, JavaScript est aujourd'hui un langage très utilisé pour des traitements que le développeur veut faire exécuter du « côté client » (*client-side scripting*).

Définitions

Une feuille de style est composée de règles qui s'appliquent soit à l'ensemble des éléments HTML du document (par exemple, l'élément `h1`), soit à un élément particulier qui est défini par un des attributs `class` ou `id`.

Les règles s'appliquant au document sont incluses dans le bloc de déclaration CSS qui est fourni par la balise spéciale `<style>` ou au sein d'un fichier externe (d'extension `.css`) référencé dans l'en-tête du document.

Une règle de style est composée d'un sélecteur et d'une déclaration. La déclaration est composée d'un certain nombre de propriétés.

Voici un exemple avec des déclarations CSS simples.

```
<style>
```
Début du bloc de déclarations CSS.

```
h1 { color: red }
```
Dans cette règle, `h1` est le sélecteur, et la déclaration définit pour la propriété `color` la valeur `red`.

```
table { background-color : white ; border : 1px black solid }
```
Cette règle définit le style de l'élément table avec une déclaration composée de trois propriétés.

```
h1, h2, h3 { color: red }
```
Cette règle s'applique aux trois éléments h1, h2 et h3.

```
</style>
```
Fin du bloc de déclaration CSS.

En dehors des sélecteurs d'éléments (voir l'exemple CSS), il existe deux autres types de sélecteurs qui permettent d'accéder à la puissance de CSS :

- les sélecteurs de classe,
- les sélecteurs d'identificateurs.

Utilisation des sélecteurs de classe

Les sélecteurs de classe permettent de regrouper des éléments HTML spécifiques, relatifs à un même sujet. La règle utilisant le sélecteur aura un effet sur tous les éléments qui l'utilisent dans leur attribut `class`.

Par exemple, prenons l'élément `div` utilisé avec la classe `commentaire` ci-dessous :

```
<div class="commentaire" >Ceci est mon commentaire</div>
```

S'il existe la règle suivante dans la feuille de style…

```
.commentaire { color: #CCCCCC }
```

… alors le texte « Ceci est mon commentaire » sera affiché en gris.

Cette règle s'appliquera à toute balise que vous utiliserez avec la valeur `commentaire`, pour son attribut `class`, et non pas uniquement aux balises `<div>` (comme dans l'exemple).

ALTERNATIVE

Si vous voulez que la règle ne s'applique qu'aux éléments `<div>`, vous devez la modifier en :
`div.commentaire { color: #CCCCCC }`

Utilisation des sélecteurs d'identificateurs

Les identificateurs permettent d'appliquer une règle de style à un élément unique du document HTML.

La déclaration d'un style par identificateur se fait avec le caractère dièse (`#`). Pour appliquer la règle à l'élément concerné, on utilise l'attribut `id`.

Dans l'exemple suivant, l'image utilise l'identificateur `specialImage`.

```
<img id="specialImage" src="plone.jpg" />
```

La règle CSS suivante permet d'obtenir une image avec une bordure en pointillé :

```
#specialImage { border-style: dashed; }
```

ASTUCE **JavaScript**

Les identificateurs peuvent également être utilisés avec JavaScript grâce à la méthode `getElementById`, qui permet d'accéder à l'élément concerné (d'après le DOM – *Document Object Model*).

Les Pages Templates de Zope

Les environnements de développement web modernes visent à séparer la logique applicative de la présentation. C'est là qu'interviennent les systèmes de développement à base de *templates*. D'un côté, les scripts Python effectuent les traitements de la logique applicative. De l'autre, les *templates* définissent des gabarits XHTML correspondant à la charte graphique où se situent des zones prédéfinies dédiées à l'affichage des données fournies par les scripts. Dans le cas de Plone 2, on peut résumer rapidement cette séparation présentation/logique/contenu :

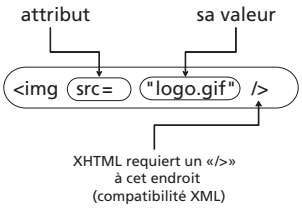
- la logique : via Python ;
- la présentation : via CSS et ZPT (dans une moindre mesure) ;
- le contenu : via les types de contenu.

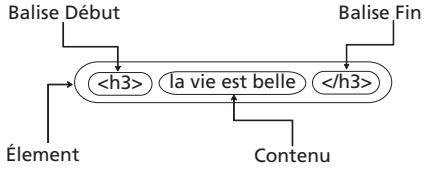
Zope fournit un système de construction de pages dynamiques, baptisé *Zope Page Templates* (ZPT). Les `Page Templates` sont des « objets exécutables Zope » permettant de définir des pages web dynamiques dont le code est conforme à la norme XHTML. Cette technologie se base sur trois nouveaux langages :

- TAL (*Template Attribute Language*) – Complète le langage HTML en fournissant à ses balises des attributs permettant d'effectuer des actions dynamiques.
- TALES (TAL Expression Syntax) – C'est le langage d'expression utilisé par les attributs fournis par TAL.
- METAL (*Macro Expansion Template Attribute Language*) – Permet la réutilisation, dans un *template* donné, de blocs de code définis au sein d'un autre *template*.

B.A.-BA Attribut

Un attribut est une valeur qui apparaît au sein d'un élément XML (en général) et donc dans une balise XHTML. Par exemple, l'élément `img` a un attribut `src` qui contient le chemin de l'image à afficher.





JARGON Template

Le terme anglais *template* correspond au français « gabarit ». En effet, les *templates* sont des scripts qui facilitent la mise en œuvre de la structure ou du gabarit des pages d'un site web **dynamique**. Elles facilitent la construction des pages, en préréglant la disposition des différents éléments graphiques : en-tête, menus, formulaires, barres de navigation, zone d'affichage d'un article, zone de liens…

Principe de fonctionnement

L'objet `Page Template` définit une page avec du code XHTML valide, dont les éléments peuvent porter des attributs spéciaux chargés d'effectuer des actions dynamiques. Ces attributs, du type `tal:content`, sont reconnus parce que la page indique (au niveau de la balise `<html>` ouvrante) l'espace de noms XML de TAL.

Quelques fonctions TAL

- `tal:content`
- `tal:define`
- `tal:repeat`
- `tal:condition`
- `tal:attributes`

Les fonctions METAL

- `metal:define-macro`
- `metal:use-macro`
- `metal:define-slot`
- `metal:fill-slot`

Pour la référence complète, voir l'annexe A sur ZPT.

Exemples d'utilisation de TAL

Insertion d'une variable

Dans cet exemple, `tal:content` est un attribut HTML spécial fourni par une fonction TAL, et `here/Title` est une expression TALES. Dans cette expression, `here` est une variable implicite qui représente l'objet courant (ou contexte).

Prenons l'exemple simple suivant :

```
<title tal:content="here/Title">Titre d'un document</title>
```

Dans votre éditeur HTML, cette balise affiche simplement « Titre d'un document ». En revanche, lorsque ce code est interprété par le serveur Zope, le titre de l'objet courant s'affiche dans la barre d'état du navigateur (ce qui est le comportement fourni par la balise HTML `<title>`).

Si le titre de l'objet est « Actualités », le résultat correspond au code HTML suivant :

```
<title>Actualités</title>
```

Définition de variables

La définition de variables est possible avec l'attribut TAL `tal:define` :

```
<span tal:define="titre here/title_or_id">
  <h1 tal:content="titre">titre</h1>
</span>
```

Il est possible d'utiliser des expressions Python comme expression TALES :

Depuis Zope 2.7, vous pouvez utiliser `context` au lieu de `here` pour accéder au contexte courant.

```
<span tal:define="court python:here.title_or_id()[:8]">
  <h2 tal:content="court">Titre raccourci</h2>
</span>
```

Dans cet exemple, vous définissez et utilisez une variable locale `court` qui ne contient que les huit premiers caractères du titre.

Boucle

Avec l'attribut `tal:repeat`, vous pouvez faire des itérations (boucles) :

```
<ul>
   <li tal:repeat="obj here/objectValues">
      <a tal:attributes="href obj/absolute_url"
         tal:content="obj/title_or_id">title</a>
   </li>
</ul>
```

La variable `obj` est l'objet actuel dans la boucle. L'URL de l'objet est affichée dans la balise `a` avec l'attribut `tal:attributes`. Le titre de l'objet est affiché dans le lien.

```
<ul>
   <li>
      <a href="http://localhost/docs/doc1">Document 1</A>
      <a href="http://localhost/docs/doc2">Document 2</A>
      <a href="http://localhost/docs/doc3">Document 3</A>
   </li>
</ul>
```

Exemple d'utilisation de METAL

Avec METAL, vous pouvez définir une macro sur un bloc de code au sein d'un *template*, afin de réutiliser ce bloc de code dans un autre *template*. Voici un exemple :

```
<html metal:define-macro="master">
  <head>
    <title tal:content="template/title">Le Titre</title>
  </head>
  <body>
    <h2><span tal:replace="here/title_or_id">content title or id
        </span>
        <span tal:condition="template/title"
              tal:replace="template/title">optional template id
        </span>
    </h2>
    <div> Ceci est ma macro </div>
  </body>
</html>
```

Ici, l'expression `define-macro="master"` permet de définir la macro en la nommant `master`.

Dans un autre *template*, défini au sein du même conteneur, insérez le code suivant :

```
<html metal:use-macro="container/macro_template/macros/master">
</html>
```

Quand vous prévisualisez le nouveau *template* en cliquant sur l'onglet `Test` de son interface, vous découvrez le même rendu qu'avec le premier *template*. De plus, la vérification du code HTML produit en utilisant la commande `Afficher la source` du navigateur permet de prouver que le code correspondant à la macro a été utilisé dans le second *template*.

Les langages TAL, TALES et METAL sont expliqués en annexe A.

Mise en œuvre

Pour réussir votre personnalisation graphique, il convient de comprendre comment fonctionne le système de *skins* de Plone, l'outil qui fournit la gestion de l'interface utilisateur.

L'architecture de Plone (basée sur le CMF) intègre un composant nommé `portal_skins` qui fournit les mécanismes de gestion d'une ou plusieurs *skin(s)*. On peut l'appeler le « moteur de *skins* » de Plone. Une *skin* est composée de *sous-skins*, des conteneurs d'objets de présentation ou de logique applicative. Ainsi l'agrégation de toutes les *sous-skins* disponibles fournit l'interface utilisateur du site.

Vous trouverez deux types de *sous-skins* au sein du composant `portal_skins` :

- les objets de type `File System Directory View`, qui sont stockés sur le système de fichiers du serveur (dans le dossier `/Products/CMFPlone/skins/`),
- les objets de type `Folder`, qui sont créés de toute pièce via le Web par l'intégrateur du site. Typiquement, la *sous-skin* nommée `custom` créée à l'installation de Plone et réservée pour votre personnalisation via le Web.

Éléments composant la skin "Plone Default"

Plone est fourni avec la *skin* `Plone Default`, qui s'appuie principalement sur les *sous-skins* suivantes :

- `custom` – *Layer* prévue par le CMF pour la personnalisation ;
- `gruf` – *Layer* pour le produit Group User Folder ;
- `plone_ecmascript` – Collection des JavaScript de Plone ;
- `plone_wysiwyg` – Gestion des éditeurs visuels ;
- `plone_prefs` – Configuration des préférences de Plone ;
- `plone_portlets` – Boîtes d'action ou d'information ;
- `plone_templates` – *Templates* qui assurent la structure des pages ;
- `plone_3rdParty/CMFTopic` – *Layer* pour le produit CMFTopic ;
- `plone_styles` – Feuilles de style (CSS) ;
- `plone_form_scripts` – Scripts d'action des formulaires ;
- `plone_scripts` – Scripts divers ;
- `plone_forms` – Formulaires ;
- `plone_images` – Images ;
- `plone_content` – Présentation du contenu ;
- `cmf_legacy` – *Layer* avec quelques scripts CMF encore utilisés par Plone.

Chaque fois que l'utilisateur interagit avec un service du site (en visitant une page), le système parcourt une série de *sous-skins* définies par le concepteur, jusqu'à trouver l'objet Zope à exécuter. S'il s'agit d'une méthode ou d'un script qui fait appel à un autre objet, une nouvelle recherche est effectuée pour trouver cet objet, et ainsi de suite jusqu'à ce que la requête de l'utilisa-

Skin et sous-skins

Une *skin* est une collection d'objets de présentation et/ou de logique applicative permettant aux graphistes et aux développeurs de construire et faire évoluer une interface utilisateur spécialisée, sans altérer le code source du logiciel. Pour faciliter cette gestion, une *skin* est organisée en objets conteneurs encapsulant les objets de présentation, et appelés *layers* ou *sous-skins*.
Selon le contexte, le terme *skin* se traduit généralement par « habillage » ou « apparence ».
En général, une *sous-skin* est chargée d'un aspect ou d'un besoin fonctionnel spécifique : charte graphique, contenu, formulaires, logique de site (scripts), interface d'administration (« Control Panel »), etc.

Moteur de skins

Le système ou moteur de *skins* intègre tout ce qui permet d'industrialiser la mise en œuvre de l'interface du site : les *templates* et les scripts, des outils d'analyse syntaxique et de création de code HTML à la volée, des mécanismes d'optimisation (compression du code, gestion du cache), etc.

Rappel

Un script Python est un objet Zope permettant d'éditer et d'exécuter du code Python via le Web.

teur soit satisfaite. La liste des *sous-skins* et l'ordre de priorité avec lequel la recherche s'effectue, sont déterminés à partir de deux paramètres complémentaires :

- La définition des *sous-skins* correspondant à chaque *skin*, faite par l'administrateur sous l'onglet `Properties` du composant `portal_skins`. C'est également là que l'administrateur définit la *skin* par défaut globale du site.
- La *skin* de préférence sélectionnée par un membre du site via sa page d'options personnelles (ou préférences d'utilisateur).

Ainsi, par défaut, les images, les scripts et les *templates* contenus dans la *sous-skin*`custom` (en première position dans la chaîne d'évaluation de la *skin*) sont toujours prioritaires sur les objets portant le même nom mais contenus dans une *sous-skin* qui est positionnée en dessous (et donc moins prioritaire).

L'utilisation de CSS dans Plone

Toutes les définitions de la CSS de Plone se trouvent dans la *layer*`plone_styles`. Les fichiers les plus importants de ce répertoire sont `plone.css` et `ploneCustom.css`.
Notez qu'il ne faut jamais modifier le fichier `plone.css`, mais que le travail s'effectue sur le fichier `ploneCustom.css` explicitement conçu pour la personnalisation.

Le gabarit des pages

Présentation

Le gabarit des pages est mis en œuvre au sein du *template*`main_template`, qui se trouve dans la *sous-skin*`plone_templates`. Elle est très technique et appelle un certain nombre de *templates* spécialisés, via le mécanisme des macros (METAL).

La partie visuelle du gabarit est composée des principaux blocs suivants :

- La barre de haut de page (ou *top bar*) – Elle contient les principaux éléments de marque tels que le logo, ainsi que l'interface de recherche et les liens vers les pages importantes du site (onglets vers les rubriques, actions personnelles de l'utilisateur, etc.).

L'identifiant CSS de ce bloc est `portal-top`.

- Le bloc des trois colonnes – Il permet d'afficher les trois colonnes correspondant à la zone de contenu (au centre) et aux barres latérales qui l'encadrent et qui contiennent les *portlets* ou « boîtes d'informations contextuelles ». Le système permet de réduire le nombre de colonnes à 2 ou même 1, en fonction de vos besoins.

L'identifiant CSS de ce bloc est `portal-columns`.

- La barre de bas de page (ou *footer bar*) – Elle contient des informations telles que le *Copyright*. Typiquement, si la place disponible le permet, on pourra y ajouter toute information ou lien complémentaire que l'on veut faire apparaître sur toutes les pages.

L'identifiant CSS de ce bloc est `portal-footer`.

- Le colophon – Il contient des détails sur les solutions utilisées pour mettre le site en production, typiquement la fameuse phrase « Plone Powered » ou « Conforme à XHTML ».

L'identifiant CSS de ce bloc est `portal-colophon`.

EXTRAIT DU GABARIT **Le bloc « portal-top »**

```
<div id="portal-top" i18n:domain="plone">
    <a href="#documentContent" class="hiddenStructure"
       i18n:translate="label_skiptocontent">Skip to content.</a>
    <a metal:use-macro="here/global_logo/macros/portal_logo">
        The portal logo, linked to the portal root
    </a>
    <div metal:use-macro="here/global_skinswitcher/macros/skin_tabs">
        The skin switcher tabs. Based on which role you have, you
        get a selection of skins that you can switch between.
    </div>
    <div metal:use-macro="here/global_siteactions/macros/
                          site_actions">
        Site-wide actions (Contact, Sitemap, Help, Style Switcher etc)
    </div>
    <div metal:use-macro="here/global_searchbox/macros/quick_search">
        The quicksearch box, normally placed at the top right
    </div>
    <div metal:use-macro="here/global_sections/macros/portal_tabs">
        The global sections tabs. (Welcome, News etc)
    </div>
    <div metal:use-macro="here/global_group/macros/portal_group">
        Portal group definition
    </div>
    <div metal:use-macro="here/global_personalbar/macros/
                          personal_bar">
        The personal bar. (log in, logout etc.)
    </div>
    <div metal:use-macro="here/global_pathbar/macros/path_bar">
        The breadcrumb navigation ("you are here")
    </div>
</div>
```

Pour la personnalisation, vous devez considérer `main_template` comme une boîte noire qui fournit le support technique du gabarit des pages, et ne jamais la modifier (sauf cas ou besoin exceptionnel).

Il y a trois méthodes différentes pour changer un aspect du gabarit, la structure ou le comportement visuel des pages :

- utiliser des propriétés bien placées telles que `right_slots` et `left_slots` (aucune ligne de code à écrire) ;
- utiliser CSS pour influencer le rendu d'un *template* appelé par `main_template` ;
- modifier le code ZPT/HTML d'un *template* appelé par `main_template`.

Il est recommandé de tenter de respecter cet ordre lorsque vous recherchez comment modifier le comportement d'un bloc du site.

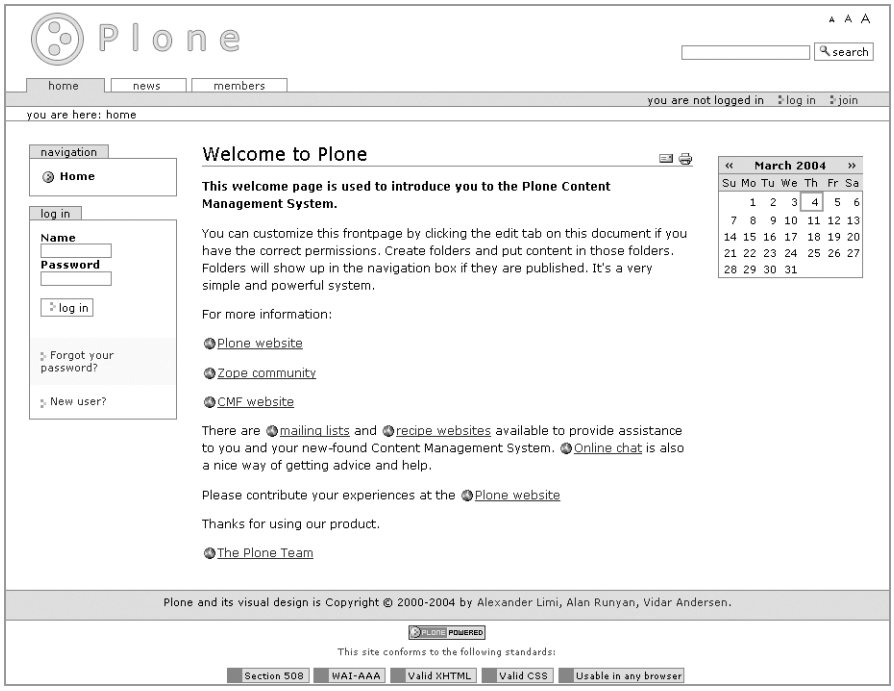


Figure 5–1
Le gabarit, appliqué à la page d'accueil

Exemples de personnalisation

Supprimer les colonnes latérales

Vous pourriez décider de modifier la structure du gabarit sur une page donnée, par exemple pour n'avoir que la colonne centrale. Par exemple, dans le cas de la page d'accueil, ce choix peut constituer une variation sur la page d'accueil classique d'un site Plone.

Une page d'accueil à une seule colonne

Pour changer le comportement de la page d'accueil afin que les colonnes latérales ne s'y affichent pas, il suffit de modifier l'objet `index_html` qui est utilisé pour cette page, en lui ajoutant les deux propriétés `right_slots` et `left_slots` de type `lines` vides (ce qui correspond à une liste vide). Mais comment s'y prendre ? Il y a une astuce !

En tant qu'administrateur, vous pouvez accéder aux interfaces de gestion technique des objets. Dans ce cas de figure, vous pouvez – même si l'action n'est pas exposée dans l'interface de Plone – accéder au formulaire de la ZMI (*Zope Management Interface*), qui permet de modifier les propriétés de l'objet `index_html` (un objet Zope comme les autres). Il suffit de visiter l'URL http://intranet/index_html/manage_propertiesForm, puis d'ajouter chacune des propriétés en question.

IMPORTANT

Les propriétés `right_slots` et `left_slots` doivent être de type `lines`.

RAPPEL « Inline CSS »

Plone a prévu cette façon de faire et a réservé à cet effet, dans toutes les *templates*, une zone nommée `css_slot` dans laquelle on peut ajouter des règles CSS. Les règles ainsi ajoutées surchargent celles des principaux fichiers CSS (`plone.css` et `ploneCustom.css`).

Des templates ne présentant pas de colonne latérale

Vous pouvez personnaliser un *template* spécifique en utilisant des déclarations CSS en ligne.

Prenons une page de formulaire telle que `login_form` ou `join_form`, qui se trouve dans la *sous-skin*`plone_forms`, et dans laquelle nous ne voulons plus afficher les colonnes de droite et de gauche. Il existe deux définitions dans le CSS qui nous permettent de cacher les colonnes: `portal-column-one` et `portal-column-two`.

Donc pour personnaliser ce *template*, dans la *sous-skin*`custom`, ajoutez le code suivant après la balise ouvrante `body` :

```
<tal:block metal:fill-slot="css_slot">
  <style>
     #portal-column-one {display: none;}
     #portal-column-two {display: none;}
  </style>
</tal:block>
```

Dès le rechargement de la page, vous pouvez vérifier que les visiteurs ne voient plus les fameuses colonnes latérales.

Modifier le contenu du bas de page

Vous pouvez redéfinir le contenu de la zone de bas de page, en personnalisant le *template* `colophon` contenu dans `plone_templates`. Remplacez, dans la version de personnalisation (dans `custom`), le code par défaut de Plone…

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en-US" lang="en-US"
      i18n:domain="plone">
<body>
<div id="portal-footer" metal:define-macro="portal_footer"
     i18n:domain="plone">
<span i18n:translate="description_copyright" tal:omit-tag="">
Plone and its visual design is Copyright &copy; 2000-
<span i18n:name="current_year"
      tal:define="now modules/DateTime/DateTime"
      tal:content="now/year" />
by

<span i18n:name="limi">
<a href="http://www.plonesolutions.com">Alexander Limi</a></span>,
<span i18n:name="runyaga">
<a href="http://www.runyaga.com">Alan Runyan</a></span>,
<span i18n:name="blacktar">
<a href="http://blacktar.com">Vidar Andersen</a></span>.
</span>
</div>
</body>
</html>
```

… par votre propre code, par exemple :

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en-US" lang="en-US"
      i18n:domain="plone">
<body>
<div id="portal-footer" metal:define-macro="portal_footer"
     i18n:domain="plone">
  Copyright &copy; 2003 –
  <span tal:define="now modules/DateTime/DateTime"
        tal:content="now/year"/>
  by Société Géniale
</div>
</body>
</html>
```

Voilà ! Ainsi, vous obtenez un gabarit dont le bas de page est personnalisé pour votre entreprise.

La charte graphique

Le logo de l'entreprise

Nous allons maintenant remplacer le logo de Plone, qui se trouve dans `plone_images`, par notre propre logo. La *sous-skin* `custom` ayant précédence sur la *sous-skin* `plone_images`, si vous ajoutez dans `custom` une image portant le même nom que le logo de Plone, c'est-à-dire `logo.jpg`, c'est cette nouvelle image qui sera affichée au sein du gabarit du site.

RAPPEL Charte graphique

C'est l'ensemble des éléments et des concepts qui participent à définir l'identité visuelle du site :

- couleurs,
- fond de page,
- typographie (polices),
- éléments graphiques : logo, icônes, boutons.

Elle définit également les règles de composition des pages. Elle doit être bien étudiée afin de respecter les exigences d'ergonomie et d'accessibilité.

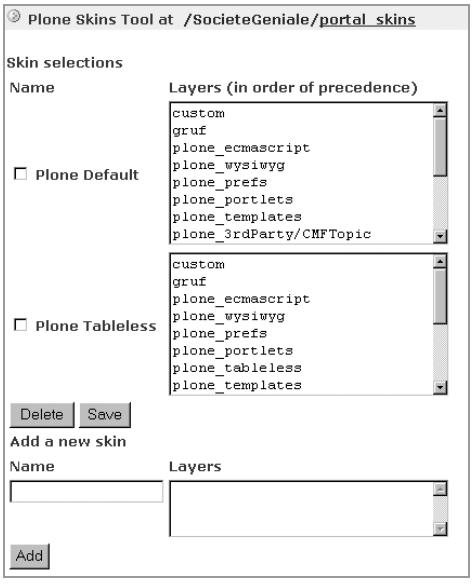


Figure 5–2 Paramétrage des sous-skins

Figure 5–3 Ajout d'une image

Pour commencer, ouvrez le dossier `custom` et ajoutez-y un objet de type `Image` que vous nommez `logo.jpg`. Vous devez télécharger vers le serveur le contenu du fichier image provenant du système de fichiers de votre poste.

Rechargez ensuite la page d'accueil du site pour vérifier que le nouveau logo est affiché.

PLUS DE TECHNIQUE **Avec CSS**

Vous pouvez avoir plus de contrôle sur l'image du logo en utilisant CSS. Par exemple, nous souhaitons appeler notre logo `societegeniale.jpg`.

La première étape consiste à renommer notre `logo.jpg`, déjà situé dans la *layer*`custom`, en `societegeniale.jpg`.

En regardant le code HTML de la page d'accueil de Plone, vous remarquerez que le logo `logo.jpg` n'est pas indiqué.

```
<h1 id="portal-logo">
    <a href="http://localhost:8080/Plone">Portal</a>
</h1>
```

Il va ensuite falloir que vous recherchiez la déclaration `portal-logo`, qui permet d'afficher le logo. La partie de `plone.css` qui est spécifique pour l'affichage du logo est la suivante :

```
/* Logo properties */
#portal-logo {
    background: url(&dtml-portal_url;/&dtml-logoName;) no-repeat;
    border: 0;
    margin: 0.75em 0em 0.75em 1.5em;
    padding: 0;
}
```

Allez maintenant dans l'objet `base_properties` (qui se trouve probablement déjà dans `custom`) et changez la valeur de la propriété `logoName` en `societegeniale.jpg`. Vous remarquez alors que notre logo est toujours affiché sur la page d'accueil de Plone, mais qu'il s'appelle `societegeniale.jpg`.

Autres exemples de personnalisation

Changer les couleurs

Les couleurs de Plone sont définies, en même temps que certaines autres caractéristiques, dans une page de propriétés nommée `base_properties`, que l'on trouve dans la *layer* `plone_styles`.

Pour changer ces propriétés, cliquez sur l'objet `base_properties`, sélectionnez `custom` comme destination et validez en cliquant sur `Customize`. Vous venez donc faire une copie exacte de `base_properties` dans le dossier `custom`, ce qui va vous permettre de l'éditer.

Nous allons maintenant changer la couleur des liens en modifiant la propriété `linkColor`.

ZOOM **Arrêt sur le fichier plone.css**

Les liens sont définis dans `plone.css` par la déclaration suivante :

```
a {
   text-decoration : none;
   color : &dtml-linkColor;;
   background-color: transparent;
}
```

Dans cette déclaration, c'est la commande DTML `&dtml-linkColor;` qui permet d'insérer la valeur de la propriété `linkColor`. C'est la seule commande DTML que vous deviez connaître pour modifier `ploneCustom.css`.

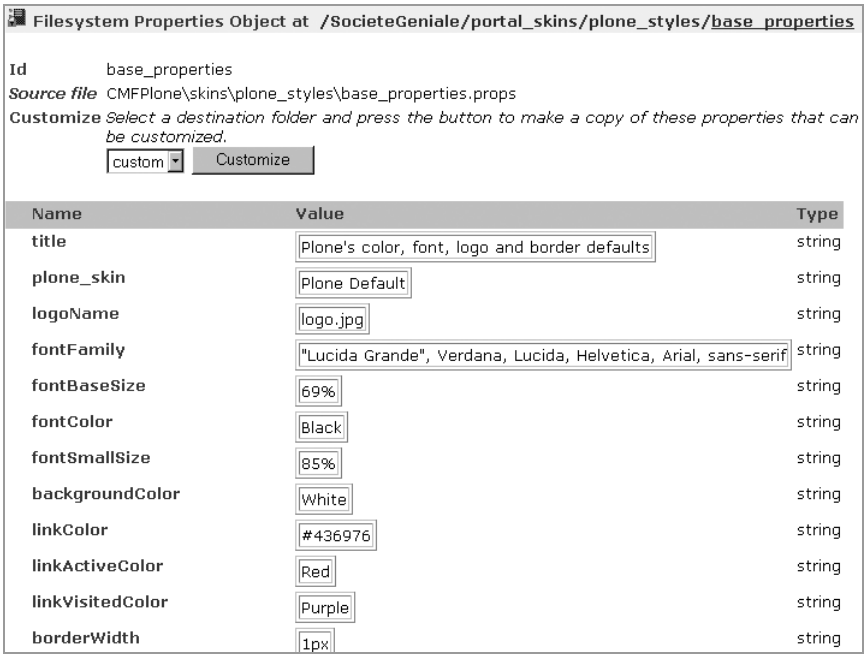


Figure 5–4
L'objet `base_properties`

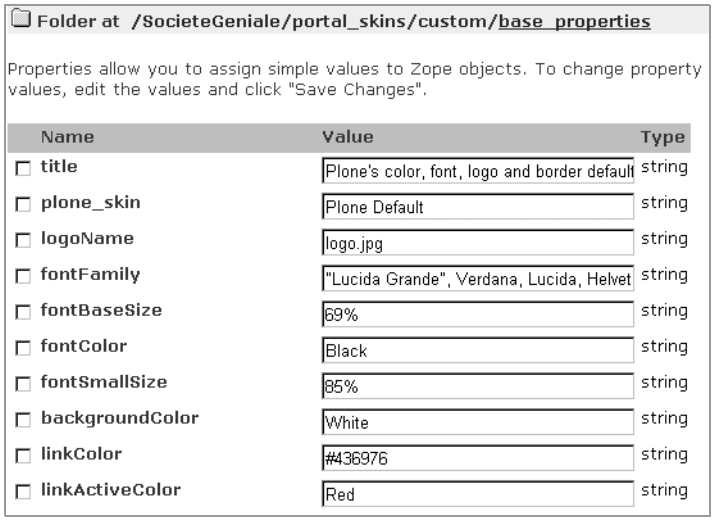


Figure 5–5
Personnalisation de l'objet `base_properties`

Pour obtenir une couleur plus vive, remplacez la valeur par défaut de `linkColor` (`#436976`) par `#3399CC`, dans l'objet `base_properties`.

Changer les polices de caractères

Vous pouvez facilement personnaliser les polices de caractères, les couleurs, le logo et d'autres éléments CSS dans une page de propriétés (*Property Sheet*) utilisée par la CSS de Plone.

Pour changer les polices de caractères utilisées par Plone, il suffit de modifier les propriétés `fontFamily` et `headingFontFamily`.

La propriété `fontFamily` est utilisée dans la déclaration CSS du `body` :

```
body {
    font: &dtml-fontBaseSize; &dtml-fontFamily ;;
    background-color : &dtml-backgroundColor;;
    color: &dtml-fontColor;;
    margin : 0 ;
    padding: 0;
}
```

La propriété `headingFontFamily` est utilisée pour les en-têtes des pages du site Plone :

```
h1, h2, h3, h4, h5, h6 {
    font-family: &dtml-headingFontFamily;;
    color: &dtml-fontColor;;
    background-color : transparent;
    font-size: &dtml-headingFontBaseSize;;
    font-weight: normal;
    margin : 0 ;
    padding-top: 0.5em;
    border-bottom: &dtml-borderWidth; &dtml-borderStyle; &dtml-
globalBorderColor;;
}
```

Afin d'obtenir une police de caractères personnalisée, nous allons remplacer la valeur des propriétés `fontFamily` et `headingFontFamily` par la valeur `Arial, Geneva, Helvetica, Verdana, Sans-Serif;`.

Vous pouvez alors constater que tous les textes et les en-têtes des pages de Plone sont affichés avec notre propre famille de polices de caractères.

La page d'accueil

Par défaut

À la création initiale d'un site Plone, un document (c'est-à-dire un contenu de type `Document`) est créé à la racine pour fournir le contenu de la page d'accueil. Ce document est nommé `index_html`, ce qui est nécessaire au fonctionnement de cette configuration.

Ainsi, il vous suffit de modifier ce document pour y ajouter votre texte d'accueil personnalisé. Mais vous aurez une page d'accueil assez pauvre en termes de services ! En effet, en tant qu'objet de type `Document`, celle-ci ne peut recevoir que du texte « statique ».

Enrichir la page

Vous avez la possibilité d'enrichir cette page avec du contenu dynamique – après tout, un serveur d'application, ça sert à ça – que vous injectez dans une zone bien choisie, par exemple sous le texte d'accueil.

Pour obtenir cette nouvelle page, vous devez personnaliser le *template*`document_view` responsable d'afficher le contenu d'un objet de type `Document`. Ce *template* se trouve dans la *sous-skin*`plone_content`. Visitez-là, puis en utilisant le bouton `Customize` exposé dans l'interface, créez la nouvelle version dans laquelle vous ajouterez, au bon endroit, le bloc de code suivant :

Pour aller au bout…

Il ne restera plus qu'à y insérer du code capable d'injecter le contenu dynamique, typiquement, du code de requête sur le `portal_catalog`. Voir le chapitre suivant pour les différentes techniques pour produire du contenu dynamique au sein d'un *template*.

```
<div tal:omit-tag=""
     tal:condition="python:container.meta_type=='Plone Site'
                    and here.getId() == 'index_html'">
    <h1> Nouveautés de l'intranet </h1>
    <!-- Zone réservée au code du contenu dynamique -->
</div>
```

Il faudra également faire quelques modifications mineures ou d'ordre « cosmétique » dans le code initial pour obtenir la bonne présentation pour votre page d'accueil. Le code complet résultant de ces différentes modifications devrait ressembler à ce qui suit :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      lang="en"
      metal:use-macro="here/main_template/macros/master"
      i18n:domain="plone">
<body>
<metal:main fill-slot="main">
<tal:main-macro metal:define-macro="main"
        tal:define="len_text python:len(here.text);">
    <h1 tal:content="here/title_or_id" class="documentFirstHeading">
      Title or id
    </h1>
  <div metal:use-macro="here/document_actions/macros/
                    ➥ document_actions">
           Document actions (print, sendto etc)
  </div>
  <div class="documentDescription"
            tal:content="here/Description">
           description
  </div>
  <p tal:condition="python: not len_text and is_editable"
          i18n:translate="no_body_text"
          class="discreet">
           This item does not have any body text, click the edit tab
           to change it.
  </p>
```

Le titre du document ou de la page.

La description du document. Peut être utilisée pour la phrase qui introduit le site intranet.

La gestion de l'affichage du contenu (statique) du document.

```
<div class="stx"
  tal:condition="len_text"
  tal:attributes="class python:test(here.text_format==
                  ➥'structured-text', 'stx', 'plain')">
      <div tal:replace="structure
                  ➥ python:here.CookedBody(stx_level=2)" />
</div>
<div tal:omit-tag=""
     tal:condition="python:container.meta_type=='Plone Site'
                    and here.getId() == 'index_html'">
  <h1> Nouveautés de l'intranet </h1>
  <!-- Zone réservée au code du contenu dynamique -->
</div>
```

Enfin, le bloc qui va fournir du contenu dynamique dans la page d'accueil.

```
<div tal:omit-tag=""
   tal:condition="not: python:container.meta_type=='Plone Site'
                        and here.getId() == 'index_html'">
   <div metal:use-macro="here/document_byline/macros/byline" >
        Get the byline - contains details about author and
        modification date.
   </div>
</div>
```

Nous ajoutons ici un traitement conditionnel avec le tal :condition pour que l'élément « byline » ne soit pas pris en compte dans le cas de la page d'accueil.

```
</tal:main-macro>
</metal:main>
</body>
</html>
```

AUTRES ÉLÉMENTS **Les actions**

Les actions sont des objets de paramétrage qui permettent d'ajouter des éléments d'interaction (liens, boutons, onglets, actions « javascript », etc.) à l'interface utilisateur.

Le composant `portal_actions` fournit un service de gestion des actions afin de faciliter la personnalisation du site et sa maintenance. Les actions sont paramétrables soit au niveau de chaque type de contenu, soit au niveau d'un composant qui respecte le contrat d'`Action Provider` vis-à-vis de `portal_actions`, soit au sein de `portal_actions` lui-même.

En résumé…

Maintenant que votre site est à vos goûts et à vos couleurs, il reste le plus important : personnaliser son fonctionnement, pour qu'il fonctionne vraiment comme vous le souhaitez et que les utilisateurs de l'intranet puissent enfin y participer ! C'est précisément l'objet des prochains chapitres…

Les workflows

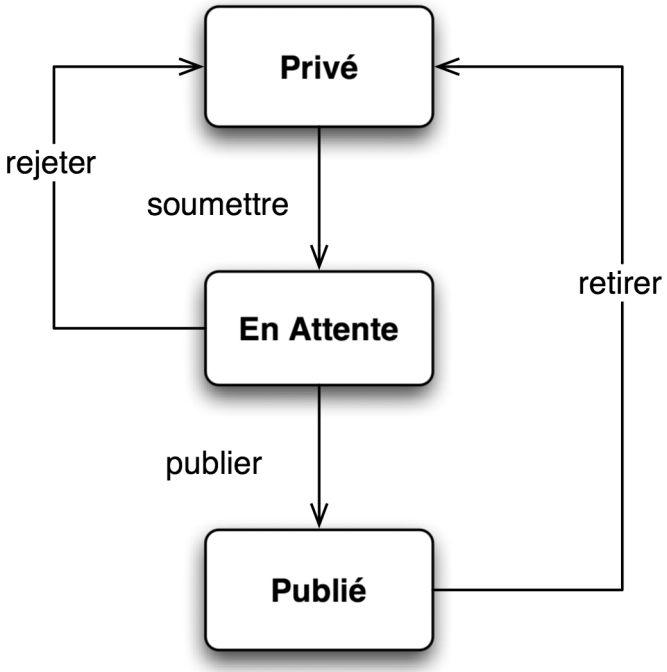
6

Zope Plone

SOMMAIRE

MOTS-CLÉS

- *Workflow*
- Sécurité
- Publication



Il est possible d'aller beaucoup plus loin dans la configuration que ce que nous avons vu précédemment – et c'est là toute la force de Plone par rapport à d'autres outils de gestion de contenu. Nous allons présenter maintenant comment configurer très précisément les règles de publication des documents.

Introduction aux workflows

Comme nous l'avons expliqué dans les précédents chapitres, un utilisateur authentifié a la possibilité de placer du contenu dans son espace personnel. Pour ce qui est de la publication, il doit s'en remettre à un responsable (appelé relecteur) qui va décider du bien-fondé ou non de sa demande. Ces règles, qui jusqu'à présent nous ont paru quelque peu empiriques, sont régies par un *workflow*. Avant de proposer une définition « scientifique » de la chose, voyons ce qu'est, au juste, un *workflow*.

Les états

Jusqu'à présent, vous avez pu constater qu'un document du site pouvait être dans un « état » particulier :

- `privé` (ou *private*) ;
- `visible` (identique en anglais) ;
- `en attente` (ou *pending*) ;
- `public` (identique en anglais).

Ces états sont mutuellement exclusifs : un document est dans un et un seul état à la fois, mais peut prendre plusieurs états au cours de son existence.

Intuitivement, on peut dire que chaque état définit « qui peut faire quoi » sur le document. On peut ainsi imaginer une matrice qui résumerait la fonction précise de chaque état :

Tableau 6–1 Tableau des permissions affectées à chaque état

État	Qui peut lire ?	Qui peut écrire ?
`Privé`	Le propriétaire du document L'administrateur du site	Le propriétaire du document L'administrateur du site
`Visible`	Tout le monde	Le propriétaire du document L'administrateur du site
`En attente`	Tout le monde	Les relecteurs concernés L'administrateur du site
`Public`	Tout le monde	L'administrateur du site

Par défaut, un document nouvellement créé par un contributeur est à l'état `visible`. Nous verrons plus loin dans ce chapitre qu'il est possible de changer l'état par défaut d'un *workflow*.

Ce tableau permet d'avoir une vision plus claire des droits affectés à chaque état d'un document.

Un état est donc avant tout une expression des restrictions de sécurité s'appliquant à un document. Changer un document d'état permet de lui affecter d'autres règles de sécurité : par exemple, passer de l'état `privé` à l'état `visible` permet aux utilisateurs anonymes du site de visualiser le document – ce qu'ils n'avaient pas le droit de faire lorsque celui-ci était à l'état `privé`.

La sécurité dans Zope

Avant d'aller plus loin dans notre étude des *workflows*, il est indispensable de poser les bases de la sécurité sous Zope.

Les principes de base

Nous avons découvert, dans les chapitres précédents, les notions de « rôles ». Dans Zope (ces notions ne sont pas particulières à Plone), la sécurité est distribuée grâce à trois concepts interdépendants :

- utilisateurs ;
- rôles ;
- permissions.

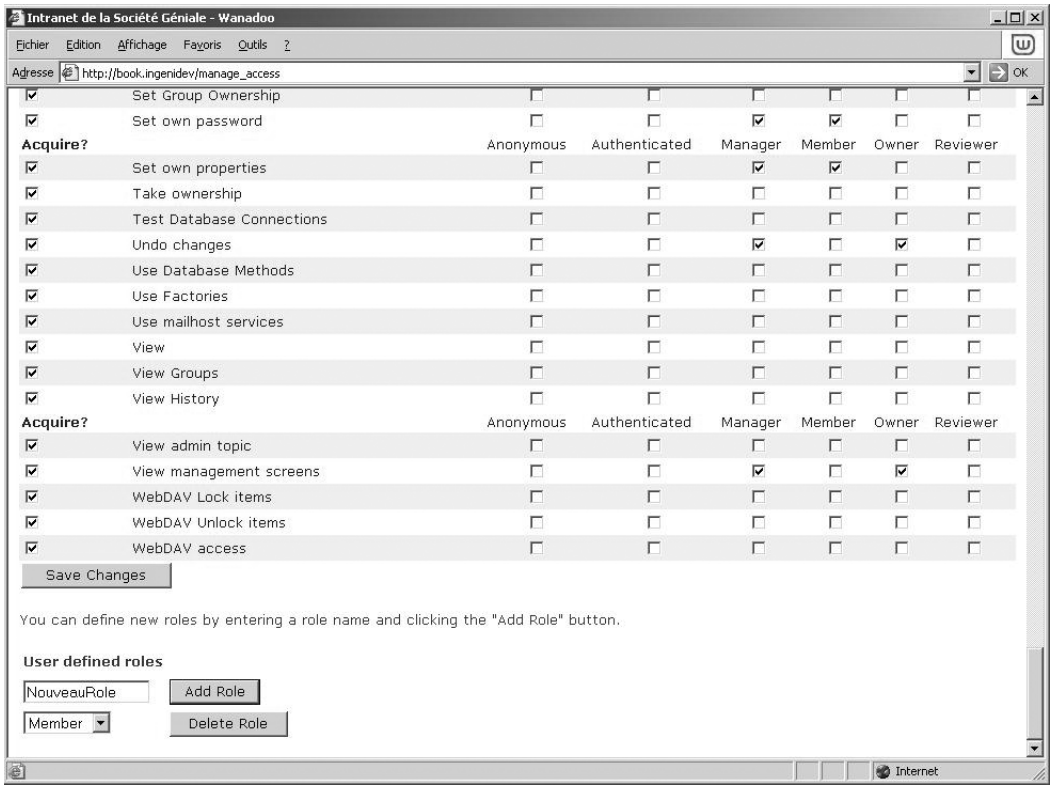


Figure 6–1 Notions d'utilisateur, de rôle et de permission

Comme nous le voyons, un utilisateur possède un certain nombre de rôles ; à chaque rôle est affecté une liste de permissions. Si un utilisateur possède un rôle, il aura donc toutes les permissions accordées à ce rôle. Si un utilisateur possède plusieurs rôles, toutes les permissions de chaque rôle lui seront accordées.

Les rôles

Dans Plone, nous rencontrerons essentiellement les rôles suivants :

- `Manager` – C'est le rôle de l'administrateur du site. Par défaut, un utilisateur doté du rôle `Manager` a le droit de « tout faire », un peu comme l'utilisateur `root` sur Unix. Pour éviter les dérives, il convient de n'attribuer ce rôle qu'avec parcimonie et discernement !
- `Reviewer` – Appelé en français `Relecteur` », il s'agit du rôle que possèdent les membres du site chargés de valider le contenu (notamment de valider le contenu placé à l'état `En attente`).
- `Member` – Rôle de base d'un membre d'un site Plone. Un utilisateur ne pourra pas contribuer correctement à un site Plone s'il n'est pas doté de ce rôle.
- `Owner` – Ce rôle est très particulier : il est attribué automatiquement à un contributeur lorsqu'il crée un objet, mais uniquement sur l'objet qu'il a créé, et non pas sur l'ensemble du site. Il s'agit d'un « rôle local », c'est-à-dire d'un rôle attribué à certains utilisateurs uniquement pour certains objets. Nous décrirons ce mécanisme plus loin dans cette section.

En dehors de ces quatre rôles par défaut, il vous est tout à fait possible de créer de nouveaux rôles. Pour cela, il convient de se rendre dans l'interface d'administration de Zope (ou la ZMI), via l'URL suivante : http://localhost:8080/manage. Puis, allez dans le Folder correspondant à votre site Plone et cliquez sur l'onglet Security. Tout en bas de la page (et quelle page !), vous trouverez une zone d'édition permettant de saisir le nom du nouveau rôle. Cliquez sur Add Role, et ce rôle est créé pour tout le site Plone. Inversement, le bouton Delete Role vous permet de supprimer les rôles que vous auriez déjà créés.

Nous verrons plus loin une autre notion, appelée « Rôle local », qui permet d'étendre ce modèle.

À RETENIR

Un rôle est créé à partir d'un niveau précis de l'arborescence de Zope, et est valable pour tous les sous-niveaux de cette arborescence. Autrement dit, si vous créez un rôle `NouveauRole` au niveau de votre site Plone, ce rôle n'existera pas dans la racine de votre site Zope.

Attention

Ne supprimez jamais les rôles par défaut de Plone ! Les rôles `Member` et `Reviewer` sont indispensables à son bon fonctionnement !

Les permissions

Sous Zope, une permission est, dans son acception la plus rigoureuse possible, une chaîne de caractères associée d'une part à des méthodes de classes Zope et d'autre part à des rôles. Une méthode ne peut être appelée par un utilisateur que s'il possède un rôle qui lui autorise la permission associée à la méthode.

Plus simplement, une permission définit un droit à une action. Par exemple, sous Plone, la consultation d'un document est soumise conjointement aux permissions `Access contents information` et `View`. La modification est soumise à la permission `Modify Portal Content`.

Utilisateurs et groupes

Les deux derniers concepts à aborder avant de se plonger dans la sécurité de notre intranet sont les utilisateurs et les groupes. Un utilisateur est une personne : rien de plus simple à imaginer !

À chaque utilisateur peuvent être affectés des rôles (mais jamais de permissions individuelles).

Sous Plone (grâce à l'outil `Group User Folder` appelé aussi `GRUF`), les groupes ne sont « que » des utilisateurs particuliers. Ils possèdent donc des rôles. Un utilisateur, dans l'interface de `GRUF`, peut être affecté à un (ou plusieurs) groupe(s), et ainsi « hériter » des rôles de ces groupes. De même, un groupe peut appartenir à un (ou plusieurs) groupe(s), permettant ainsi de produire des organisations d'utilisateurs très complètes.

Retroussez vos manches : il est temps maintenant de se plonger dans la création de ces groupes !

Comme nous l'avons vu plus haut, nous limitons notre analyse à trois populations :

- Marketing ;
- Service Après-Vente ;
- Direction Générale.

Une rapide analyse nous permet d'anticiper que chacune de ces populations va faire l'objet d'un… groupe, et que le groupe `Direction Générale` aura probablement quelques droits supplémentaires par rapport aux autres groupes…

Mais n'anticipons pas : si vous n'avez pas encore créé ces groupes, c'est le moment de le faire. Ouvrez l'interface de Plone en tant qu'administrateur et cliquez sur le lien Configurer Plone, puis sur Users and Groups administration. Un magnifique panneau récapitulatif apparaît, avec un onglet Users et un onglet Groups. Par défaut, l'onglet Users est ouvert, comme sur la page ci-après.

Cliquez sur l'onglet Groups puis sur le bouton Add new group, afin de créer les trois groupes dont nous aurons besoin :

- `marketing` ;
- `support` ;
- `dg`.

Il n'est pas nécessaire de remplir tous les champs (en particulier l'adresse électronique).

Puisque nous y sommes, il est également intéressant de créer quelques utilisateurs pour manipuler le site : cliquez sur le bouton Add new user de l'onglet Users, remplissez les champs du formulaire, et voilà ! Il ne nous reste plus qu'à affecter ces utilisateurs à des groupes.

Attention

Il est parfois tentant, dans certains projets, de « mutualiser » des comptes utilisateurs, c'est-à-dire de donner le même nom de compte et le même mot de passe à plusieurs personnes pour des raisons de simplicité. Ceci est à proscrire formellement en Plone, où le compte utilisateur revêt une grande importance !

Sous Plone, le rôle minimum d'un utilisateur est `Member`. Un utilisateur qui n'est pas doté de ce rôle ne pourra probablement pas utiliser Plone correctement.

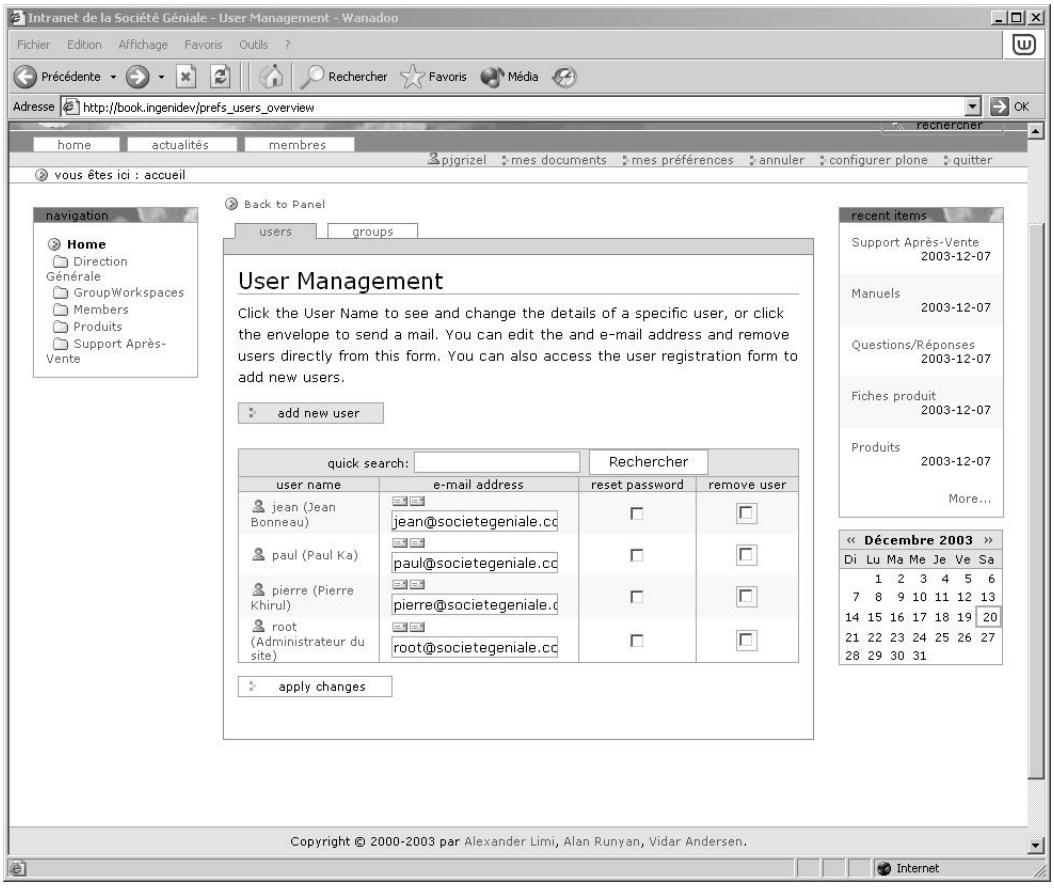


Figure 6–2 Gestion des utilisateurs dans Plone

Retournez à l'onglet Groups, cliquez sur un groupe de votre choix (par exemple dg), puis sur l'onglet group members de la page qui apparaît. Vous obtenez une interface présentée comme à la figure 6-3.

La première partie de la page est une interface de recherche, vous permettant de choisir les membres que vous voulez associer à ce groupe, tandis que la partie du bas vous permet éventuellement de supprimer ces utilisateurs des groupes. Affectez les utilisateurs que vous souhaitez aux groupes que vous souhaitez, rien de plus facile !

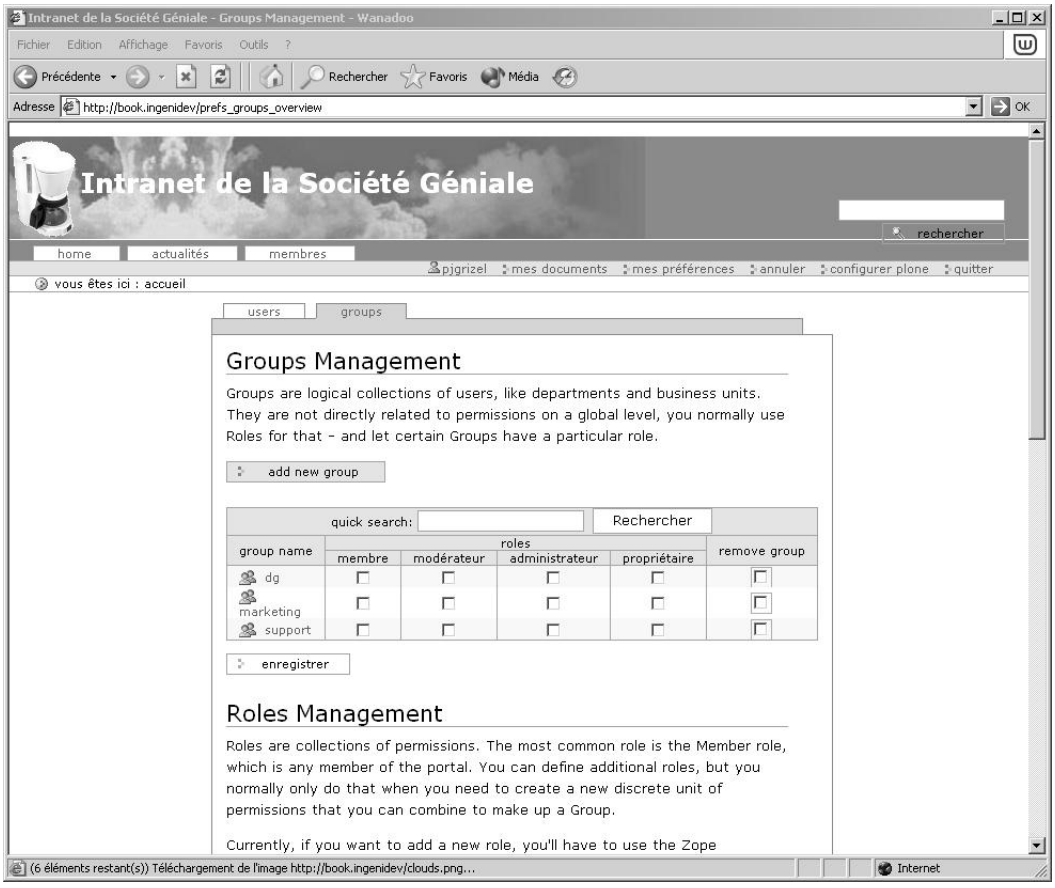


Figure 6–3 Gestion des groupes dans Plone

Workflows et type de contenus

Après cette incursion dans la sécurité sous Zope, voyons maintenant quel est le rapport entre cette sécurité et les *workflows*.

Sous Plone, donc, un contenu peut prendre plusieurs état. À chacun de ces états correspond un ensemble de permissions associées à un ensemble de rôles. Un état est donc avant tout un *snapshot* (une photographie) des permissions sur un contenu.

Chaque fois que l'on fait changer un document d'état, on lui applique les permissions associées à ce nouvel état.

La règle de base des *workflows* est qu'à un type de contenu est associé un et un seul *workflow*. Nous allons, dans les sections qui suivent, regarder à quoi servent les *workflows*, comment les créer et les modifier.

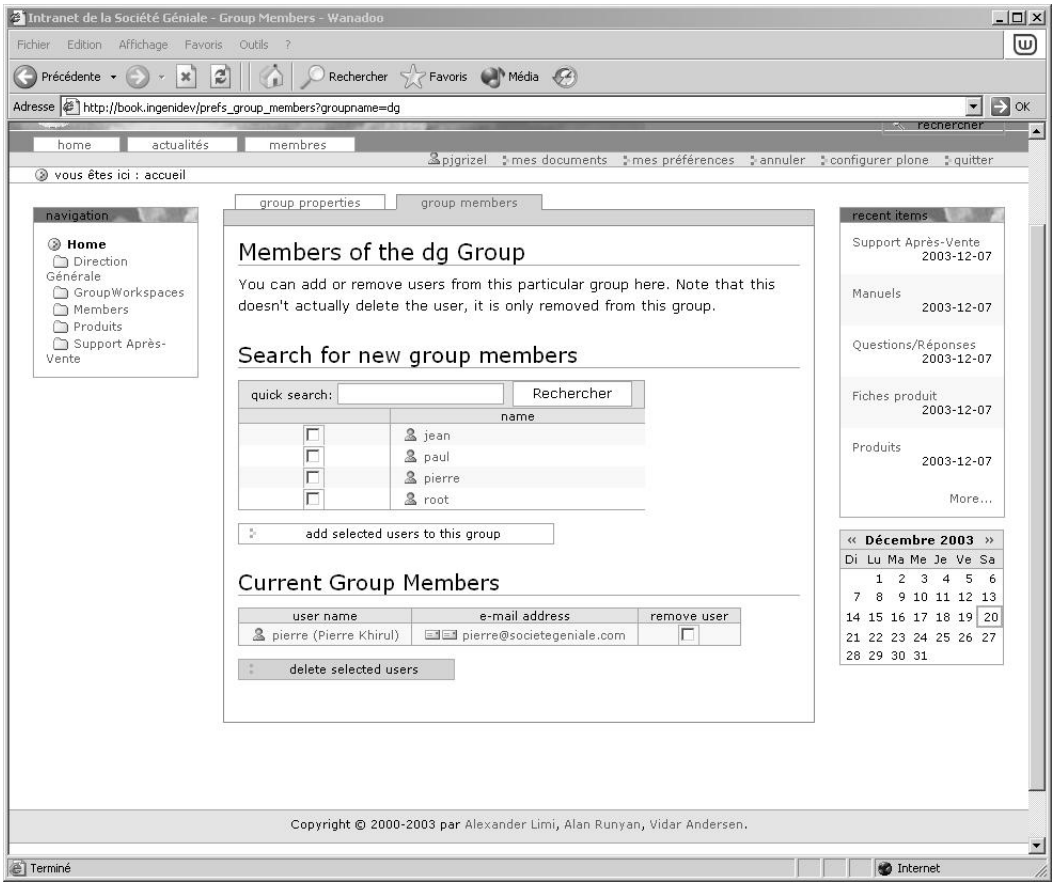


Figure 6–4 Affectation d'utilisateurs à des groupes

L'outil portal_workflow

Placez-vous dans l'interface d'administration de Zope, sur l'outil `portal_workflow`. Vous pouvez alors consulter l'interface suivante.

Associer un workflow à un type de contenu

En face de chaque type de contenu, on peut saisir le nom d'un *workflow* associé. Si la chaîne `(Default)` est utilisée pour un type de contenu, alors le *workflow* mentionné à la section `(default)` est utilisé. En standard dans Plone, il existe deux *workflows* :

- `folder_workflow` – Il s'agit d'un *workflow* particulièrement adapté aux dossiers, comme son nom l'indique. Il possède trois états : `visible`, `published` et `private`.

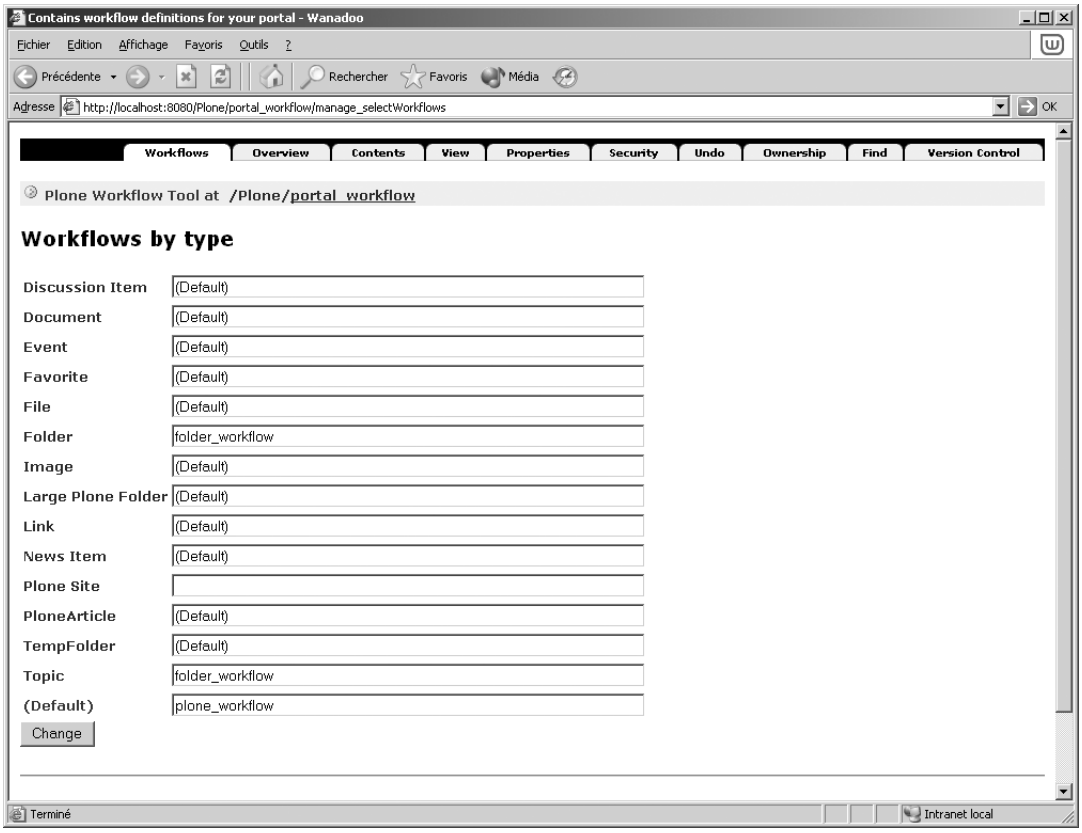


Figure 6–5 L'interface de sélection des workflows par type de contenu

- `plone_workflow` – Ce *workflow* et plus adapté aux types de contenu traditionnels et possède un niveau de modération. Ainsi, il expose les états `private`, `visible`, `pending` et `published`.

Il est possible, en remplaçant un champ par le nom d'un *workflow* valide, d'utiliser ce nouveau *workflow* pour un type de contenu donné. Par exemple, vous pouvez décider que les types de contenu `Image` respecteront désormais un *workflow* de type `folder_workflow` en lieu et place de `plone_workflow`.

Les dossiers restreints

Voyons maintenant le détail de chaque *worfklow*. Cliquez sur l'onglet Contents, puis sur le premier *workflow*. Plusieurs onglets apparaissent, en particulier States et Transitions, qui vous donnent un aperçu des états et transitions applicables pour ce *workflow*.

Cliquez sur l'onglet States. Les trois états du *workflow* apparaissent clairement. En face de l'état `visible`, une astérisque indique qu'il s'agit là de l'état par défaut, c'est-à-dire l'état que prend automatiquement un contenu à sa création.

Attention

Nous l'avons vu, un état d'un *workflow* est une sorte de photographie des permissions appliquées à des contenus. Si vous changez de *workflow* pour un type de contenu donné, il faut remettre à jour la sécurité sur le site ! De même, il se peut que certains contenus se trouvent dans des états qui n'existent plus dans le nouveau *workflow*. Il faut alors leur faire prendre un état « par défaut » et surtout en appliquer les permissions.
Pour réaliser cette opération, chaque fois que vous modifiez des *workflows*, cliquez sur le bouton Update security settings. Cette étape est indispensable pour appliquer vos modifications à tous les contenus déjà présents sur le site.

Attention

Lorsque l'on modifie un *workflow*, c'est généralement une mauvaise idée de supprimer des états. Il vaut mieux conserver un état et supprimer toutes les transactions qui mènent vers cet état.

Dans notre exemple de site, nous allons modifier le *workflow* pour prendre en compte un nouvel état, « restreint » (`restricted` pour conserver l'anglicisme des états déjà existants). Il ne s'agit pas d'une modification anodine et cela réclame un maximum de concentration ! Allons-y, étape par étape…

Pour résumer notre cas d'utilisation, nous voulons que toutes les rubriques soient visibles par défaut à tous les membres du site, mais que certaines rubriques soient visibles uniquement à certains membres, que nous devrions pouvoir définir rubrique par rubrique. Une fois qu'une règle d'accès est établie pour une rubrique, elle est valable pour toutes ses sous-rubriques. Nous souhaitons conserver les états `visible`, `private` et `published`. Il sera cependant nécessaire d'appliquer quelques modifications à ces trois états pour rendre notre système parfaitement opérationnel.

Attachons-nous en premier lieu à étudier la sécurité des états `published` et `visible`. Bien que leur sémantique soit différente, ces deux états doivent dans notre schéma être porteurs des mêmes permissions. De même, ils doivent être configurés de manière à ce que si un dossier est `visible` ou `published` au sein d'un dossier `restricted`, ces dossiers publiés « héritent » des restrictions de sécurité du dossier qui les contient, et soient donc restreints eux aussi. La notion d'acquisition va nous être fort utile !

Cliquez sur l'état `visible`. Une nouvelle page de propriétés apparaît, avec un onglet Permissions : c'est celui-ci qui nous intéresse ici.

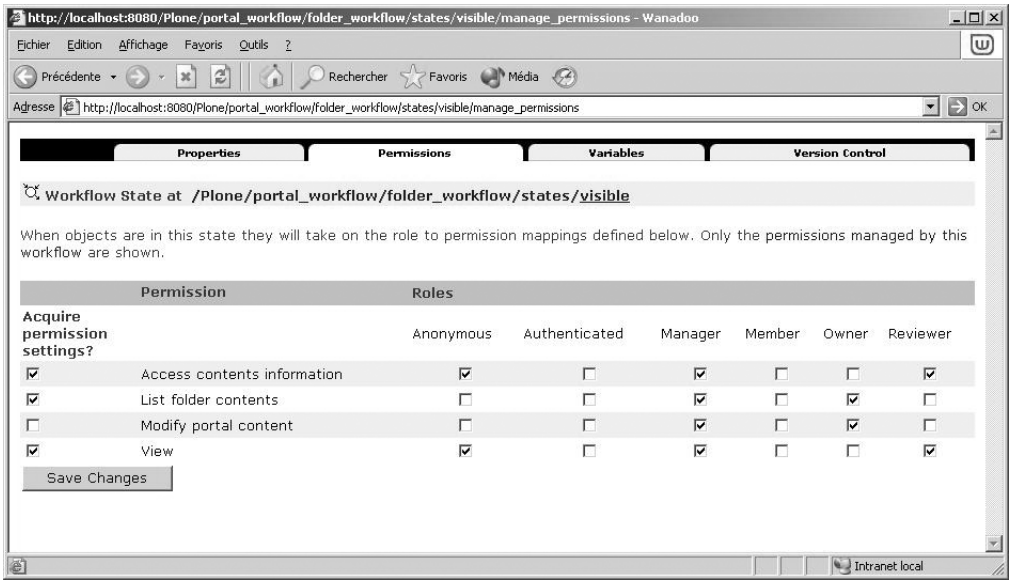


Figure 6–6
La page « Permissions » de l'état « Visible »

Tel qu'il est configuré par défaut dans Plone, cet état pose un petit souci : en effet, certaines permissions sont systématiquement accordées au rôle `Anonymous` (c'est-à-dire aux utilisateurs anonymes), quel que soit l'état du dossier qui les contient. Cela va à l'encontre de la règle que nous nous sommes fixée, qui stipule qu'un dossier publié dans un dossier restreint doit être restreint lui-aussi. Il faut donc ôter aux anonymes cette permission, et se baser uniquement sur l'acquisition des permissions pour autoriser l'accès à un dossier. Placez donc les permissions comme présentées dans le tableau suivant :

Acq. Perm. settings		Anonymous	Authenticated	Manager	Member	Owner	Reviewer
X	Access contents information			X			X
X	List folder contents			X		X	
X	Modify portal content			X		X	
X	View			X			X

Sauvegardez les permissions ainsi affectées. Allez sur l'état `published` et appliquez strictement le schéma précédent. Ici encore, il convient essentiellement de retirer la case `Anonymous`.

L'état `private` n'appelle pas de commentaire particulier. Nous ne nous y attarderons pas. Il convient maintenant de créer le fameux état `restricted`, mais… avant cela, une question subsiste. Comment indiquer à Plone quels sont les utilisateurs qui ont accès à un dossier restreint ? Le meilleur moyen (et probablement le seul !) consiste en l'utilisation de rôles locaux. Aucun rôle, par défaut, ne correspond vraiment à l'usage que nous souhaitons en faire. Aussi convient-il de créer un rôle à la racine du site. Placez-vous sur l'objet `Plone Site` et cliquez sur l'onglet **Security**. Tout en bas de la page, dans le formulaire, créez un rôle appelé `LocalViewer` et validez.

Retournez dans le *workflow* du dossier, sur la page **States**, et dans la zone **Add a state**, créez notre fameux état `restricted`. Avant de remplir les autres propriétés, placez-vous sur l'onglet **Security** de ce nouvel état et appliquez les permissions présentées dans le tableau ci-dessous. Vous noterez au passage l'apparition du rôle `LocalViewer` dans le tableau !

Les permissions de Plone

Usuellement, dans Plone, nous travaillons avec les permissions suivantes :

- `Access contents information` – Cette permission régit l'accès à de nombreuses propriétés du contenu, comme son titre, son auteur, sa date de dernière modification, etc.
- `List folder contents` – Cette permission s'applique à la méthode de listing des objets contenus dans un dossier.
- `Modify portal content` – Cette permission permet de modifier un contenu.
- `View` – Cette permission permet de voir le contenu « brut ». Elle n'est pas suffisante pour *afficher* un contenu complet dans une page : elle doit être associée à `Access contents information`.

Acq. Perm. settings		Anonymous	Authenticated	LocalViewer	Manager	Member	Owner	Reviewer
	Access contents information			X	X		X	X
	List folder contents			X	X		X	
X	Modify portal content				X		X	
	View			X	X		X	X

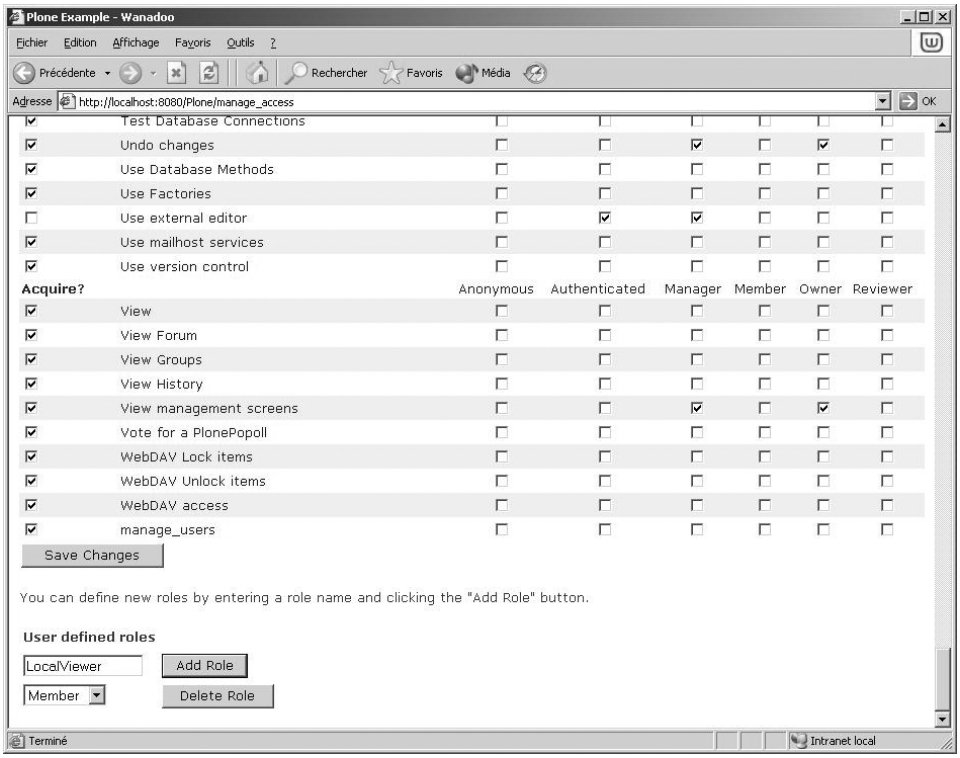


Figure 6–7
Création du rôle `LocalViewer`

Ici, toute l'astuce consiste à supprimer les cases `Acquire permission settings` pour les permissions que nous souhaitons réellement restreindre. Et le tour est joué ! Après avoir validé, cliquez sur l'onglet **Properties** de l'état, remplissez son titre et sa description avec un texte susceptible de vous rappeler, dans un an, à quoi sert cet état ! :-)

Ne quittez pas la page de propriétés : la zone **Possible transitions** nous rappelle qu'un état sans transitions est aussi utile qu'une bouteille sans tire-bouchon.

Afin de savoir quelles transitions permettre à partir de quel état, voici un petit récapitulatif :

- état `private` : `publish`, `show`, `restrict` ;
- état `published` : `hide`, `retract`, `restrict` ;
- état `restricted` : `show`, `hide`, `publish`, `retract` ;
- état `visible` : `hide`, `publish`, `restrict`.

Cochez donc les cases `show`, `hide` et `publish` pour les transitions possibles à partir de l'état `restricted`. Pour modifier les autres états, il faut avant tout créer la transition `restrict`, qui n'existe pas encore. Allez sur l'onglet **Transitions** du *workflow* et créez, en bas de la page, la nouvelle transition appelée `restrict`. Utilisez les informations fournies dans la figure suivante.

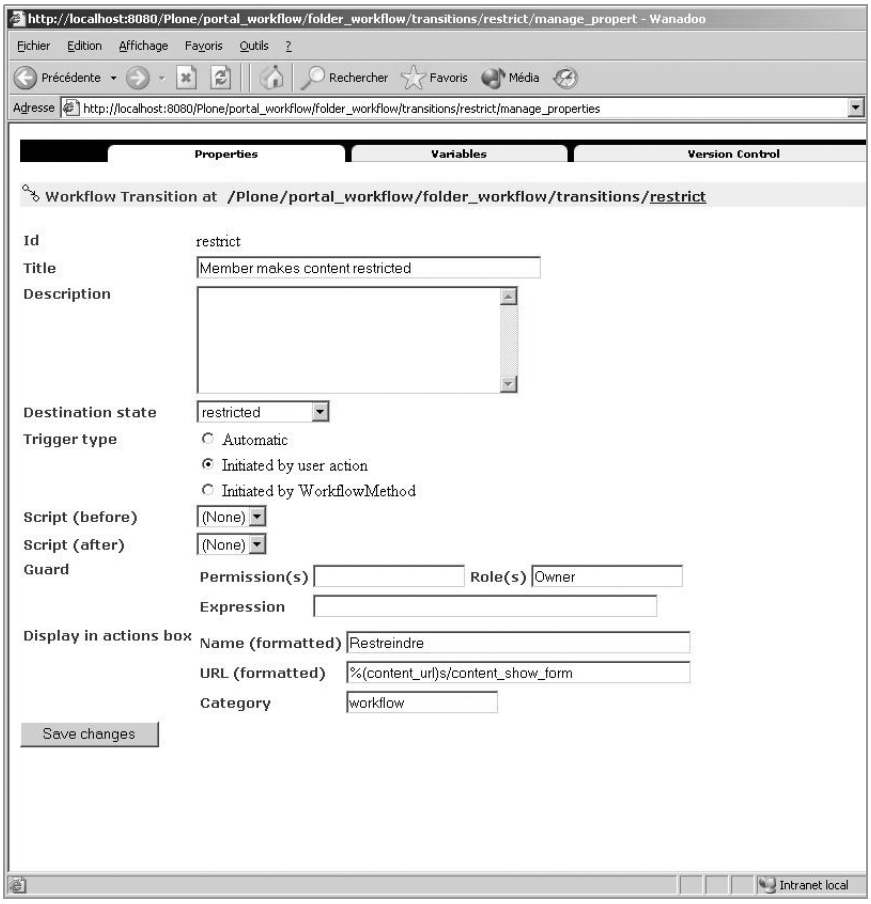


Figure 6–8
Les propriétés de la transition `restrict`

Voici une explication de chacun des champs :

- Id : l'identifiant de la transition ;
- Title : un titre pour la transition, qui n'apparaîtra pas à l'utilisateur final ;
- Description : une description succincte du rôle de l'action ;
- Destination state : état de destination de la transaction (`restricted` dans notre exemple) ;
- Trigger type : type de déclenchement de la transition ;
- Scripts : scripts Python qui peuvent être déclenchés avant ou après l'exécution de la transition (nous verrons plus loin dans ce chapitre leur grande utilité) ;
- Guard : permissions et/ou rôles et/ou expression TAL à vérifier pour que la transition soit proposée à un utilisateur (dans notre cas, il faut simplement qu'un utilisateur possède le rôle `owner` sur le dossier pour pouvoir appliquer la transition, mais nous pourrions imaginer des schémas plus complexes) ;

L'état de destination d'une transaction

Sous Plone, une transaction est définie par son état de destination et non pas son état d'origine. Ainsi, une même transaction peut être utilisée depuis plusieurs états mais ne peut mener qu'à un et un seul état.

- Display in actions box : titre de l'action, tel qu'il apparaîtra à l'utilisateur final. Sous Plone, le champ `URL` n'est pas utilisé, et le champ `Category` doit toujours avoir pour valeur `workflow`.

Et voilà ! Notre transition est créée. Ajoutez la transition `restrict` aux états `visible`, `private` et `published` de manière à pouvoir rendre un dossier restreint.

Il ne reste plus qu'à cliquer, comme nous l'avons vu ci-dessus, sur le bouton Update security settings de la page d'accueil de `portal_workflow`.

Retournez dans le site, créez un dossier et vérifiez que vous pouvez bien lui appliquer la transition `restrict` et que son comportement est conforme aux attentes. Félicitations ! Il ne reste plus qu'à nous attaquer au *workflow* du contenu…

Une modération à deux étages

Qui dit *workflow* dit modération ! En guise d'exercice, nous nous proposons de créer deux niveaux de modération pour le *workflow* du contenu.

Par défaut, sous Plone, un contenu est publié à partir du moment où il est validé par le relecteur (rôle `reviewer`). Nous allons ajouter un état intermédiaire entre la relecture et la publication, que nous appellerons « en attente de validation », ou, pour se conformer à l'utilisation de l'anglais dans la programmation, l'état `validation_pending`. Un contenu passe dans cet état après avoir été validé par le relecteur. Il ne sera effectivement publié qu'après validation par un membre doté du rôle `manager` sur ce contenu. Voici comment nous y prendre pour modifier le `plone_workflow`…

Il convient de créer un état `validation_pending` et la transition associée. En tout premier lieu, créez le nouvel état en allant dans les pages appropriés de `portal_workflow`. Associez-lui une description et un titre, et associez-lui les transitions suivantes :

- `publish` (bien sûr !) ;
- `reject` ;
- `retract`.

Dans l'onglet Security, appliquez la matrice suivante :

Acq. Perm. settings		Anonymous	Authenticated	LocalViewer	Manager	Member	Owner	Reviewer
	Access contents information				X		X	X
	Change portal events				X			
	Modify portal content				X			
	View				X		X	X

Nous autorisons le `Reviewer` à consulter le contenu, mais pas à le modifier.

Créez ensuite une transition appelée `review`, qui sera appelée par le rôle `reviewer` en lieu et place de la transition `publish`. Voici les informations à saisir :

Champ	Valeur
Id	`review`
Title	`Valider la relecture`
Description	`...`
Destination state	`validation_pending`
Trigger type	`Initiated by user action`
Script (before)	
Script (after)	
Guard Permission(s)	`Review portal content`
Guard Role(s)	
Guard Expression	
Display name	`Valider`
URL	
Category	`workflow`

Attention

Attention, certains champs sont tatillons quant au respect de la casse (majuscules/minuscules).

Il nous reste à « brancher » cet état en lieu et place du `publish` de l'état `pending`. La démarche est logique : placez-vous sur la page consacrée à l'état `pending`, et cochez la case `review` en plus de la case `publish`. Et pourquoi ne pas retirer la case `publish` ? Tout simplement parce qu'un individu doté à la fois des rôles `Reviewer` et `Manager` pourra, de la sorte, publier directement son contenu sans passer par la phase de validation. La vie est belle.

À ce propos, il ne reste plus qu'à empêcher le rôle `Reviewer` d'appeler la transition `publish` lui-même. Placez-vous sur la page consacrée à cette transition, et au lieu d'un `guard` sur la permission `Review portal content`, utilisez un `guard` sur le rôle `Manager`. C'est fait ? Félicitations, votre *workflow* peut alors être testé copieusement… Enfin presque.

Vous souvenez-vous de notre *workflow* consacré aux dossiers ? Nous avons méticuleusement retiré les cases `Anonymous` pour permettre à notre système basé sur le `LocalViewer` de fonctionner. Il convient de faire de même pour le *workflow* des types de contenu. Voici le positionnement de la sécurité à adopter pour les états qui le nécessitent.

D'abord, l'état `published` :

Acq. Perm. settings		Anonymous	Authenticated	LocalViewer	Manager	Member	Owner	Reviewer
X	Access contents information				X		X	
	Change portal events				X			
	Modify portal content				X			
X	View				X		X	

Ensuite, l'état `visible` :

Acq. Perm. settings		Anonymous	Authenticated	LocalViewer	Manager	Member	Owner	Reviewer
X	Access contents information				X		X	
	Change portal events				X		X	
	Modify portal content				X		X	
X	View				X		X	

Et voilà ! Tout est prêt… N'oubliez pas de cliquer sur le bouton Update security settings du *workflow* pour mettre la sécurité en ordre, et pensez bien à tester copieusement votre *workflow*, dans tous les sens, pour vérifier que vous n'avez rien oublié !

Il manque enfin la petite touche finale… Avez-vous remarqué qu'en tant que modérateur vous disposez d'une boîte qui vous indique les tâches en attente ? C'est toujours le cas pour l'état `pending`, mais il manque un moyen d'indiquer au `Manager` qu'il a des documents à valider. Placez-vous dans la zone Worklists du `plone_workflow`. C'est ici qu'est établie la liste des tâches, de manière dynamique. Autrement dit, il nous suffit de déclarer une nouvelle liste de tâches pour que celle-ci soit automatiquement affichée dans la boîte existant pour le `manager`.

Créez une *worklist* appelée `validation_queue`. Saisissez les informations suivantes :

Champ	Valeur
Id	`manager_queue`
Description	`Manager tasks`
Cataloged variable matches (formatted) review_state=	`validation_pending`
Display name	`Pending (%(count)d)`
URL	`%(portal_url)s/ search?review_state=validation_pending`

Le cas particulier de la racine du site

La racine d'un site Plone n'est soumise à aucun *workflow*. C'est très pratique dans la mesure où cela vous permet d'appliquer des règles de sécurité complexes à la totalité du site, sans devoir associer votre objet `Plone Site` à un *workflow* qui finirait de toutes façons par être un cas particulier…

Champ	Valeur
Category	`global`
Guard Permission(s)	
Guard Role(s)	`Manager`
Guard Expression	

Vous pourrez vérifier par vous-même que les tâches en attente sont maintenant indiquées automatiquement au `Manager`.

Encore plus d'automatisation !

Nous voilà heureux de disposer d'un *workflow* fort pratique pour les dossiers et le contenu. Mais nous pouvons aller plus loin… Par exemple, faire en sorte d'envoyer un courrier électronique au créateur d'un document à chaque fois que son état est modifié. Cela n'est pas très compliqué, voici la marche à suivre…

Envoi d'un message électronique à l'auteur du document

Placez-vous dans la zone script du `plone_workflow`. La page est vide. Nous allons créer un script Python capable d'identifier le propriétaire d'un document, d'aller glaner son adresse électronique (si elle est valide), et de lui envoyer un message pour lui indiquer que l'état du document a été changé.

Créez un script Python appelé `send_author_email`, prenant un paramètre `state_info` et contenant le code suivant :

```
transition_id = state_info.transition.id
user = container.portal_membership.getAuthenticatedMember().
                                          ➥ getUserName()



owner = container.portal_membership.getMemberById(
                                ➥ state_info.object.Creator)
if owner:
  owner_name = owner.getUserName()
  owner_email = owner.email
else:
  owner_name = "(Utilisateur inconnu)"
  owner_email = None
url = state_info.object.absolute_url()
title = state_info.object.Title()
```

Nous initialisons déjà moult variables en utilisant l'API de Plone. L'objet `state_info` contient un objet particulier et est transmis par le workflow lors de l'appel du script (nous verrons précisément comment un peu plus loin) :

Il faut prendre en compte le fait que l'`owner` (ou créateur) d'un document n'existe plus dans le portail. Il convient donc de se protéger du cas où sa valeur serait None.

```
if owner_email:
  container.MailHost.send(
"""From: Portal Manager <manager@monplone.org>
To: %s <%s>
Subject: L'état de votre document %s a changé
  Bonjour,
  Nous avons le plaisir de vous informer que l'état de votre document
  %s situé à l'adresse %s a été changé par l'utilisateur %s.
  Cordialement,
    -- Le webmaster
""" % (owner_name, owner_email, title, title, url, user))
```

Nous n'envoyons le message que si l'adresse du créateur est valide. Cela paraît logique, mais un test de plus dans le code vaut mieux qu'un bogue de plus dans le site ! :-)
Nous utilisons les services du composant `MailHost` pour envoyer le message, sous forme d'une chaîne de caractères relativement brute et rudimentaire (le lecteur assidu, curieux, volontaire et courageux pourra implémenter un formulaire de messagerie plus sympathique en ZPT).

Jusque là, tout va bien. Il ne nous reste plus à indiquer à notre *workflow* qu'il doit exécuter ce script à chaque passage d'une transition à une autre.

Pour ce faire, placez-vous sur chacune des transitions du *workflow*, et sélectionnez le script `send_author_email` pour l'option `Script (after)`. Testez votre *workflow* : si votre serveur SMTP est valide et que les adresses électroniques de vos utilisateurs sont renseignées correctement, vous devriez alors avoir la joie d'envoyer un courrier à l'auteur d'un contenu chaque fois que vous changez l'état de ce dernier !

En résumé…

Le *workflow* est un outil particulièrement puissant s'il est bien utilisé et maîtrisé. Les possibilités sont infinies et les scripts de transitions vous permettent d'obtenir des facultés d'automatisation très faciles à réaliser avec Plone – alors qu'elles seraient cauchemardesques avec un autre système.

Libre au lecteur curieux d'aller plus loin dans l'exploration du *workflow*… nul doute que cet outil répondra avec bonheur aux plus exigeants !

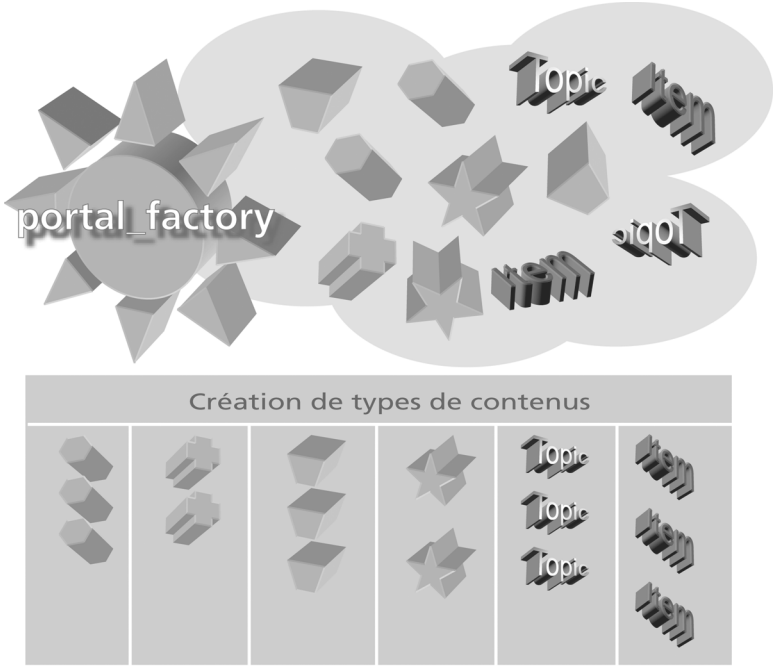
7 Types de contenu

Zope
Plone

SOMMAIRE

MOTS-CLÉS

- Type de contenu
- Archetypes
- ZClasses



Dans les chapitres précédents, nous avons étudié comment adapter Plone graphiquement et au travers de mécanismes de gestion de publication tel que les workflows. Mais la puissance de Plone ne s'arrête pas là : il est en effet possible de créer de nouveaux types de contenu en définissant les données qui seront portées ainsi que le comportement de ce contenu. C'est là l'objet de ce chapitre.

Gestion des types de contenu

L'outil portal_types

Sous Plone, les types de contenu sont gérés à travers l'outil `portal_types`. Si, dans l'interface d'administration de Zope, vous cliquez sur ce composant, vous y verrez la liste des types de contenu disponibles pour le portail.

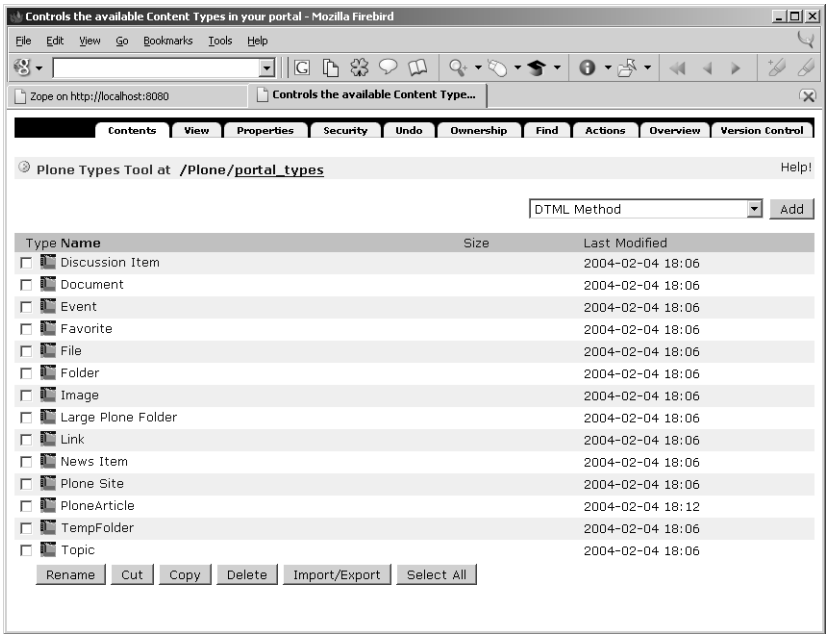


Figure 7–1 Un premier aperçu de l'outil portal_types

Certains de ces types de contenu vous sont déjà familiers, d'autres peuvent vous paraître étranges (`Discussion Item` ou `Topic`, par exemple). Cliquez sur le type `Document` pour consulter le détail de ses propriétés dans la page qui s'affiche (figure 7-2).

Vous pouvez, grâce à cette page, modifier certaines des informations liées à ce type de contenu, comme son titre (le nom qui apparaît dans la boîte déroulante d'ajout de contenu du site), sa description, etc. Voici ce que vous pouvez modifier :

- Title – Il s'agit bien entendu du titre du type de contenu.
- Description – Description du type de contenu, apparaissant dans certaines pages de Plone.
- Icon – Nom de l'icône (devant être dans la *skin*) utilisée pour représenter un objet de ce type. Il s'agit de la modification la plus simple et la plus parlante à effectuer sur un type de contenu !

B.A.-BA Qu'est-ce qu'un type de contenu

C'est le support principal de l'information, sous Plone. Un type de contenu est une classe structurée, possédant des propriétés et des méthodes, intégrée à un portail Plone. Un type de contenu dérive de certaines classes de base ; ainsi, il présente un ensemble de méthodes et de propriétés standards permettant une parfaite intégration à Plone. Par exemple, certaines méthodes permettent d'interfacer le type de contenu au catalogue, d'autres méthodes permettent de définir les onglets qui apparaissent dans l'interface d'édition. Nous verrons qu'en plus des méthodes et des propriétés, les types de contenus possèdent des vues et des actions.

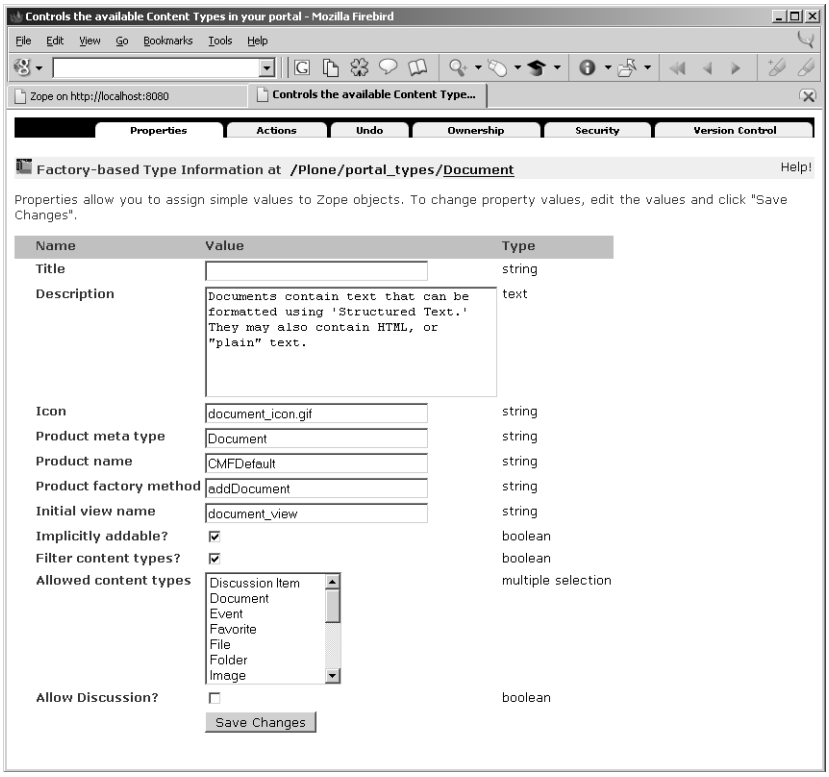


Figure 7–2 Le type de contenu Document

- Product meta type, Product name et Factory method – Nous décrirons ces informations ultérieurement.
- Initial view name – Nom de la page (depuis la *skin*) qui apparaît lorsqu'un nouvel élément de ce type est créé. Ici, il s'agit de `document_view`, mais nous aurions pu mettre `document_edit` pour indiquer que lors de la création d'un objet de ce type, on se place directement sur l'onglet Edit.
- Implicitly addable – Si cette case est cochée, alors un membre peut lui-même ajouter un objet de ce type. Si elle n'est pas cochée, il n'est possible de créer des objets de ce type que par programmation. Retirer la coche de cette case est sans conteste le moyen le plus simple et le plus rapide pour inhiber un type de contenu.
- Filter content types et Allowed content types : pour les objets conteneurs (`Folder`, par exemple), ces deux options permettent de spécifier les types de contenu autorisés en tant que contenu. Ces options sont très rarement utilisées.
- Allow discussion – Si cette case est cochée et si la discussion est autorisée sur le site, alors le type de contenu pourra être commenté. Dans le cas contraire, il ne pourra pas l'être.

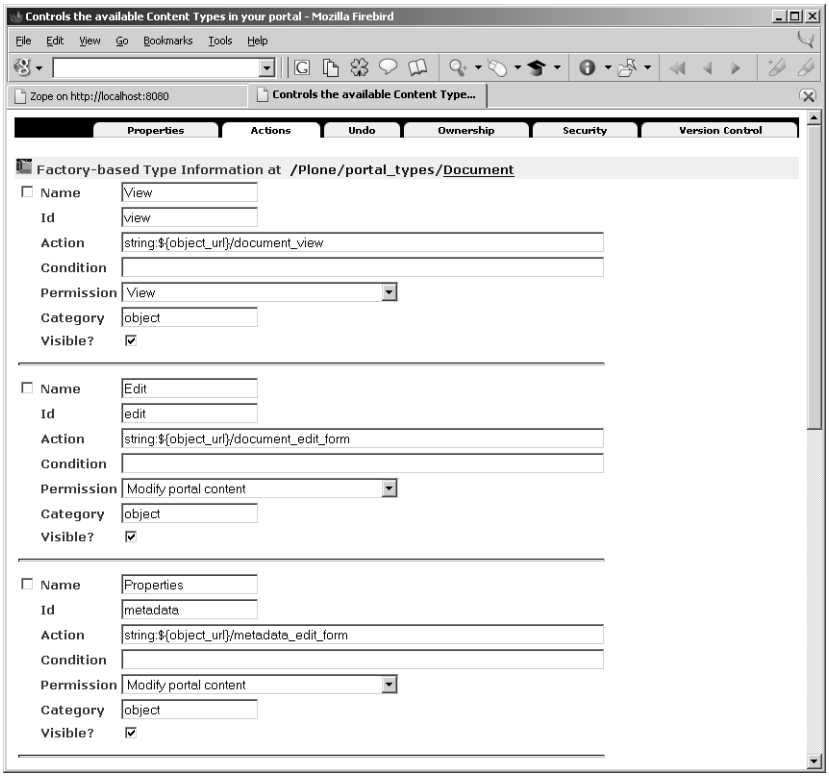


Figure 7–3 Page d'édition des actions

Dans la suite de ce chapitre, nous allons découvrir comment créer de toutes pièces de nouveaux types de contenu.

Comment créer un nouveau type de contenu ?

Il y a plusieurs méthodes pour créer de nouveaux types de contenu. Nous allons passer rapidement en revue trois d'entre elles, pour nous attarder ensuite sur la quatrième, nettement plus intéressante, puissante et efficace que les trois précédentes.

Ces méthodes sont :

- duplication à partir d'un type existant ;
- création à partir d'une ZClass ;
- développement d'un produit Python complet ;
- utilisation d'Archetypes.

Les voici donc en détail…

Créer à partir d'un type existant

Un type de contenu est un objet qui se base sur un produit de contenu CMF. Il est donc possible de définir plusieurs types différents, à partir d'un même produit. Ainsi, vous pouvez créer un type de contenu personnalisé à partir d'un type existant. C'est la solution de personnalisation la plus simple, mais c'est également la plus limitée. Faites le test avec le type `Document` par exemple.

L'opération peut être très simple et prendre deux minutes. Via l'interface d'administration de Zope, faites une nouvelle copie du type `Document` (avec les boutons **Copy** et **Paste**), puis renommez-le en `Article`, par exemple. Ensuite, dans l'interface d'édition du type, modifiez son titre, sa description et son méta-type. Le résultat est immédiat, comme vous pouvez vous en rendre compte en accédant au site en tant que membre via l'interface publique. Vous pouvez, en utilisant la liste déroulante d'ajout de contenu, créer un nouvel objet de type `Article` : celui-ci aura les mêmes propriétés que le type `Document` mais pourra par exemple être lié à un *workflow* différent, ou être muni d'actions différentes.

Pour aller au bout de la personnalisation à partir d'un type existant, il faut également changer les objets de présentation correspondant aux actions, notamment celles de visualisation et d'édition des contenus `Article`. Il est logique que chaque type ait sa propre présentation.

Créer à partir d'une extension ZClass

Zope permet d'étendre les classes d'objets standards, telles que `Image` ou `Folder`, en créant via le Web des composants nommés `ZClass`, avec la possibilité de leur définir des propriétés, des méthodes d'édition, des vues, et de les indexer avec le `ZCatalog`. Grâce à cette technique, vous pouvez donc définir une nouvelle classe d'objets héritant de classes de CMF, principalement la classe de base `PortalContent`. Cette solution est idéale pour ceux qui ne veulent pas trop programmer mais créer de nouveaux types de contenu relativement complets.

Nous allons utiliser le cas, par exemple, d'un système d'offres d'emploi internes, et créer un nouveau type de contenu `Job`.

B.A.-BA Les ZClass de Zope

Qu'est-ce qu'une ZClass ?

Une `ZClass` ou `Z Classe` est un produit Zope, qu'il est possible de créer, programmer, maintenir et distribuer entièrement via le Web. Il est possible de créer une `ZClass` à partir de rien, ou de bâtir une classe dérivée d'une `ZClass` existante ou d'une classe Zope.

Pourquoi utiliser une ZClass ?

Les `ZClass` ont leurs défenseurs et leurs détracteurs. Il faut envisager la `ZClass` comme un outil permettant de créer facilement et rapidement ses propres objets, en gardant à l'esprit que les `ZClass` ont aussi leurs limites. Nous avons, au chapitre précédent, opposé les *Python Scripts* aux *External Methods*. Les `ZClass` peuvent être vues d'une certaine manière comme la version objet des *Python Scripts*. Nous verrons au chapitre suivant qu'il est possible de créer des produits qui, eux, ne souffrent d'aucune limitation.

Une `ZClass` permet donc de définir de nouveaux types d'objets Zope.

Les `ZClass` présentent toutefois les inconvénients suivants :

- Pour les créer ou les maintenir, il faut disposer des droits suffisants dans le dossier `Control Panel` de Zope : ceci peut dissuader les hébergeurs basés sur Zope d'ouvrir une partie de leur serveur.
- Leur fonctionnement est soumis à certaines limites : le programmeur n'a pas accès à la structure interne de Zope et doit construire sa `ZClass` à partir d'un moule figé. Cette contrainte empêche la création d'applications vraiment complexes avec des `ZClass` ; nous verrons que les produits Python, en revanche, n'ont pas de telles limites.

L'utilisation de `ZClass` se fait de plus en plus rare sous Zope. En effet, les limitations sont telles qu'elles ne valent plus la peine de les subir… Et il est désormais possible, avec juste un peu plus de programmation, de créer des types de contenu complets et parfaitement fonctionnels. C'est ce que nous allons voir à l'étape suivante.

Créer à partir d'une extension Produit Python

Développer votre produit d'extension Python

Si vous avez une classe d'objets relativement complexe à mettre en œuvre pour votre site Plone, il est recommandé de créer un produit. Imaginons par exemple le cas d'un annuaire interne où chaque fiche serait un contenu dans le site. Le produit doit définir une classe permettant aux employés de la société d'ajouter des entrées à l'annuaire.

Démarrage

Le développement d'un produit Python se fait essentiellement sur le système de fichiers, et non pas à travers l'interface d'administration de Zope. Créez un répertoire nommé `Annuaire` dans le répertoire `$ZOPE/lib/python/Products`. Créez dans ce nouveau répertoire, à partir de votre éditeur de texte préféré, les fichiers suivants : `FicheAnnuaire.py` et `FicheAnnuairePermissions.py`. Ils vous serviront à créer respectivement le module de la classe d'objets Plone nommée `FicheAnnuaire` et le module de définition des permissions de cette classe.

Le module de définition des permissions

C'est le module le plus simple à mettre en place, et le plus court (cinq lignes de code ici). Vous créez respectivement la permission d'ajout des objets et celle de leur modification, puis vous affectez à ces deux permissions des rôles par défaut (`Manager` et `Owner`). Votre module contient le code suivant :

```
from Products.CMFCore.CMFCorePermissions import setDefaultRoles
# Définissons les permissions
AddFiche = 'Add Fiches'
ChangeFiche = 'Change Fiches'
# Affectons les rôles par défaut à ces permissions
setDefaultRoles(AddFiche, ('Manager', 'Owner',))
setDefaultRoles(ChangeFiche, ('Manager', 'Owner',))
```

Le module de définition de la classe

Dans le fichier `FicheAnnuaire.py`, Vous devez importer un ensemble de modules qui apportent les fonctionnalités utiles à la mise en œuvre de votre classe Zope :

```
from Globals import InitializeClass
from AccessControl import ClassSecurityInfo
import string
```

Les propriétés associées à la classe Entreprise

Typiquement, vous choisirez la liste de propriétés suivante :

Propriété	Nom	Type
Raison sociale	raison_sociale	Chaîne de caractères
Nom du contact	Contact	Chaîne de caractères
Adresse	Adresse	Chaîne de caractères
Code postal	code_postal	Chaîne de caractères
Ville	Ville	Chaîne de caractères
Téléphone	telephone	Chaîne de caractères
Fax	Fax	Chaîne de caractères
E-mail	Email	Chaîne de caractères
Site Web	site_url	Chaîne de caractères

Pourquoi importer ces classes ? Zope fournit une API avec un certain nombre de classes fondamentales. Pour qu'une classe d'objets Zope soit valide et puisse utiliser toute la puissance de l'environnement, il faut en quelque sorte signer un contrat avec ces classes au début du module. L'importation de chacune de ces classes représente un contrat avec le service ou le protocole concerné. Ainsi, sur la base de ces contrats, notre classe aura accès aux fonctionnalités de bas niveau fournies par Zope. Par exemple, la classe `AccessControl.ClassSecurityInfo` permet de gérer la sécurité des objets au sein de votre produit.

Plone est un *framework* spécialisé au-dessus de Zope. Il définit donc également des classes de base différentes de celles de Zope. Vous devez permettre l'accès aux fonctionnalités et services fournis par le *framework* en ajoutant les importations suivantes :

```
from Products.CMFDefault.DublinCore import DefaultDublinCoreImpl
from Products.CMFCore.PortalContent import PortalContent
from Products.CMFCore import CMFCorePermissions
```

Notez le fait qu'on importe le module `CMFCorePermissions` du produit de base `CMFCore`. Mais ce n'est pas tout. Vous avez commencé le travail en définissant des permissions spécifiques pour votre produit, dans le module `EntreprisePermissions`. C'est le moment d'importer ce module, puisque vous en aurez besoin pour mettre en œuvre la sécurité de votre nouvelle classe d'objets.

```
import EntreprisePermissions
```

Le Factory Type Information

Il s'agit de la définition d'une structure de données Python de type `tuple` nommée `factory_type_information`. Cette définition est utilisée pour construire le type d'objet au moment du chargement dans la base de données de Zope, au sein du composant outil `portal_types`.

Définition du type Entreprise (pour l'outil portal_types)

```
factory_type_information = ({
  'id' : 'Entreprise',
  'content_icon' : '',
  'meta_type' : 'Entreprise',
  'description' : "Entrée d'une entreprise dans l'annuaire",
  'product' : 'CMFAnnuaire',
  'factory' : 'addEntreprise',
  'immediate_view' : 'entreprise_edit_form',
  'actions' : ({
       'id'            : 'view',
       'name'          : 'Voir',
       'action'        : 'entreprise_view',
       'permissions'   : (CMFCorePermissions.View, )
     },
    { 'id'            : 'edit',
      'name'          : 'Editer',
      'action'        : 'entreprise_edit_form',
      'permissions'   : (EntreprisePermissions.ChangeEntreprises, )
    },
    { 'id'            : 'metadata',
      'name'          : 'Méta-données',
      'action'        : 'metadata_edit_form',
      'permissions'   : (EntreprisePermissions.ChangeEntreprises, )
    },
   ),
  },)
```

Détails du Factory Type Information

- `id` – L'identifiant unique de l'objet « type de contenu » (comme pour tout objet Zope) ;
- `meta_type` – Le nom du produit tel qu'indiqué à l'utilisateur dans `Control_Panel/Products` ;
- `description` – Comme son nom l'indique, la description de l'objet telle qu'indiquée à l'utilisateur ;
- `content_icon` – L'icône correspondant au type de contenu (optionnel) ;
- `product` – Le produit auquel appartient notre type (correspond au nom du répertoire) ;
- `factory` – La méthode Python qui va créer l'instance de notre objet dans le site CMF ;
- `immediate_view` – La page *template* qui va être appelée après la création de l'objet ;
- `actions` – La structure de type `tuple` définissant les actions.

Résumé

Vous devez définir au minimum les trois actions présentées dans le tableau suivant :

Id	Nom/Label	Template (skin)	Permission
view	Voir	entreprise_view	View
edit	Éditer	entreprise_edit_form	ChangeEntreprises
metadata	Méta-données	metadata_edit_form	ChangeEntreprises

La méthode d'ajout (factory)

Après ces définitions, nous passons à la méthode de création d'un objet `Entreprise` au sein de la base de données objet de Zope, ce sans quoi nous ne

pourrions pas gérer nos objets de contenu de manière persistante (un des avantages de la ZODB ou *Z Object Database*).

```
def addEntreprise(self, id, title='', REQUEST=None) :
   """Méthode « factory » de l'objet Entreprise"""
   obj = Entreprise(id)
   obj.id = id
   obj.title = title
   self._setObject(id, obj)
   if REQUEST is not None :
      REQUEST.RESPONSE.redirect('manage_main')
```

- Création de l'objet de type Entreprise.
- Affectation des propriétés idoines.
- Ajout de l'objet à l'arborescence Zope.
- Redirection vers la page d'administration.

À partir des attributs `id` et `title` et de l'objet de la requête HTTP, cette méthode crée une instance de la classe `Entreprise` et l'ajoute au sein de la base de données de Zope (la ZODB) en appelant la méthode interne `_setObject`.

Maintenant que la méthode d'instanciation des objets est prête, nous pouvons nous atteler à définir la classe elle-même.

La classe Entreprise

```
class Entreprise(PortalContent, DefaultDublinCoreImpl):


   """Une entreprise"""
   meta_type = "Entreprise"
   security = ClassSecurityInfo()
   security.declareObjectProtected(CMFCorePermissions.View)




   def __init__(self, id, title=''):



      DefaultDublinCoreImpl.init(self)


      self.id=id
      self.title=title
      self.raison_sociale = ''
      self.contact = ''
      self.adresse = ''
      self.code_postal = ''
      self.ville = ''
      self.telephone = ''
      self.fax = ''
      self.email = ''
      self.site_url = ''
```

- Notre classe hérite des classes `PortalContent` et `DefaultDublinCoreImpl`.
- Chaîne de documentation.
- Définition du méta-type.
- Création d'un objet chargé de la gestion de la sécurité des méthodes et attributs de la classe : il s'agit d'un mécanisme offert par Zope permettant aux Produits Python de disposer de méthodes privées, protégées ou publiques, et régies par les permissions de Zope.
- La méthode `__init__` est le constructeur de la classe : cette méthode est automatiquement appelée par Python au moment de l'instanciation de la classe.
- Appel de la méthode d'initialisation de la classe parente.
- Ces lignes de code permettent de renseigner les champs de notre objet avec des valeurs par défaut.

Grâce à l'héritage multiple permis par Python, votre classe possède deux ancêtres. En fonction de vos besoins, vous pouvez lui faire hériter d'autres classes telles que `CMFCore.PortalFolder` ou `OFS.Image.File`.

L'attribut `meta_type` caractérise les objets créés à partir de cette classe.

Vient ensuite la méthode `__init__`, chargée d'initialiser l'instance de la classe. Notez l'instruction `security.declareObjectProtected(CMFCorePermissions.View)` où vous vous servez de l'instance de la classe `ClassSecurityInfo` pour protéger l'accès à cette méthode par la permission `View`.

Dans cette méthode `__init__`, vous faites appel à la méthode d'initialisation de `DefaultDublinCoreImpl`, ce qui initialise toutes les métadonnées de l'objet, telles

RAPPELS

La classe `DefaultDublinCoreImpl` vous permet d'avoir accès aux fonctionnalités de métadonnées selon le standard Dublin Core, telles qu'implémentées dans le CMF.

L'héritage de la classe `PortalContent`, comme nous l'avons déjà vu, est nécessaire pour que l'objet soit utilisable en tant que contenu au sein du site CMF.

Interfaces des métadonnées (classe DefaultDublinCoreImpl)

Interface	Description
`Title`	Titre du contenu
`Creator`	Créateur
`Subject`	Liste de mots-clés représentant le sujet du contenu
`Description`	Description
`Contributors`	Liste des contributeurs
`CreationDate`	Date de création
`EffectiveDate`	Date de publication effective
`ExpirationDate`	Date d'expiration du contenu
`ModificationDate`	Date de modification
`Type`	Type du contenu (par exemple, `Document`, `Lien`, ou `Image`)
`Format`	Format (par exemple texte, texte structuré ou texte HTML)
`Language`	Langue
`Rights`	Droits (copyright)

Interfaces du contenu (classe PortalContent)

Interface	Description
`indexObject`, `reindexObject`	Permet d'indexer le contenu après sa création ou sa modification (`reindexObject`).
`unindexObject`	Permet de supprimer le contenu de la base d'indexation, avant sa suppression physique.
`manage_afterAdd`	Interface des actions à effectuer sur l'objet après son ajout dans la ZODB. *En général, vous n'avez pas à le surcharger dans votre produit.*
`manage_beforeDelete`	Interface des actions à effectuer sur l'objet avant sa suppression de la ZODB. *En général, vous n'avez pas à le surcharger dans votre produit.*
`ListFilteredActionsFor`	Retourne toutes les actions disponibles de manière contextuelle, sous la forme d'un dictionnaire contenant les actions liées à l'utilisateur telles que `Undo`, les actions liées à l'objet, et les actions globales telles que `folder_contents`.
`SearchableText`	Permet la recherche textuelle du contenu en retournant la liste des propriétés que le ZCatalog doit prendre en compte pour les index de type « texte ». Vous pouvez avoir à le surcharger en fonction des spécificités de votre classe d'objets.

que la description, la date de création et l'auteur. Vous initialisez ensuite les propriétés qui constituent le schéma de votre classe.

Où en sommes-nous ? Chaque instance de la classe peut s'initialiser avec tous ses attributs et hérite du comportement défini par les classes `PortalContent` et `DefaultDublinCoreImpl`, notamment l'indexation et la réindexation automatique. Vous allez maintenant permettre son édition, c'est-à-dire la modification de ses propriétés. Cela se fait grâce à la méthode suivante, nommée `edit` et protégée par la permission `ChangeEntreprises`.

```
security.declareProtected(EntreprisePermissions.ChangeEntreprises,'edit')
def edit(self, raison_sociale=None, contact=None, adresse=None,
   code_postal=None, ville=None, telephone=None, fax=None,
   email=None, site_url=None) :
   """Edition de l'objet"""
   self.raison_sociale = raison_sociale
   self.contact = contact
   self.adresse = adresse
   self.code_postal = code_postal
   self.ville = ville
   self.telephone = telephone
   self.fax = fax
   self.email = email
   self.site_url = site_url
   self.portal_catalog.reindexObject()
```

Ensuite, on réindexe l'objet (pour le `portal_catalog`).

La méthode `declareProtected` permet de déclarer une méthode de notre classe comme étant protégée par une permission de Zope. Ici, la méthode `edit` est protégée par la permission `ChangeEnterprises` : il n'est pas possible à un utilisateur de se servir d'`edit` s'il n'en a pas la permission.

La méthode `reindexObject` permet de mettre à jour l'objet dans le catalogue du portail. Sans cet appel à `reindexObject`, le catalogue du portail indexerait toujours l'ancienne version de l'objet, sans tenir compte des modifications apportées à ses attributs, ce qui est plutôt gênant pour faire des recherches sur des objets à jour !

La dernière ligne de code du module sert à initialiser la classe `Entreprise`.

```
# initalisation
InitializeClass(Entreprise)
```

La skin du produit

Vous pouvez maintenant créer au sein du répertoire `CMFAnnuaire`, un répertoire nommé `Skins` dans lequel vous créez un sous-répertoire nommé `Entreprise`.

Le formulaire d'édition

Dans le sous-répertoire `/CMFAnnuaire/Skins/Entreprise`, créez un fichier texte nommé `Entreprise_edit_form.pt` (l'extension étant nécessaire pour une *Page Template*) et ajoutez-y le code HTML suivant afin de fournir le formulaire d'édition des nouveaux objets :

```
<html metal:use-macro="here/main_template/macros/master">
<body>
<div metal:fill-slot="main">
    <h1>Edition des informations de l'entreprise</h1>
    <form class="group"
          name="edit_form"
          action="entreprise_edit"
          method="post">
        <div class="row"
             tal:define="Title request/title | here/Title;">
            <span class="label">Titre </span>
            <span class="field">
                <input type="text"
                       name="field_title"
                       size="40"
                       value="#"
                       tal:attributes="value Title" />
            </span>
        </div>

        <div class="row"
             tal:define="raison_sociale request/raison_sociale |
                      ➥ here/raison_sociale" >
            <span class="label">Raison sociale</span>
            <span class="field">
                <input type="text"
                       name="field_raison_sociale"
                       size="40"
                       value="#"
                     tal:attributes="value raison_sociale" />
            </span>
        </div>

        <div class="row" >
            <span class="label"> Catégorie(s) </span>
            <span class="field">
               <select name="field_categories:list" multiple
                       tal:define="categories here/Subject;
                       allowedSubjects
                       python:here.portal_metadata.
                         ➥listAllowedSubjects(here)">
                   <option value="cat"
                           tal:repeat="cat allowedSubjects"
                           tal:attributes="value cat;
                               ➥ selected python:cat in categories
                               ➥ and 1 or 0"
                           tal:content="cat">Une catégorie</option>
               </select>
            </span>
        </div>

        <div class="row"
             tal:define="contact request/contact | here/contact" >
            <span class="label">Nom du contact</span>
            <span class="field">
```

```
        <input type="text"
               name="field_contact"
               size="40"
               value="#"
               tal:attributes="value contact" />
    </span>
</div>

<div class="row"
     tal:define="adresse request/adresse | here/adresse" >
    <span class="label">Adresse</span>
    <span class="field">
        <input type="text"
               name="field_adresse"
               size="50"
               value="#"
               tal:attributes="value adresse" />
    </span>
</div>
<div class="row"
     tal:define="code_postal request/code_postal |
                 ➥ here/code_postal" >
    <span class="label">Code Postal</span>
    <span class="field">
        <input type="text"
               name="field_code_postal"
               size="25"
               value="#"
               tal:attributes="value code_postal" />
    </span>
</div>

<div class="row"
     tal:define="ville request/ville | here/ville" >
    <span class="label">Ville</span>
    <span class="field">
        <input type="text"
               name="field_ville"
               size="25"
               value="#"
               tal:attributes="value ville" />
    </span>
</div>
<div class="row"
     tal:define="telephone request/telephone | here/telephone" >
    <span class="label">Téléphone</span>
    <span class="field">
        <input type="text"
               name="field_telephone"
               size="25"
               value="#"
               tal:attributes="value telephone" />
    </span>
</div>
```

```
        <div class="row" tal:define="fax request/fax | here/fax" >
            <span class="label">Fax</span>
            <span class="field">
                <input type="text"
                       name="field_fax"
                       size="25"
                       value="#"
                       tal:attributes="value fax" />
            </span>
        </div>
        <div class="row"
             tal:define="email request/email | here/email" >
            <span class="label">E-mail</span>
            <span class="field">
                <input type="text"
                       name="field_email"
                       size="25"
                       value="#"
                       tal:attributes="value email" />
            </span>
        </div>

        <div class="row"
             tal:define="site_url request/site_url | here/site_url" >
            <span class="label">Site Web (URL)</span>
            <span class="field">
                <input type="text"
                       name="field_site_url"
                       size="25"
                       value="#"
                       tal:attributes="value site_url" />
            </span>
        </div>
        <div class="row">
            <span class="label"></span>
            <span class="field">
                <input class="context" type="submit"
                       name="submit" value=" OK " />
            </span>
        </div>

    </form>
</div>

</body>
</html>
```

Le script d'édition

Créez également un fichier nommé `Entreprise_edit.py` et ajoutez-y le code du script d'action du formulaire d'édition.

```
## Script (Python) "entreprise_edit"
## parameters=
## title=
REQUEST=context.REQUEST
context.editMetadata(title=REQUEST['field_title']
                   , subject=REQUEST['field_categories'])
context.edit(raison_sociale=REQUEST['field_raison_soiciale']
            , contact=REQUEST['field_contact']
            , adresse=REQUEST['field_adresse']
            , code_postal=REQUEST['field_code_postal']
            , ville=REQUEST['field_ville']
            , telephone=REQUEST['field_telephone']
            , fax=REQUEST['field_fax']
            , email=REQUEST['field_email']
            , site_url=REQUEST['field_site_url'])
REQUEST.set('portal_status_message', 'Entreprise+changée.')
qst='portal_status_message=Entreprise+changée.'
target_action = context.getTypeInfo().getActionById( 'view' )
context.REQUEST.RESPONSE.redirect( '%s/%s?%s' % (
context.absolute_url()
                                          , target_action
                                          , qst
                                          ) )
```

Appel de la méthode d'édition des métadonnées.

Appel de la méthode edit, que nous avons créée dans notre classe.

Le template de visualisation

Enfin, vous devez créer un fichier nommé `Entreprise_view.pt` pour la *template* de visualisation du contenu. Ajoutez-y le code HTML suivant :

```
<html metal:use-macro="here/main_template/macros/master">
<body>
<div metal:fill-slot="main" >
    <h1 tal:content="here/Title >Titre</h1>
        <div class="row">
            <span class="label">Raison sociale de l'entreprise</span>
            <span class="field" tal:content="here/raison_sociale" >
               Raison sociale
            </span>
        </div>
        <div class="row">
           <span class="label">Catégorie(s)</span>
           <span class="field"
                 tal:content="python:modules['string'].
                    ➥ join(here.Subject(), ' ')">Catégorie
           </span>
        </div>
```

```
        <div class="row">
            <span class="label">Contact</span>
            <span class="field" tal:content="here/contact" > Contact
            </span>
        </div>
        <div class="row">
            <span class="label">Adresse</span>
            <span class="field" tal:content="here/adresse" > Adresse
            </span>
        </div>
        <div class="row">
            <span class="label">Code Postal</span>
            <span class="field"
                  tal:content="here/code_postal" > Code postal
            </span>
        </div>
        <div class="row">
            <span class="label">Ville</span>
            <span class="field" tal:content="here/ville" > Ville
            </span>
        </div>
        <div class="row">
            <span class="label">Téléphone</span>
           <span class="field" tal:content="here/telephone" >Téléphone
            </span>
        </div>
        <div class="row">
            <span class="label">Fax</span>
            <span class="field" tal:content="here/fax" > Fax</span>
        </div>
        <div class="row">
            <span class="label">E-mail</span>
           <span class="field" tal:content="here/email" > Email</span>
        </div>
        <div class="row">
            <span class="label">Site web</span>
            <span class="field">
               <a href="" tal:attributes="href here/site_url"
                        tal:content="here/site_url" > URL du site </a>
            </span>
        </div>
</div>
</body>
</html>
```

Le module d'initialisation du produit

Il faut maintenant ajouter au répertoire du produit le fameux module `__init__.py` afin de permettre qu'à son démarrage, le serveur Zope puisse trouver le produit et le rendre accessible.

Vous devez importer d'une part le (les) module(s) de votre produit, et d'autre part un certain nombre de classes et de modules de `CMFCore`. Pour ce produit, les trois lignes suivantes devraient suffire :

```
import Entreprise
from Products.CMFCore import DirectoryView, utils
from Products.CMFCore import utils, CMFCorePermissions
```

Ensuite, vous transformez les répertoires contenant les fichiers de *skin* du produit en objets de type `File System Directory View` :

```
DirectoryView.registerDirectory('skins', globals())
DirectoryView.registerDirectory('skins/entreprise', globals())
```

Enfin, vous définissez la méthode `initialize` au sein de laquelle vous déclarez le produit de contenu pour Zope, en lui indiquant la classe d'objets, la permission d'ajout de contenu, le constructeur et le type associé. C'est également là que vous choisissez, pour votre nouveau produit, le nom qui apparaît dans le menu déroulant d'ajout d'objets au sein de l'interface de management de Zope.

```
def initialize(context):
    utils.ContentInit(
            'Entreprise Content Objects'
            , content_types=(Entreprise.Entreprise,)
            , permission=CMFCorePermissions.AddPortalContent
            , extra_constructors=(Entreprise.addEntreprise,)
            , fti=Entreprise.factory_type_information
            ).initialize(context)
```

RAPPEL

Un produit de contenu CMF reste un produit Zope. Vous pouvez ajouter les instances des classes définies dans le produit via la ZMI. Ceci dit, CMF apporte une nouvelle interface pour les gestionnaires de contenu, et ceux-ci n'ont jamais accès à la ZMI.

QUALITÉ Bogues

Pour tester votre produit, vous devez bien sûr démarrer Zope. Au moment où Zope interprète le code du produit, il se peut qu'il vous affiche des messages d'erreur. Soyez donc attentif à ce qui s'affiche sur la fenêtre de démarrage du serveur, afin de pouvoir efficacement « chasser les bogues » s'il y en a. Si aucun message d'erreur n'est affiché, alors le plus dur est fait.

Intégrer le nouveau type

Maintenant que votre produit fournissant la classe d'objets est prêt, vous pouvez créer une définition de type de contenu qui se base sur ce produit. La procédure est la même que celle du produit `ZClass`, sauf qu'au moment où vous arrivez sur la fenêtre d'ajout du `Factory-based Type Information`, vous pouvez sélectionner la classe dans la liste des classes de contenu disponibles. Vous sélectionnez alors `CMFAnnuaire:Entreprise` et vous validez, sans oublier de spécifier un identifiant tel que `Entreprise`. L'objet de type se crée et contient les paramètres tels que vous les avez définis au sein du produit avec le `factory_type_information`. Bien sûr, vous pouvez ensuite faire des modifications mineures sur certains paramètres, indépendamment du produit.

Voilà. Tout est en place, ou presque. Il ne reste plus qu'à charger la sous-skin permettant l'interaction Web avec le nouveau type de contenu.

Intégrer la sous-skin du nouveau type

Cette dernière étape consiste à ajouter la sous-skin au sein de l'outil `portal_skins`, et à mettre à jour les chemins de *skins* au niveau de son interface `Properties`. Comme vous avez un nombre sans cesse croissant de sous-skins,

l'opération peut être fastidieuse et source d'erreur. Il y a une solution, qui consiste tout simplement à faire exécuter ce travail de configuration par Python.

Créez le module suivant au sein du sous-répertoire `Extensions` de votre répertoire Zope, en le nommant `cmf_utils` :

```
from Products.CMFCore.utils import getToolByName
import string
def addFSDirectoryView(self, skin_id, physical_directory):
    skins_tool=getToolByName(self, 'portal_skins')
    skins_tool.manage_addProduct['CMFCore'].
          ➥ manage_addDirectoryView(id=skin_id,
          ➥ filepath=physical_directory)
    selections = skins_tool._getSelections()
    for id, skinpath in selections.items():
        skinpath=skinpath.split(',')
        skinpath.insert(2, skin_id)
        selections[id]=string.join(skinpath, ',')
    skins_tool._selections=selections
```

Ensuite, au sein de Zope, à la racine de votre instance de CMF, créez une méthode externe avec les paramètres suivants :

- id : `addFSDirectoryView` ;
- module : `cmf_utils` ;
- function : `addFSDirectoryView`.

Créez enfin un script Python, nommé `addSkinElement`, faisant appel à la méthode externe que vous venez de créer, en lui passant l'identifiant avec lequel la sous-skin doit être créée, et le chemin du répertoire sur le système de fichiers (à adapter selon votre cas) :

```
context.addFSDirectoryView(context, 'entreprise',
     '/home/zope/lib/python/Products/CMFAnnuaire/skins/entreprise')
return 'fin'
```

Vous lancez l'opération via l'onglet **Test** du script `addSkinElement`, et le tour est joué.

Utiliser le nouveau type de contenu

Paramétrer une liste de mots-clés pour le champ des catégories

Le dernier paramétrage à faire par le gestionnaire ou le concepteur du site consiste à définir une liste de mots-clés pour la valeur du champ `Catégories`, qui est proposée aux membres lorsqu'ils créent du contenu `Entreprise` au sein de l'annuaire. C'est exactement ce que permet de faire le composant de gestion des métadonnées `portal_metadata`.

Cliquez sur ce composant, puis sur son onglet **Elements**, et enfin sur le lien correspondant à l'élément de métadonnées qui vous intéresse, afin d'obtenir son

formulaire de définition. En fait, le formulaire de l'élément `Subject` étant ouvert par défaut lorsqu'on accède à l'interface des éléments, vous n'avez pas besoin de cliquer une deuxième fois.

Une première section montre une définition commune à tous les types. Le but est de faire une définition spécifique pour certains types. C'est ce qui a été fait au sein de Plone pour le type `Event` (du produit `CMFCalendar`), dont la section apparaît en deuxième position. À la suite, vous avez la possibilité d'ajouter votre définition pour le type `Entreprise`. Sélectionnez votre type à partir de la liste déroulante des types de contenu disponibles sur le site. Vous pouvez maintenant faire votre définition dans la section réservée à ce type en vous inspirant de ce qui a été fait pour le type `Event` :

- Required : cochez la case.
- Default : entrez par exemple le mot-clé « SSII ».
- Supply default : cochez la case.
- Vocabulary : entrez par exemple les mots-clés « Conseil », « SSII », « Éditeur », et « R&D », en allant à la ligne après chaque entrée.
- Enforce Vocabulary : cochez la case.

Créer les entrées de l'annuaire

Une fois de plus, via l'interface publique du site, les membres identifiés peuvent créer et publier du contenu correspondant au type que vous venez de mettre en œuvre.

Définir le nouveau type de contenu

La phase suivante consiste à définir votre nouveau type de contenu au sein de l'outil `portal_types` du site CMF. Vous y ajoutez un objet `Factory-based Type Information` avec l'identifiant `Job` par exemple, et les onglets Properties et Actions vous permettent de faire son paramétrage.

Propriétés du type

À ce stade, les propriétés importantes à définir sont les suivantes :

- Meta type : `Job` ;
- Product name : `ZoperaExtensions` ;
- Factory method in product : `Job_add` ;
- Initial view name : `job_edit_form`.

Vous voyez là le lien avec le produit `ZClass` que nous avons étudié plus haut.

Actions associées au type

Vous devez ensuite définir, sous l'onglet Actions du nouveau type, les trois principales actions d'édition du contenu : l'action de voir le contenu (`view`), celle de l'éditer (`edit`), et celle de modifier ses métadonnées (`metadata`).

RAPPEL Définition d'une action

- `Name` : nom de l'action, tel qu'il apparaîtra dans l'interface du CMF ;
- `Id` : identifiant unique de l'action ;
- `Action` : méthode de l'objet invoquée par cette action ;
- `Permission` : permission nécessaire pour effectuer l'action ;
- `Category` : catégorie de l'action (`object` ou `folder`) permettant de situer l'action dans l'interface du CMF ;
- `Visible` : cochez cette case pour que l'action apparaisse dans l'interface du CMF. Autrement, elle sera utilisable mais pas listée par défaut dans l'interface du CMF.

Voici, résumés dans un tableau, les paramétrages des différentes actions.

	Voir	Éditer	Éditer les métadonnées
Name	Voir	Éditer	Méta-données
Id	view	edit	metadata
Action	job_view	job_edit	metadata_edit_form
Permission	View	Add Jobs	Modify portal content
Category	Object	Object	Object
Visible ?	Oui	Oui	Oui

ASTUCE

Vous pouvez déplacer des actions vers le haut ou vers le bas (pour changer leur ordre d'affichage dans l'interface publique du site).

Vous pouvez créer d'autres actions en plus des trois actions de base du type de contenu. La seule condition est de disposer, au sein de vos objets, d'une méthode à insérer dans le champ `Action`. Par exemple, les objets de Zope disposent d'une méthode `manage_main` permettant d'afficher la page d'administration de Zope de l'objet concerné. Vous pouvez donc créer une nouvelle action d'après les informations suivantes :

- Name : `Administrer` ;
- Id : `manage` ;
- Action : `manage_main` ;
- Permission : `View management screens` ;
- Category : `object` ;
- Visible : *cochée*.

Utilisation d'Archetypes

RAPPEL Qu'est-ce qu'Archetypes ?

Archetypes est un *framework* fournissant des classes de base et des services aux programmeurs pour créer de nouveaux types de contenu. Il est ainsi nettement plus facile de créer, en partant de zéro, des types complexes et relativement riches.
Les avantages d'Archetypes sont bien entendu une très grande facilité de prise en main, la richesse des outils proposés, mais aussi et surtout la force de la communauté participant au développement d'Archetypes.
La référence des champs et widgets Archetypes est donné dans l'annexe 3 de cet ouvrage.

Archetypes est un outil plus que puissant pour simplifier la création de produits sous Plone, permettant notamment d'améliorer la qualité des développements en réduisant le nombre de lignes de code nécessaires. Désormais intégré à Plone depuis sa version 2.0, il serait dommage de ne pas en profiter !

Nous allons nous attacher à la création d'un type de contenu décrivant les objets vendus par la Société Géniale. Il ne s'agit pas de faire une boutique en ligne, mais juste une sorte de catalogue permettant d'identifier tous les articles que propose la société. Le lecteur averti pourra bien entendu s'attacher à la réalisation d'une boutique.

Commencez par créer, dans le répertoire `Products` de Zope, un répertoire appelé `SGObject`. Il s'agira du nom de la classe que nous allons utiliser.

Créez, à l'intérieur de ce répertoire, un répertoire `skins` et un répertoire `Extensions`. Enfin, dans le répertoire `skins`, créez un sous-répertoire appelé `sgobject` (en minuscules). Et c'est parti !

Le principe de base

La création d'un produit basé sur Archetypes est réduite à son minimum vital. Pour fournir un type de contenu, il faut les fichiers suivants :

- le script d'initialisation du module (`__init__.py`) ;
- la classe du type de contenu (nous l'appellerons `SGObject.py`) ;
- un script de configuration (appelé `config.py`) ;
- un script d'installation du produit (`Extensions/Install.py`).

Archetypes permet de définir très simplement les champs qui vont être portés par le contenu. Nous allons commencer par un exemple simple, avec juste les champs `id`, `titre` et une zone de texte quelconque. Voici comment procéder.

La configuration du produit

Sauf mention contraire, les fichiers présentés ci-après doivent être placés dans le répertoire `Products/SGObject` que vous venez de créer. Ainsi, le code suivant doit être placé dans le fichier `config.py`. Il contient simplement des informations permettant d'identifier le projet.

Avant toute chose, nous importons quelques modules utiles.

```
from Products.CMFCore.CMFCorePermissions import AddPortalContent,
ModifyPortalContent
from Products.Archetypes.utils import DisplayList
```

Puis, il convient de définir des informations relatives au projet, comme son nom, l'emplacement du répertoire des *skins* spécifique au projet, le méta-type de la classe du contenu, et enfin une variable globale utilisée au moment de l'installation.

```
# General settings
PROJECTNAME = "SGObject"
SKINS_DIR = 'skins'
PROJECTMETATYPE = "SGObject"
GLOBALS = globals()
```

Enfin, nous définissons les permissions nécessaires pour créer et modifier les contenus de notre type. Il s'agit ici des permissions standard de CMF, pour garder le modèle le plus simple possible.

```
# Permissions
ADD_CONTENT_PERMISSION = AddPortalContent
EDIT_CONTENT_PERMISSION = ModifyPortalContent
```

Le code d'initialisation du module

Le code d'initialisation est chargé par Zope lors de son démarrage. Il doit généralement contenir les déclarations de base permettant à Zope de déterminer quelles classes sont utilisées par le produit. Avec Archetypes, le principe est toujours le même, mais l'écriture du code est plus simple. Placez, dans un fichier `__init__.py`, le code suivant.

Attention

La fonction de rafraîchissement (`refresh`) de produits, bien connue des développeurs Zope, fonctionne malheureusement très mal avec Archetypes : elle est donc à proscrire !

Comme d'habitude, quelques imports bien sentis…

```
from Globals import package_home
from Products.Archetypes.public import *
from Products.CMFCore.DirectoryView import registerDirectory
from Products.CMFCore import utils
import os, os.path
```

Ah ! … nous faisons ici référence aux variables définies dans le fichier `config.py`. Celles-ci permettent au code d'initialisation d'être commun à plusieurs produits. En effet, si vous voulez créer un nouveau produit basé sur Archetypes, il vous suffira de dupliquer tout le code en changeant simplement les informations fournies dans `config.py`. C'est là que commence la véritable « réutilisabilité » !

```
from config import *
```

La ligne de code suivante permet de déclarer que les *skins* de notre produit sont contenues dans le répertoire `skins` (que vous avez dû créer précédemment).

```
registerDirectory(SKINS_DIR,GLOBALS)
```

Enfin, le code d'initialisation doit fournir une méthode `initialize` contenant les procédures de déclaration de notre type de contenu. Là encore, l'utilisation de `config.py` rend ce code tout à fait réutilisable pour d'autres projets. Ce code ne présente d'ailleurs pas un énorme intérêt : il ne fait que s'appuyer sur l'API de Plone. Seule la ligne en gras revêt une importance particulière : elle permet d'initialiser notre type de contenu (nous verrons le module correspondant ultérieurement).

```
def initialize(context):
    # process our custom types
    import SGObject
    content_types, constructors, ftis = process_types(
        listTypes(PROJECTNAME),
        PROJECTNAME)
    utils.ContentInit(
        PROJECTNAME + ' Content',
        content_types = content_types,
        permission = ADD_CONTENT_PERMISSION,
        extra_constructors = constructors,
        fti = ftis,
        ).initialize(context)
```

Le script d'installation

Le script d'installation doit être placé sous `Extensions/Install.py`. Il s'agit d'un bout de code relativement simple. Ce script a pour but de faciliter la tâche de l'intégrateur, en effectuant pour lui le travail de déclaration des *skins* et du type de contenu sous Plone. Ce module est appelé par le QuickInstaller pour permettre l'installation facile du produit.

Voyons en détail, en commençant par notre sempiternelle liste d'imports :

```
from Products.Archetypes.public import listTypes
from Products.Archetypes.Extensions.utils import installTypes,
install_subskin
from Products.ArchExample.config import PROJECTNAME, GLOBALS
from StringIO import StringIO
```

La méthode principale de ce module est `install`. C'est elle qui sera appelée par le QuickInstaller pour effectuer les tâches d'installation.

```
def install(self):
    out = StringIO()
```

La première chose à faire est d'installer les types de contenu dans l'outil `portal_types` : c'est la méthode `installTypes` qui s'en charge.

```
    installTypes(self, out, listTypes(PROJECTNAME), PROJECTNAME)
```

Enfin, il convient de déclarer la *skin* au sein de l'outil `portal_skins`. Ici, c'est la méthode `install_subskin` qui réalise le travail

```
    install_subskin(self, out, GLOBALS)
```

Afin d'afficher, dans le QuickInstaller, un gentil message de confirmation, nous utilisons un fichier en mémoire (`StringIO`). C'est une chaîne de caractères qui possède le comportement d'un fichier. Autrement dit, cela permet d'utiliser les méthodes `write` et `getvalue` pour respectivement ajouter du contenu ou lire le contenu de la chaîne. Ce module n'est utilisé que pour le message affiché par QuickInstaller et nous ne le décrivons qu'à l'attention du lecteur curieux !

```
    out.write("Successfully installed %s." % PROJECTNAME)
    return out.getvalue()
```

La classe du produit

Cette classe est fondamentale pour notre étude. C'est elle, en effet, qui est porteuse de toutes les informations réellement spécifiques à notre type de contenu. Autant les fichiers précédents servent surtout de « fioritures », autant le module qui porte cette classe est celui dans lequel vous passerez le plus de temps à coder. Commençons par notre exemple tout simple d'un type de contenu portant juste un titre et une description. Comme d'habitude, nous commençons par… les imports !

```
from Products.Archetypes.public import *
from Products.Archetypes.BaseContent import BaseContent,
BaseContentMixin
from Products.Archetypes.utils import DisplayList
from Products.CMFCore import CMFCorePermissions
from config import *
from AccessControl import ClassSecurityInfo
import string
```

Les schémas

Sous Archetypes, les types de contenu sont décrits par des schémas. Un schéma est une liste de définition de champs avec, pour chaque champ, une expression de son identifiant, de son type, de son titre, et du composant utilisé pour afficher ou éditer sa valeur. On peut spécifier beaucoup d'autres choses dans un schéma, comme nous le verrons plus loin.

Voici maintenant la partie la plus intéressante : le schéma. C'est ici que nous décrivons les différents champs de notre type de contenu.

La ligne suivante décrit simplement un champ de type texte appelé « Description de l'objet », représenté par une balise `TextArea`, obligatoire et dont la valeur est stockée dans l'attribut `body` de la classe et cataloguée. Voyons plutôt (attention aux parenthèses, elles sont toutes obligatoires !) :

```
schema = Schema(
    (
    TextField(
        'body',
        searchable=1,
        required=1,
        widget=TextAreaWidget(
            label="Description de l'objet",
            ),
        ),
    ))
```

Et voilà. Le schéma est défini. Il reste encore à créer le corps de la classe… Voici toute la puissance d'Archetypes résumée ici en deux lignes de code (attention aux yeux, c'est éblouissant) :

```
class SGObject(BaseContent):
    schema = BaseSchema + schema
```

Juste pour la forme : outre la déclaration de l'utilisation du schéma défini ci-dessous, nous déclarons au passage que la classe se base également sur `BaseSchema`, qui est le schéma standard d'Archetypes définissant les champs « identifiant » et « titre » (autrement dit, les champs dont vous aurez sans arrêt besoin !). Car, oui, les schémas peuvent être concaténés aussi simplement qu'un *tuple* Python… Nous verrons plus loin que l'on peut même faire beaucoup plus avec ces fameux schémas.

Enfin, il convient d'enregistrer ce nouveau type de contenu :

```
registerType(SGObject)
```

Et voilà. C'est fini. Nous n'avons pas créé une seule ZPT, car Archetypes va s'en charger pour nous. Redémarrez Zope, allez dans QuickInstaller, installez Archetypes si ça n'est pas déjà fait, puis installez votre produit. Vous pouvez maintenant aller dans Plone et constater qu'il fonctionne à merveille !

Archetypes… la tête contre les murs

Les nombreuses qualités d'Archetypes ne sauraient dissimuler un défaut : son extrême rigueur syntaxique et la pauvreté des messages d'erreur ou d'avertissement qu'il peut produire lors de la création d'un produit. Il suffit d'une parenthèse mal placée, d'un guillemet oublié, d'une ligne de code omise pour empêcher votre type de contenu d'apparaître dans la liste déroulante du back-office de Plone. Si cela vous arrive, vous aurez beau chercher rationnellement dans la *traceback* de Zope, vous ne trouverez rien : Archetypes a « avalé » silencieusement votre type de contenu.

Dans un pareil cas, adoptez une démarche rationnelle : reprenez chaque bout de code, vérifiez si vous n'avez rien oublié. L'erreur est forcément quelque part ! Si cela vous arrive après une modification de votre produit, surveillez attentivement les lignes de code que vous avez modifiées. Enfin, nous ne saurions trop rappeler que l'utilisation d'un serveur CVS dans l'écriture d'un produit peut être salvatrice à bien des titres !

Étendre le schéma

Notre schéma est certes concis, mais surtout notoirement insuffisant ! Nous cherchons à décrire des objets vendus par la société, et il convient de leur ajouter nombre d'informations pertinentes, comme leur description, leur prix, ou même une catégorie. Nous allons donc étendre le schéma présenté ci-dessus pour ajouter les champs pertinents.

```
schema = Schema(
    (
    TextField(
        'body',
        searchable = 1,
        required = 1,
        widget=TextAreaWidget(
            label="Description de l'objet",
            ),
        ),
```

Sous la description de l'objet (les champs d'Archetypes sont ordonnés), nous pouvons ajouter, par exemple, un prix. Ici il ne s'agit plus d'un champ de type texte mais d'un entier. Archetypes fournit une liste de champs suffisamment riche pour y trouver son bonheur. Ici, le champ est mentionné comme indexé (`searchable`) et obligatoire (`required`).

En se basant sur l'exemple du champ précédent, nous pouvons écrire le morceau de code suivant :

```
    FloatField(
        "price",
        searchable = 0,
        required = 0,
        default = 0.0,
        widget = StringWidget(
            label = "Prix de l'objet",
            ),
        ),
```

Ici, nous mentionnons que le champ n'est pas obligatoire (`required=0`) et que sa valeur ne doit pas être cataloguée (`searchable = 0`) : cela n'a aucun sens de rechercher un prix dans le moteur de recherche du site ! Nous avons ajouté une mention pour forcer une valeur par défaut à `0` pour ce champ. Enfin, ce prix est saisi dans une zone d'édition classique, ce que nous appelons `StringWidget`. Le `label`, comme nous l'avons constaté avec l'exemple précédent, est l'intitulé du champ tel qu'il apparaît dans les écrans.

Plus fort encore : nous pouvons ajouter une image. Archetypes fournit l'attirail nécessaire pour la gestion simple d'une image dans un type de contenu, comme le montre la suite de notre nouveau schéma :

```
    ImageField(
        "photo",
        widget = ImageWidget(
            label = "Photo de l'objet",
            ),
        ),
    ))
```

Une fois ces modifications effectuées, relancez Zope : vous constatez alors que les nouveaux champs sont automatiquement pris en compte, et qu'il est facile d'ajouter un prix et une image pour vos contenus de type objet. La vie est belle !

Field et Widget

Archetypes utilise, pour la description de ses schémas, deux notions fondamentales : les champs (`Fields`) et les zones visuelles (`Widgets`). Un champ est la définition de la manière dont est stocké et géré le contenu par la classe, en particulier s'il est obligatoire, s'il doit être indexé, son nom, les valeurs qu'il lui est possible de prendre, etc. Un `widget` est la définition de la manière dont le champ est affiché, tant en saisie qu'en modification. Un champ de type texte, par exemple, pourrait être édité dans un `input` HTML classique ou dans une zone de texte (`TextArea`). Le programmeur a la possibilité de forcer l'un ou l'autre cas en spécifiant le `widget` du champ.

ASTUCE Les valeurs par défaut des attributs des champs

En toute rigueur, dans le cas du `FloatField`, il n'est absolument pas nécessaire de spécifier `searchable=0` ou `required=0` : chaque champ utilise des valeurs par défaut pour ses différentes options de configuration, et, dans le cas du `FloatField`, ces propriétés sont positionnées telles que nous l'aurions souhaité.

Redémarrage de Zope et modifications du schéma

Dans tous les cas, Archetypes se charge de l'initialisation de votre produit au moment du démarrage de Zope. Autrement dit, une fois que vous avez installé une fois votre produit avec le QuickInstaller, il n'est pas nécessaire de le réinstaller lorsque vous modifiez votre code. Pire : si votre code comporte une erreur, vous pouvez rendre votre Plone instable en tentant de désinstaller un produit qui ne fonctionne pas !

Si votre code comporte des erreurs, il se peut que votre type de contenu « disparaisse » de `portal_types`. Dans ce cas, concentrez-vous sur vos modifications, corrigez votre code, mais n'incriminez pas QuickInstaller et ne tentez pas non plus de désinstaller ou réinstaller votre produit à la main.

Exploiter la richesse d'Archetypes

Les possibilités d'Archetypes sont bien plus grandes que la simple définition de quelques champs. Nous allons, dans la section qui suit, exposer quelques unes des options offertes pour augmenter la productivité des développeurs.

Les différents types de champs et widgets

Nous avons vu qu'il était possible, avec les champs d'Archetypes, de créer des champs très simples (zone de texte, par exemple), comme des champs beaucoup plus complexes. Voici une liste de types de champs, présentés dans des exemples, qui vous donneront une idée de l'efficacité d'Archetypes. Nous allons donc étendre notre schéma avec les exemples suivants.

D'abord, nous avons la possibilité de placer un fichier dans notre type de contenu. Il est possible, par exemple, d'associer un fichier de description, un document Word, etc. Archetypes ne se charge pas de l'interprétation du fichier attaché, mais le composant PloneArticle permet d'étendre Archetypes pour proposer une indexation du contenu et un aperçu directement en HTML.

```
FileField(
    "fichier",
    widget = FileWidget(
        label = "Fichier attaché",
        ),
    ),
```

Il est possible de proposer des champs de gestion de la date : ceux-ci seront affichés d'une manière particulièrement élégante, avec une saisie de la date dans un calendrier créé en JavaScript. La définition d'un tel champ est d'une simplicité enfantine :

```
DateTimeField(
    "shipping_date",
    widget = CalendarWidget(
        label = "Date de sortie du produit",
        ),
    ),
```

On peut également utiliser un champ de type booléen, comme ceci :

```
BooleanField(
    "obsolete",
    widget = BooleanWidget(
        label = "Obsolète",
        description = "Cliquez ici si ce produit n'est
                       plus disponible",
        ),
    ),
```

Notez au passage la ligne `description`, qui permet d'ajouter un texte dans les formulaires d'édition des valeurs. Ceci est très pratique pour ajouter un texte d'explication à un champ complexe.

Enfin, il existe un champ très pratique pour afficher une valeur issue de l'exécution d'une méthode. Voici un exemple simple :

```
ComputedField(
    "auteur",
    widget = StringWidget(
        label = "Auteur du document",
        ),
    accessor = "Creator",
    ),
```

Nous avons ici créé un champ calculé, qui sera visible mais pas modifiable. Sa valeur est donnée par l'attribut `accessor` : il s'agit du résultat de l'appel de la méthode `Creator` qui, pour un type de contenu donné, renvoie l'identifiant de son auteur.

En fait, il est possible d'utiliser l'attribut `accessor` sur n'importe quel champ. De même, il est possible de rendre n'importe quel champ en lecture seule en lui spécifiant l'attribut :

```
        mode = 'r'
```

Et de même, on peut rendre un champ en mode écriture seule (il n'apparaîtra pas pour les lecteurs) en spécifiant l'attribut :

```
        mode = 'w'
```

Liste de valeurs (vocabulaire)

Enfin, un dernier type de champ est la sélection dans une liste. La seule difficulté technique réside dans le fait que, pour un tel champ, il est indispensable de mentionner la liste de valeurs possibles. Voici le code correspondant :

```
StringField(
    "etat",
    vocabulary = "getEtats",
    widget = SelectionWidget(
        label = "Etat de l'objet",
    ),
```

La zone de sélection est restreinte au vocabulaire mentionné par l'attribut `vocabulary`. Il faut définir la méthode `getEtats` dans le corps de la classe du type de contenu pour que tout cela fonctionne. Rien de plus facile :

```
security = ClassSecurityInfo()
security.declarePublic("getEtats")
```

```
def getEtats(self,):
    "getEtats"
    return DisplayList((
        ("1", "Etude", ),
        ("2", "Prototype", ),
        ("3", "Fabrication en série", ),
        ("4", "Ecoulement du stock", ),
        ("5", "Epuisé", ),
        ))
```

Nous déclarons une méthode publique, qui retourne un objet de type `DisplayList` : il s'agit d'un type utilisé par Archetypes pour représenter des listes de sélection sous forme de paire clé/valeur. Ainsi, dans l'exemple ci-dessus, la clé sera le numéro passé en paramètre.

Redémarrez Zope, et admirez votre nouveau type de contenu !

En résumé...

Il est possible, avec Plone, de créer ses propres types de contenu pour étendre le modèle de données déjà disponible. La création d'un type de contenu peut se faire par duplication d'un type existant – il s'agit là du cas le plus simple, mais aussi du plus limité. Le programmeur peut également écrire une `ZClass` pour son nouveau type : il s'agit d'un mécanisme permettant de créer rapidement de nouvelles classes dans Zope, mais cette technique est tombée en désuétude au profit de l'écriture de produits Python. Il peut être fastidieux d'écrire des produits complets pour des types de contenu structurés de la même manière : pour cela, Archetypes répond efficacement aux besoins de la plupart des développeurs. Avec Archetypes, il est simple et rapide de partir d'un modèle de base pour l'étendre au gré des besoins.

8 Mise en production

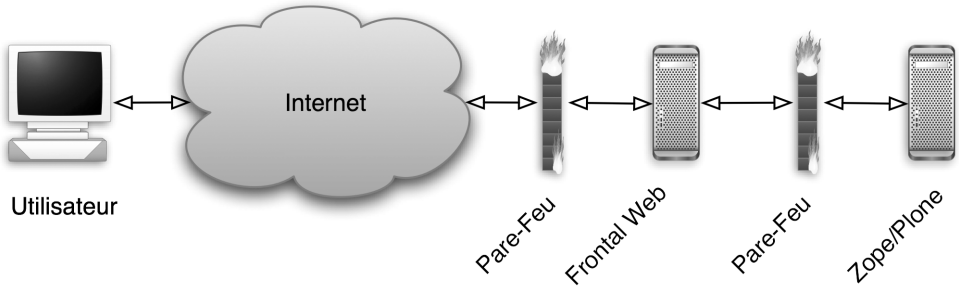
Zope Plone

SOMMAIRE

MOTS-CLÉS

- Installation
- Production
- Apache

Installation | Production | Apache



Nous sommes maintenant prêts à faire migrer le site sur un serveur de production. Ce chapitre nous donne des informations complémentaires pour aller plus loin dans la mise en œuvre d'un système de haute performance reposant sur Zope.

Une bonne installation est fondamentale

Bien installer le système, c'est se garantir dès le début que l'on va se simplifier la tâche pour sa maintenance. En effet, en réduisant au maximum les risques d'incidents de production pour l'exploitant, on se met en position de faire durer le site longtemps et donc d'aider à la pérennité du projet. Pour ce faire, nous allons respecter un certain nombre de règles fondamentales.

Nous allons notamment assurer une séparation franche entre Python, Zope et les instances (les composants propres et les données).

De plus, pour chaque instance de Zope, nous allons garantir la séparation franche entre :

- les données (`data.fs` notamment) ;
- les produits propres à l'instance ;
- les fichiers de logs ;
- les fichiers de configuration.

En respectant ces principes, il est aisé d'effectuer les tâches de maintenance : identifier les éléments à sauvegarder régulièrement, mettre à jour un site d'une version de Zope à l'autre, etc.

Autant les versions 2.6 et antérieures étaient difficiles à bien mettre en œuvre sur ces points, autant les versions 2.7 et supérieures proposent des outils idéaux pour ces tâches.

En effet, depuis Zope 2.7, l'installation ne mélange plus le moteur de Zope avec les données de l'instance installée.

Zope 2.7 propose de facto l'arborescence d'installation du type :

```
/Zope-2.7
   /bin
   /doc
   /lib
      /python
         /Products
   /…
/instances/
   /instance1
      /Extensions
      /Products
      /bin
      /etc
      /import
      /log
      /var
   /instance2
      /Extensions
      /Products
      /bin
```

```
      /etc
      /import
      /log
      /var
   /…
/…
```

Installation d'un site en production

Préparation de la plate-forme

Le présent chapitre couvre l'installation d'un site Zope sous Unix. Les lecteurs qui utilisent MS-Windows pour une mise en production adapteront ces explications à leur environnement.

Hypothèse est faite que l'utilisateur a à sa disposition un serveur Unix fraîchement installé. Les auteurs recommandent les Unix libres comme Linux ou les systèmes BSD. Zope est connu pour également parfaitement fonctionner sous Mac OS X, Solaris, AIX et autres versions d'Unix.

ATTENTION Distributions

Attention à Zope sous MS-Windows, au moins jusqu'à la version 2.6.1. En effet, compilé avec les outils Microsoft Visual Studio, Python 2.1.3 n'accepte pas les fichiers de plus de 2 Go ; la base de données de Zope souffrira alors de ce handicap. La version 2.6.2, qui fonctionne avec Python 2.3.2, est censée corriger cette limitation.

La distribution Redhat 7.x est connue pour poser des problèmes insolubles de dépendances entre paquets si l'on utilise les RPMs. Redhat 9.x fonctionne parfaitement. Debian est recommandée pour ses qualités de facilité de déploiement grâce au système de gestion de paquets APT.

Installation des paquetages logiciels principaux

Sur une plate-forme Linux Debian, ces éléments s'installent très simplement avec les commandes (en tant que `root`) :

```
$ apt-get update
$ apt-get install wget
$ apt-get install gcc
$ apt-get install make
$ apt-get install automake
$ apt-get install libc6-dev

$ apt-get install python2.3
$ apt-get install python2.3-dev
$ apt-get install locales(choisir fr_FR ISO-8859-1 dans le menu à
l'install)
$ locale-gen
```

Suivant la « fraîcheur » de votre Debian, vous pouvez être amené à installer certains paquets (notamment `python2.3`) depuis la source `testing`.

Sur une plate-forme FreeBSD, l'installation de ces mêmes paquets se fait avec (en tant que `root`) :

```
$ cd /usr/ports/ftp/wget && make install clean
$ cd /usr/ports/lang/python && make install clean
```

Les autres paquets sont déjà installés en standard sur cette plate-forme.

Dans la version MS-Windows du produit ZAAPlugins, ces compléments sont directement livrés sous une forme binaire : aucune installation de ces logiciels n'est donc à prévoir sous MS-Windows. De même, la présence de Microsoft Office ou Adobe Acrobat n'est pas requise.

Installation des autres paquetages logiciels

Les paquets suivants sont optionnels mais seront nécessaires, notamment pour indexer le contenu des fichiers bureautiques ou travailler avec des images sur le site :

```
$ apt-get install python2.3-imaging
$ apt-get install wv
$ apt-get install xlhtml
$ apt-get install xpdf
$ adduser zope
Adding user zope...
Adding new group newbie (1001).
Adding new user newbie (1001) with group zope.
Creating home directory /home/zope.
Copying files from /etc/skel
Changing password for zope
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Re-enter new password:
Password changed.
Changing the user information for newbie
Enter the new value, or press return for the default
        Full Name []: Zope Virtual User
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/n] Y
```

Installation de Zope

Utilisateur Zope

Commençons par ajouter un utilisateur zope à notre système (en tant que root) :

Nous voilà donc avec un nouvel utilisateur zope, pour lequel nous aurons pris soin de choisir un mot de passe bien compliqué.

Le moteur de Zope

Nous allons maintenant installer le moteur de Zope. Premier point : connectons-nous en tant qu'utilisateur zope :

```
$ su - zope
zope$ mkdir src
zope$ cd src
```

Puis, avec ce nouvel utilisateur, téléchargeons l'archive de Zope (ici la 2.7.0) :

```
zope$ wget http://zope.org/Products/Zope/2.7.0/Zope-2.7.0.tgz
```

Décompressons l'archive :

```
zope$ tar xvzf http://zope.org/Products/Zope/2.7.0/Zope-2.7.0.tgz
zope$ cd Zope-2.7.0
```

Compilons :

```
zope$ ./configure --optimize --prefix=/home/zope/Zope2.7
zope$ make
```

La compilation prend quelques minutes. On peut ensuite installer Zope :

```
zope$ make install
```

Nous disposons maintenant d'une version du moteur de Zope installé dans le répertoire `/home/zope/Zope2.7`.

Une première instance de Zope

Notre instance va s'appeler (de manière très originale) `plone1`.
En tant qu'utilisateur `zope`, créons `plone1` :

```
zope$ cd
zope$ mkdir instances
zope$ Zope2.7/bin/mkzopeinstance.py --dir=instances/plone1
```

Ce script demande de choisir l'identifiant et le mot de passe de l'administrateur de l'instance :

```
Username: admin
Password: *******
```

Voilà : notre instance est installée !
Intéressons-nous maintenant au fichier de configuration de Zope.

Fichier de configuration

Le fichier de configuration de Zope 2.7 a le mérite d'être particulièrement complet et clair (surtout pour les personnes qui ont connu les versions précédentes !).
Ce fichier de configuration, `zope.conf`, se situe dans le répertoire `etc` de notre instance, soit dans notre cas :

```
/home/zope/instances/plone1/etc/zope.conf
```

Ce fichier contient un grand nombre de sections et de paramètres, qui dépassent le cadre du présent ouvrage. Cependant, nous allons présenter les paramètres les plus importants à connaître.

ZOOM Les fichiers d'une instance Zope

La structure des fichiers d'une instance Zope est la suivante.

`/Extensions` – Les méthodes externes (*External Method*) de l'instance sont à déposer ici.

`/Products` – C'est dans ce répertoire que sont installés les produits Zope de l'instance. L'installation de Zope apporte déjà de nombreux produits standards, qui restent stockés dans le répertoire d'installation de Zope. Donc, même si le répertoire `Products` de l'instance reste vide, les produits standards de Zope sont présents.

`/bin` – Ce répertoire contient tous les exécutables de l'instance, notamment `zopectl` (ou `runzope.bat` pour MS-Windows), qui permet de démarrer et arrêter l'instance.

`/etc` – Ce répertoire contient les fichiers de configuration de Zope. Le plus important est `zope.conf`, détaillé plus loin dans cet ouvrage.

`/import` – C'est dans ce répertoire que l'on dépose les fichiers d'export Zope (`.zexp`) destinés à l'instance.

`/log` – Ce répertoire contient tous les fichiers de logs de l'application Zope. Le plus utile est `events.log`, qui va garder trace de tous les messages importants du serveur. La production des logs est configurée dans le fichier `etc/zope.conf` de votre instance.

`/var` – Ce répertoire contient les fichiers fondamentaux suivants, créés au premier lancement : `data.fs` (la base de données objet, fichier principal à sauvegarder), `Z2.log` (l'ensemble de toutes les requêtes traitées par le serveur Web de Zope, rarement utilisé) et `Z2.pid` (le numéro du processus Zope, si celui-ci est lancé).

debug-mode

- Usage (au choix) : `debug-mode on` (valeur par défaut) ou `debug-mode off`.
- Définit si Zope tourne en mode `debug` ou non.
- Il convient de laisser le mode `debug` pendant la phase de développement et de test, et de le retirer dès la mise en production.
- Attention : en mode `debug`, Zope est 2 à 6 fois plus lent qu'en mode normal !

effective-user

- Usage : `effective-user zope`.
- Spécifie le nom de l'utilisateur qui sera propriétaire du processus Zope, dans le cas où ce dernier est démarré par `root`. C'est utile pour le cas où les processus Zope sont lancés au démarrage de la machine.

locale

- Usage (au choix) : `locale fr_FR` (pour Debian) ou `locale fr_FR.ISO-8859-1` (pour FreeBSD).
- La locale est le nom de la page de code qu'utilise l'instance. En termes plus clairs, cette valeur désigne les jeux de langue avec lesquels on manipule le site ; pour pouvoir utiliser les caractères accentués du français, nous positionnons la locale sur le jeu français.

port-base

- Usage : `port-base 8000`.

- Définit les ports utilisés pour les serveurs HTTP, FTP et WebDAV en une seule commande.
- Les ports sont alors définis comme suit : port_base + 80 (HTTP), `port-base` + 90 (WebDAV) ou port-base + 21 (FTP).

datetime-format

- Usage (au choix) : `datetime-format international` (recommandé) ou `datetime-format us` (par défaut).
- Précise le format des dates telles que manipulées par Zope. Le format international est recommandé.

Pour les autres champs, le fichier `zope.conf` par défaut contient la liste exhaustive, sa lecture est très instructive.

Installer les produits nécessaires

Notre instance de Zope est maintenant prête. Installons-y les composants Zope nécessaires à notre site Plone.

Voici la liste des composants de base recommandés :

- `Plone 2.0` ;
- `Epoz 0.8.x` ;
- `ImageTag_CorePatch 0.3` ;
- `PloneArticle 2.0` ;
- `PloneExFile 2.0` ;
- `ZAttachmentAttributes 2.0` ;
- `ZAAPlugins 2.0` ;
- `PloneSearchBox 2.0` ;
- `PloneSiteMap 2.0`.

Pour les installer, il faut les télécharger sur Internet et les décompresser directement dans le répertoire `Products` de l'instance.

En tant qu'utilisateur `zope`, créons un espace pour télécharger les composants :

```
zope$ cd
zope$ cd instances/plone1
zope$ mkdir tmp
zope$ cd tmp
```

Toujours en tant qu'utilisateur `zope`, rapatrions les fichiers nécessaires :

```
zope$ wget http://umn.dl.sourceforge.net/sourceforge/plone/
  ➥CMFPlone-2.0.tar.gz
zope$ wget http://mjablonski.zope.de/Epoz/releases/Epoz-0.8.0.tar.gz
zope$ wget http://zope.org/Members/bowerymarc/ImageTag_CorePatch/0.3/
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
  ➥PloneArticle-2.0.tar.gz
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
  ➥PloneExFile-2.0.tar.gz
```

Bien entendu, ces URL seront amenées à être adaptées à la lumière des nouvelles versions proposées de ces composants. Le support papier possède là ses limites :-)

```
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
   ➥ZAttachmentAttribute-2.0.tar.gz
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
   ➥ZAAPlugins-2.0.tar.gz
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
   ➥PloneSearchBox-2.0.tar.gz
zope$ wget http://umn.dl.sourceforge.net/sourceforge/ingeniweb/
   ➥PloneSiteMap-2.0.tar.gz
```

Enfin, installons ces fichiers correctement dans le répertoire `Products` :

```
zope$ tar xvzf CMFPlone-2.0.tar.gz
zope$ mv CMFPlone-2.0/* ../Products
zope$ cd ../Products
zope$ tar xvzf ../tmp/Epoz-0.8.0.tar.gz
zope$ tar xvzf ../tmp/ImageTag_CoreTag-0.3.tar.gz
zope$ tar xvzf ../tmp/PloneArticle-2.0.tar.gz
zope$ tar xvzf ../tmp/PloneExFile-2.0.tar.gz
zope$ tar xvzf ../tmp/ZAttachmentAttribute-2.0.tar.gz
zope$ tar xvzf ../tmp/ZAAPlugins-2.0.tar.gz
zope$ tar xvzf ../tmp/PloneSearchBox-2.0.tar.gz
zope$ tar xvzf ../tmp/PloneSiteMap-2.0.tar.gz
```

Tous les fichiers nécessaires sont en place : démarrons (enfin) notre instance !

Démarrer l'instance

Démarrons notre instance, en tant qu'utilisateur `zope` :

```
zope$ instances/plone1/bin/zopectl start
daemon process started, pid 8789
```

On peut tester le site avec son navigateur (figure 8-1).

Pour redémarrer une instance depuis la ligne de commande (dans le cas de l'installation d'un produit par exemple) :

```
zope$ instances/plone1/bin/zopectl restart
daemon process restarted, pid 8876
```

Arrêter l'instance

Pour arrêter notre instance, en tant qu'utilisateur `zope`, il suffit de saisir les lignes suivantes :

```
zope$ instances/plone1/bin/zopectl stop
daemon process stopped
```

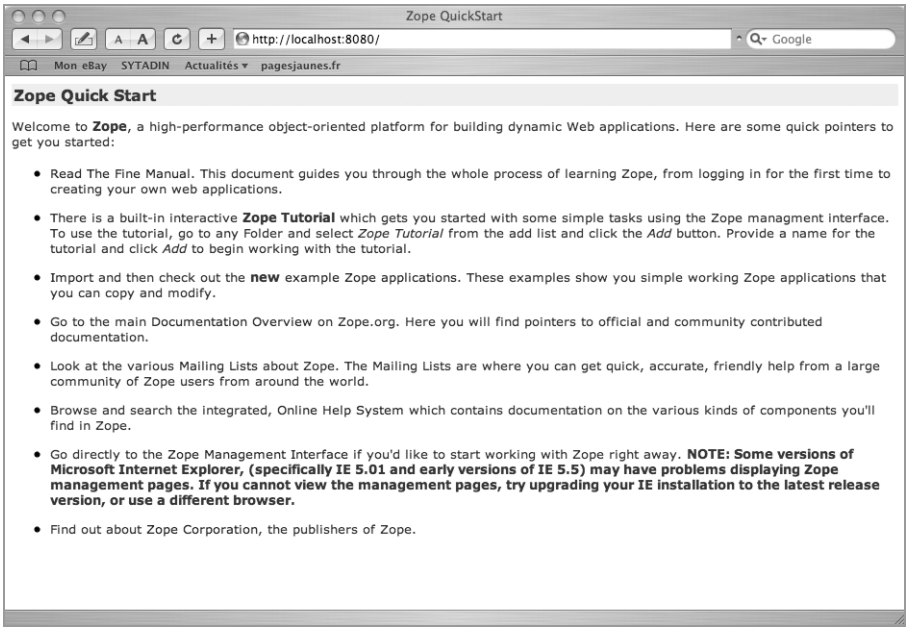


Figure 8–1
Test du site

Fichiers de démarrage

Pour faire en sorte que toutes les instances `zope` se mettent en route spontanément au redémarrage de l'ordinateur et s'arrêtent proprement à l'arrêt du serveur, il nous faut préparer un script adapté :

```
#!/bin/sh
ZOPEHOME=/home/zope
ZOPEINSTANCES=$ZOPEHOME/instances
for instance in `ls $ZOPEINSTANCES`
do
  $ZOPEINSTANCES/$instance/bin/zopectl $1
done
```

Ce script, intitulé `zope.sh`, est à placer dans `/home/zope` et doit être exécutable :

```
zope$ chmod a+x zope.sh
```

Enfin, pour ajouter ce script aux scripts de démarrage de notre serveur, tapez les lignes suivantes en tant que `root` :

```
$ cp /home/zope/zope.sh /etc/init.d/
$ chown root:root /etc/init.d/zope.sh
$ chmod 755 /etc/init.d/zope.sh
$ update-rc.d zope.sh defaults
```

Attention

Apache est traditionnellement un processus lancé à partir de `root`, qui *fork* (crée un processus fils) le serveur en `user` Nobody à son lancement. Si le compte Nobody est déjà utilisé, il est conseillé de créer un nouveau compte, spécifique à Apache, par exemple `apache`.

Frontal Apache

Nous allons utiliser Apache pour servir les pages Web du serveur Zope. Zope n'est alors utilisé que comme simple serveur d'application. Certaines subtilités sont détaillées plus loin dans ce chapitre.

La version d'Apache retenue est la version 1.3. À l'heure où nous écrivons ces lignes, la version 2 présente des problèmes dans son module de gestion du cache, qui l'écartent d'une mise en production.

Le répertoire de base recommandé est :

```
/usr/local/apache
```

Le propriétaire (*owner*) du répertoire doit être l'utilisateur `root`.

Compilation et installation

En tant qu'utilisateur normal, tapez les commandes suivantes :

```
$ wget http://apache.i-connexion.net/dist/httpd/apache_1.3.27.tar.gz
$ tar xvzf apache-1.3.27.tar.gz
$ cd apache-1.3.27.tar.gz
$ ./configure --prefix=/usr/local/apache --with-layout=Apache --
enable-module=all --enable-shared=max
```

Avec ces commandes, on prépare la compilation d'Apache avec les spécifications suivantes :

1. Installation dans `/usr/local`.
2. Utilisation du schéma de répartition des fichiers Apache ; tous les fichiers seront dans les répertoires d'Apache, notamment :
   - `conf` : fichiers de configuration ;
   - `htdocs` : fichiers HTML à servir ;
   - `proxy` : emplacement pour les fichiers temporaires du *proxy*-cache ;
   - `bin` : binaires d'Apache (dont `apachectl` et `httpd`) ;
   - `cgi-bin` : emplacement des scripts CGI, si nécessaire ;
   - `include` : les `.h` ;
   - `lib` : les librairies ;
   - `logs` : les fichiers de logs ;
   - `modules` : les modules Apache dynamiques.
3. Les modules suivants sont activés :
   - `RewriteRule` : permet de manipuler les URL des pages renvoyées par Zope ;
   - `Proxy` : permet de servir de *proxy* entre le serveur Zope et le client ;
   - `Headers` : permet d'ajouter des en-têtes aux requêtes HTTP servies par le *proxy* ;
   - `SSL` : le module SSL est inclus dans Apache 2.

Enfin, la compilation et l'installation se font avec :

```
$ make -DEAPI
$ sudo make install
```

La phase d'installation (`make install`) est opérée avec l'utilisateur `root` via `sudo` : c'est le seul utilisateur qui ait le droit d'écrire des fichiers dans le répertoire `/usr/local` de déploiement.

Configuration d'Apache

Le fichier `httpd.conf` fourni avec Apache (`/usr/local/conf/httpd.conf`) est à analyser et à corriger en fonction de la machine et des sites hébergés.
On peut notamment ajouter, en fin de fichier :

```
NameVirtualHost *
Include conf/www.monsite.com.conf
Include conf/www.monsite.com.ssl.conf
Include conf/www.monautresite.com.conf
```

Ainsi, on crée autant de fichiers de configuration que de `VirtualHost`, simplifiant la maintenance et les éventuelles migrations.

VirtualHosting HTTP

Voici un exemple de fichier `www.monsite.com.conf` :

```
<VirtualHost *>
      # Virtual Host Monster handling
      ServerName www.monsite.com
      ServerAlias monsite.com
      ServerAlias www2.monsite.com
      ServerAdmin support@ingeniweb.com
      RewriteEngine on
      RewriteCond %{HTTP:Authorization} ^(.*)
      RewriteRule ^(.*) http://localhost:8080/VirtualHostBase/http/
       ➥%{HTTP_HOST}:80/zopepath/to/www.monsite.com/VirtualHostRoot/
       ➥$1 [P]
      CacheRoot "/usr/local/apache/proxy"
      CacheSize 15000
      CacheGcInterval 2
      CacheMaxExpire 24
      CacheLastModifiedFactor 0.1
      CacheDefaultExpire 1
      CacheDirLength 2
      #
      # Expire - by request
      #
      ExpiresActive On
      ExpiresByType image/gif A86400
      ExpiresByType image/png A86400
      ExpiresByType image/jpeg A86400
```

```
        ExpiresByType text/css A86400
        ExpiresByType text/javascript A86400
        ExpiresByType application/x-javascript A86400
        CustomLog logs/www.monsite.com.access.log combined
        ErrorLog logs/error.log
</VirtualHost>
```

Les éléments importants à noter sont :

- `RewriteRule` : responsable de l'échange des informations avec Zope ; c'est la fonction *proxy* d'Apache ;
- `CacheRoot` : activation de la fonction cache d'Apache ; toutes les pages déjà servies par Apache et qui sont demandées par un navigateur – pas forcément le même – sont alors servies directement par Apache, sans solliciter Zope ;
- `CustomLog` : création des fichiers de log, pour analyse par un outil adapté, afin de mesurer la fréquentation du site.

OUTILS Analyse des logs

Le lecteur curieux regardera du côté des excellents Webalizer :
▸ http://www.mrunix.net/webalizer/
et AWStats :
▸ http://AWStats.sourceforge.net

Validation de la configuration

Avant de démarrer Apache, il est recommandé de tester la configuration avec :

```
/usr/local/apache/bin/apachectl configtest
```

Les règles de `VirtualHosting` sont analysables grâce à :

```
/usr/local/apache/bin/apachectl -S
```

Démarrage d'Apache

Pour démarrer le serveur HTTP, en tant qu'utilisateur `root`, utilisez la commande suivante :

```
/usr/local/apache/bin/apachectl start
```

Arrêt d'Apache

Pour arrêter le serveur, en tant qu'utilisateur `root`, saisissez la ligne suivante :

```
/usr/local/apache/bin/apachectl stop
```

Intégration Zope/Apache

Pour parfaire notre installation, il reste une étape cruciale : l'instanciation d'un objet de type `VirtualHostMonster`.

Il faut installer dans le serveur Zope, à la racine, un objet du type `VirtualHostMonster`. L'*ID* de l'objet importe peu.

C'est cet objet qui va être responsable de la bonne compréhension des ordres passés par Apache.

Voilà ! Nous avons une solide installation de Zope, avec son frontal Apache. Cette installation est simple à maintenir, sécurisée et évolutive.

En résumé…

Notre site est en production ! Nous avons installé toute la plate-forme depuis un système d'exploitation fraîchement déployé. Nous avons mis en place une infrastructure saine sur laquelle nous pouvons capitaliser, par exemple en ajoutant simplement de nouvelles instances, si nécessaire.

Le frontal Apache que nous avons mis en production nous garantit une capacité à la montée en charge bien contenue.

Mais comment nous comporter dans le cas d'un site à très forte charge ou dans le cas d'un site à très haute disponibilité ? Une seule réponse : ZEO, que nous présentons dans le chapitre suivant.

Montée en charge

9

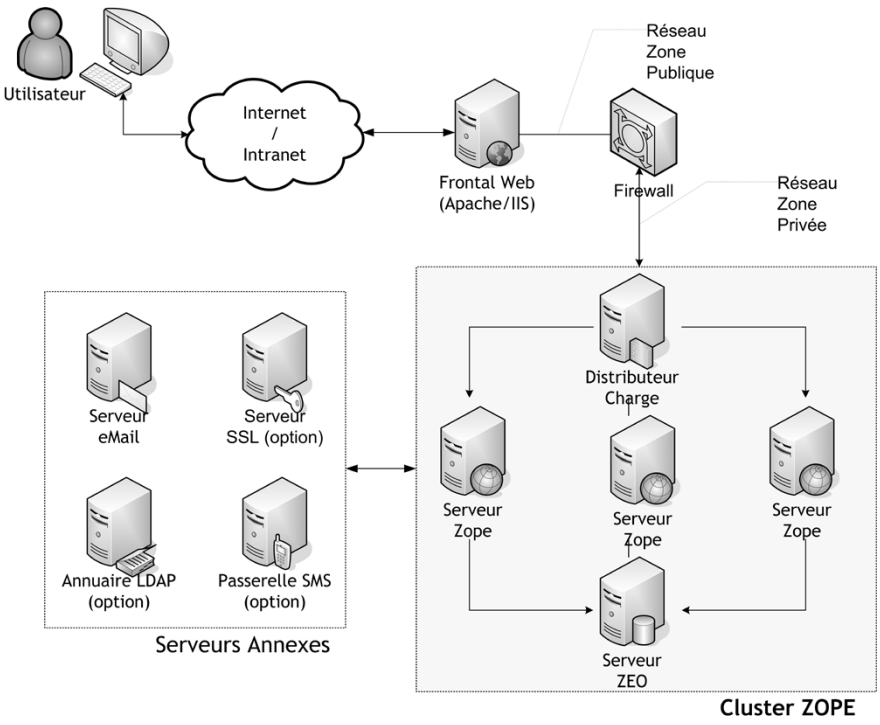
Zope Plone

SOMMAIRE

MOTS-CLÉS

- Montée en charge
- Haute disponibilité
- Robustesse

Montée en charge | haute disponibilité | robustesse



Votre site intranet et les différents services qu'il fournit sont maintenant fonctionnels. La dernière étape du projet consiste à prendre en compte les aspects de la gestion des performances et de la montée en charge, et à mettre en œuvre les solutions adéquates.

Objectifs et solutions

Dans la plupart des cas, pour un site à fréquentation moyenne (< 20 000 pages/jour), ces conseils ne sont pas utiles. Cependant, et pour fournir une solution complète, ce chapitre met en lumière les solutions techniques d'amélioration des performances et de capacité à la montée en charge.

Les différentes solutions présentées sont :

- l'utilisation d'un serveur HTTP spécialisé comme frontal à Zope : Apache (http://www.apache.org) ;
- l'utilisation d'un serveur proxy-cache comme frontal à Zope : Squid (http://www.squid-cache.org) ;
- l'optimisation de la CPU et de la bande passante grâce aux gestionnaires de cache de Zope (Zope Cache Managers) : RAM Cache Manager, HTTP Accelerated Cache Manager ;
- la mise en œuvre d'un cluster de serveurs Zope : option Zope Enterprise Objects (ZEO).

Composantes d'un service à hautes performances

Serveur frontal HTTP robuste

Les principaux serveurs web recommandés sont :

- Apache ;
- Microsoft IIS.

Apache est à ce jour le serveur web le plus utilisé sur Internet (près de 60 % de parts de marché). Pour surveiller la popularité des serveurs web :

▸ http://www.netcraft.com/survey/ !

Zope est un serveur d'applications disposant d'un serveur HTTP interne. Néanmoins, il est vivement recommandé pour les environnements de production de lui adjoindre un serveur HTTP spécialisé, notamment lorsqu'on doit servir un nombre important de pages statiques et/ou d'images à partir du système de fichiers du serveur. On parle de serveur web *proxy*. Cette approche permet de bénéficier des fonctionnalités du serveur web choisi, et de laisser Zope simplement jouer son rôle de serveur d'applications.

Optimisation des ressources

Le traitement de certaines requêtes HTTP pénalise la CPU et la bande passante du serveur. On parle de requêtes ou de traitements « coûteux ». Le mécanisme d'optimisation consiste à enregistrer les résultats d'une telle requête dans une zone de mémoire, ou « cache », la première fois qu'elle survient, et à servir le contenu à partir de cette zone chaque fois que la même requête est reçue. Ainsi, le serveur n'est sollicité qu'une fois pour refaire le même traitement coûteux.

Le gestionnaire du mécanisme de cache est paramétrable pour différents comportements ou besoins : limitation de la taille allouée au cache en mémoire vive ou sur disque, types d'objets à prendre en compte, différentes stratégies d'optimisation, etc.

Zope fournit deux gestionnaires de cache pour permettre aux concepteurs de sites d'améliorer les performances selon deux stratégies d'optimisation différentes :

- RAM Cache Manager pour l'optimisation de la CPU ;
- HTTP Accelerated Cache Manager pour l'optimisation de la bande passante.

Capacité à la montée en charge

Pour un site à forte fréquentation ou à fort besoin de ressources (CPU, base de données, calculs, etc.), un seul serveur peut ne pas suffire à absorber la charge. Voici les objectifs d'un tel site :

- Avoir une disponibilité de 99,99 % : le site doit être disponible au maximum. Si une courte panne ou une incapacité survient, c'est gênant, mais pas vital pour le site ou l'entreprise.
- Pouvoir accepter un grand nombre de clients simultanés, sans que la qualité ne se dégrade.
- Supporter des forts pics de charge : le site doit continuer à réagir de façon fluide, même dans le cas de fortes sollicitations passagères, par exemple dans le cadre d'un événement, dans le cas d'une publication presse ou *slashdot*, etc.

Pour ces différents objectifs, outre la solution d'optimisation du code exécuté par les serveurs, la solution de *cluster* est idéale : chaque machine de la grappe va assumer une partie de la charge nécessaire.

Haute disponibilité

Il existe des sites pour lesquels la disponibilité systématique est critique. Par exemple, pour des entreprises dont l'activité est à 100 % basée sur un service Internet, ou bien dans des métiers où la disponibilité d'un service est vitale : aéronautique, hôpitaux, etc.

Ces sites ont un besoin de 100 % de disponibilité. Les coupures de service ne sont pas acceptables dans ce contexte.

Pour ces besoins, ZEO et la mise en *clusters* permettent de répartir le risque de panne matérielle, en multipliant les serveurs d'applications Zope. Le risque est divisé par le nombre de machines mises à disposition. Bien entendu, l'ensemble de la chaîne doit être sécurisé de la même façon : redondance électrique, réseau, etc.

Presque toutes les solutions présentées peuvent cohabiter, et elles sont souvent complémentaires. En effet, on ne peut pas facilement utiliser à la fois Squid et Apache : dans 99 % des cas, il faut choisir entre ces deux solutions.

Récapitulatif

	FrontaHTTP	Optimisation CPU	Optimisation Bande Passante	Haute Disponibilité	Montée en Charge
Apache	X				
Squid	X		X		X
Zope Cache Managers		X (RAM C.M.)	X (HTTP Accelerated C.M.)		
ZEO		X		X	X

Les outils du Zopemaster

1. Les outils de mesures et de tests des performances suivants sont incontournables pour le webmestre Zope professionnel :
   - Apache Bench est un utilitaire de test fourni avec la distribution d'Apache (dans `/apache/bin`). Il permet de tester les performances du serveur en spécifiant le nombre de requêtes à lui envoyer et le nombre maximal de connexions simultanées. La syntaxe est du type :

     ```
     ab -n 1000 -c 100 http://www.mondomaine.com/news
     ```

   - Call Profiler est un utilitaire développé par Richard Jones (http://www.zope.org/Members/richard/CallProfiler). Il permet de connaître les temps d'exécution de tous les scripts et méthodes intervenant dans une requête afin d'aider à repérer d'éventuels scripts coûteux.
2. Les sites de la communauté Open Source vous fournissent les *howtos*, les archives des listes de diffusion, et vous tiennent au courant des nouveautés vous permettant d'améliorer votre service ou des bogues existants et corrigés. Les pointeurs suivants sont donc indispensables au webmestre :
   - forum et site communautaire Zope francophone, Zopera : http://www.zopera.org ;
   - informations et mises à jour de Zope et CMF : http://www.zope.org et http://cmf.zope.org/ ;
   - informations et mises à jour de Plone : http://plone.org ;
   - liste de diffusion du CMF : http://lists.zope.org/mailman/listinfo/zope-cmf ;
   - exemples de codes, astuces et conseils : http://www.zopelabs.com/ et http://www.zopezen.org ;
   - O'Reilly Network – Apache DevCenter : http://www.onlamp.com/apache/.

Utiliser Apache comme serveur web frontal à Zope

Ajouter un serveur web tel que Apache à l'infrastructure de mise en production d'un site Zope permet de combiner les forces de Zope pour le traitement de pages dynamiques avec les nombreux avantages d'Apache qui sont :

- servir du contenu statique (fichiers HTML et images) à partir du système de fichiers ;
- la gestion de domaines virtuels et la réécriture d'URL ;
- la prise en charge de SSL (*Secure Sockets Layer*) pour le commerce électronique.

Nous avons vu les détails de l'implémentation dans le début de ce chapitre.

B.A.-BA Apache

Apache est un logiciel libre qui fournit le service de « serveur web ». Un serveur web est responsable de la fourniture des pages HTML demandées à un client : le navigateur web.

B.A.-BA SSL

SSL (*Secure Sockets Layer*) est un protocole permettant de sécuriser les transactions effectuées via Internet. Il repose sur un procédé de cryptographie garantissant la sécurité de la transmission de données sur Internet.
Un serveur sécurisé par SSL possède une URL où le préfixe http:// a été remplacé par https://.

Apache sur MS-Windows

Il est également recommandé d'utiliser Apache comme serveur web si vous utilisez MS-Windows comme plate-forme de production. Vous trouverez toutes les informations sur la version d'Apache pour MS-Windows à partir de l'adresse :

- http://httpd.apache.org/docs/windows.html.

Les versions multitâches de MS-Windows (NT, 2000, XP) sont à préférer aux versions précédentes (95, 98, Me).

Documentation sur Apache

Quelques pointeurs :

- http://httpd.apache.org/docs/install.html.
- http://www.onlamp.com/pub/a/apache/2000/02/24/installing_apache.html.

Utiliser Squid comme serveur proxy-cache avec Zope

Configurer Squid

B.A.-BA Serveur proxy-cache

Un serveur proxy-cache permet de décharger les serveurs web de la remise de pages fréquentes. Ce système est notamment très utile pour les images qui constituent la charte graphique d'un site : celles-ci sont visibles sur toutes les pages du site. Avec un proxy-cache, les images ne sont demandées qu'une seule fois au serveur web, les requêtes suivantes sont directement servies par le proxy-cache.

Téléchargez la dernière version stable de Squid à partir de l'adresse http://www.squid-cache.org, puis procédez à l'installation de l'archive. Selon votre configuration spécifique (RedHat, Debian ou autre), cette procédure va installer tous les fichiers exécutables et de configuration de Squid dans un répertoire du type `/usr/local/squid`.

Pour éviter que Squid ne s'exécute en tant qu'utilisateur `root`, ce qui présente des risques de sécurité pour le site, éditez le fichier de configuration

Alternatives à Squid

Il est possible, à la place de Squid, d'utiliser le module de cache du serveur Apache ou de mettre en œuvre une solution commerciale comme Netscape Proxy Server ou les produits dédiés de Cisco Systems.

`squid.conf` présent dans `/usr/local/squid/etc` et enlevez le caractère # précédent les deux lignes suivantes pour les activer :

```
cache_effective_user nobody
cache_effective_group nobody
```

Paramétrez le cache et les fichiers journaux (logs). Pour définir une taille du cache de 100 Mo, modifiez la directive `cache_dir` du `squid.conf` comme suit :

```
cache_dir /usr/local/squid/cache 100 16 256
```

Bien entendu, il faut adapter la taille du cache à la taille de votre site. Il est recommandé d'utiliser le double de la taille qu'utilise votre site web sur le disque. Utilisez les commandes suivantes pour créer le répertoire de cache, nommé `cache` :

```
$ cd /usr/local/squid ; mkdir cache
$ chown -R nobody.nobody cache
$ cd /usr/local/squid/bin
$ ./squid -z
```

Utilisez les commandes suivantes pour créer le répertoire des journaux, nommé `logs` :

```
$ cd /usr/local/squid ; mkdir logs
$ chown -R nobody.nobody logs
```

L'installation place dans le sous-répertoire `/usr/local/squid/bin` un script Shell nommé `RunCache` chargé de maintenir la fonction de cache active. Démarrez Squid à l'aide de la commande :

```
$ /usr/local/squid/bin/RunCache &
```

Plus tard, vous pourrez ajouter cette ligne à votre fichier `/etc/rc.local` par exemple, afin que le serveur Squid soit démarré au boot de la machine.

Enfin, paramétrez Squid en mode d'accélération HTTP. Il s'agit de configurer Squid pour qu'il reçoive et transfère toutes les requêtes HTTP vers un port différent du port HTTP par défaut (80), sur lequel s'exécuterait le véritable serveur web. Modifiez les directives suivantes dans le fichier `squid.conf`, après les avoir « dé-commentées » si nécessaire :

```
http_port 80                              1
http_access allow all                     2
httpd_accel_host www.mondomaine.com       3
httpd_accel_port 8080                     4
```

Grâce à la première directive 1, le serveur proxy s'exécutera sur le port 80, le port standard des requêtes HTTP.
La ligne 2 gère le contrôle d'accès. Par défaut, l'accès est interdit à tout le monde via la directive `http_access deny all`. Il faut donc la modifier pour que les autres utilisateurs puissent y accéder.
Les lignes 3 et 4 spécifient le nom DNS de la machine du serveur à accélérer et le port sur lequel ce dernier s'exécute. En effet, par conception, le serveur Zope fonctionne sur le port 8080.

Configuration côté Zope

Ajouter l'objet de correspondance des domaines (VirtualHostMonster)

Comme pour la configuration avec Apache, l'objet `VirtualHostMonster` à la racine de votre structure Zope est suffisant.

Utiliser les gestionnaires de cache de Zope

RAM Cache Manager

Dans un site Plone, la plupart des pages dynamiques (page d'accueil, index internes, pages listant les membres, résultats de recherche, etc.) s'appuient sur des requêtes vers le moteur d'indexation (l'outil `portal_catalog`). Ces pages dynamiques peuvent imposer un temps de traitement coûteux en utilisation du CPU. Il faut donc les optimiser et leur appliquer une solution de cache. Le gestionnaire de cache spécifique appelé `RAM Cache Manager` est justement fait pour ce cas de figure. Il permet, de manière transparente, de cacher les pages rendues à partir des objets Zope : page templates, méthodes DTML, scripts Python, en mémoire RAM.

Typiquement, vous allez associer la page d'accueil, dont le contenu est fourni par l'objet de type Document nommé `index_html`, à l'objet RAM Cache Manager de Plone. Celui-ci est déjà créé à l'installation du site et nommé `RAMCache`. Pour cette configuration, vous pouvez procéder comme suit :

1. Renommez le `documentindex_html` en `frontpage`, par exemple.
2. Créez un script Python que vous nommez `index_html`. Celui-ci va être traité comme l'index lorsqu'on accède à la page d'accueil du site, le but étant qu'il renvoie le contenu de `frontpage` de manière à pouvoir stocker ce contenu en cache.
3. Finalement, accédez à l'interface de gestion de l'objet `RAM Cache Manager` à la racine du site. L'onglet `Associate` vous permet de sélectionner le type `script (Python)` dans la zone de sélection des types d'objets à associer au gestionnaire. Lorsque vous cliquez sur le bouton `Locate`, l'interface vous affiche la liste des scripts Python que vous pouvez associer au gestionnaire. Il ne vous reste plus qu'à sélectionner l'objet `index_html`, et à cliquer sur le bouton `Save Changes`.
4. Votre nouvelle page d'accueil est maintenant prise en compte par le gestionnaire de cache en RAM de Zope. Vous pourrez vous rendre compte du gain de temps de chargement lorsqu'on charge la page, et ce dès la seconde fois.

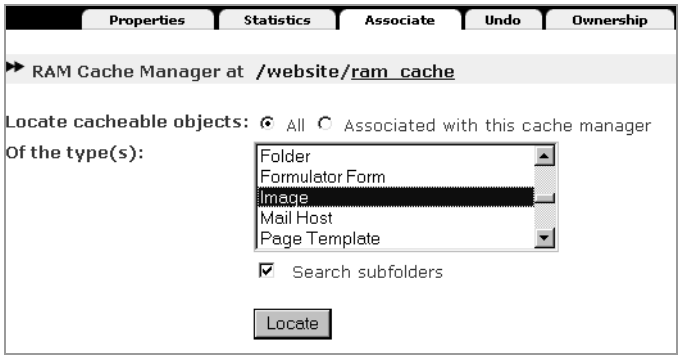


Figure 9–1
Interface du gestionnaire de cache de Zope

Vous pouvez appliquer la même stratégie aux autres pages du même type, telles que la page d'affichage de la liste des membres du site (également nommé « roster »).

HTTP Accelerated Cache Manager

Mise en garde

La mise en œuvre du cache Zope doit être faite de manière réfléchie. Il ne faut pas avoir à mettre trop de choses dans le cache, seuls les objets qui changent peu sont conseillés, sinon vous vous retrouverez avec le même problème parce que le gestionnaire de cache, à son tour, consommera trop de ressources.

Au sein de votre site, vous pouvez également avoir un nombre important d'images telles que les photos des membres du site, affichés dans la page `roster` de Plone, ou qu'une galerie d'images. Le chargement de ces images est coûteux en bande passante, d'autant plus que ce sont des éléments statiques, Zope mettra systématiquement en branle sa machinerie pour traiter les requêtes d'affichage des images, alors que leur fréquence de changement est rare. Encore un fois, Zope est optimisé pour le traitement de pages dynamiques, et non statiques.

L'autre gestionnaire de cache de Zope, *HTTP Accelerated Cache Manager*, permet de résoudre cette problématique.

De la même manière que pour le *RAM Cache Manager*, il suffit d'aller sur l'objet `HTTP Accelerated Cache Manager` à la racine du site Plone, puis de lui associer tous les objets `Image` du site à prendre en compte (onglet `Associate`).

Haute disponibilité et montée en charge avec ZEO

B.A.-BA Zope Enterprise Objects

ZEO (*Zope Entreprise Object*) est la solution fournie avec Zope qui permet de construire très simplement des clusters de serveurs Zope, pour augmenter la disponibilité ou la capacité à la montée en charge d'un site Zope.

Le besoin de faire un cluster de serveurs Zope peut surgir dans deux cas de figures principaux :

- pour des « gros » sites (montée en charge) ;
- pour des sites « critiques » (haute disponibilité).

Dans ces deux cas, on remarque que la solution est bien de multiplier les serveurs, et donc de disposer d'une solution de *clustering*.

Difficultés de mise en œuvre

Multiplier les serveurs web est une tâche aisée pour des sites statiques (pages HTML sur disque, servies par Apache par exemple) : il suffit de copier les fichiers statiques sur toutes les machines, et de démarrer les serveurs web.

Pour un site dynamique, la tâche est plus complexe :

- tous les serveurs doivent avoir les mêmes données ;
- si un serveur modifie un objet , il faut notifier les autres.

On est bien face à l'épineux problème de la synchronisation des données de plusieurs serveurs.

La solution ZEO

ZEO est donc la solution pour résoudre ces différentes problématiques avec Zope.

Fonctionnalités de ZEO

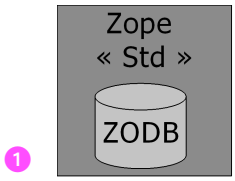
ZEO permet donc de faire des *clusters* Zope. Les fonctionnalités et avantages de cette solution sont les suivants :

- **Haute disponibilité :** si un des serveurs du cluster cesse de fonctionner, le site fonctionne toujours.
- **Montée en charge** : en divisant la charge par le nombre de serveurs, le site est capable d'accepter plus de clients simultanés et ainsi de faire face à la montée en charge.
- **Distribution de la charge** : avec plusieurs serveurs, et si la répartition de charge est conçue intelligemment, la répartition de l'activité des différents serveurs permet une plus forte réactivité du site.
- **Mise à disposition de ressources temporaires** : en cas d'événement prévisible (publication presse, événementiel, élections) ou pour faire face à une montée en charge brutale, il « suffit » d'ajouter des machines au *cluster* pour voir la disponibilité globale du site augmenter. Ces serveurs peuvent ensuite être supprimés après le passage du pic.
- **Distribution des serveurs** : en multipliant le nombre de machines, on a une diminution du risque « physique » pour les serveurs, et donc une augmentation de la disponibilité théorique. La probabilité que toutes les machines tombent en panne en même temps diminue fortement si on augmente le nombre de machines !
- **Intégrité transactionnelle** : il existe des règles permettant de régler les problèmes de mise à jour simultanée de l'information par deux serveurs concurrents.

Fonctionnement de ZEO

Comparaison entre l'architecture Zope standard et l'architecture ZEO

Schéma du cas « Zope Standard » 1

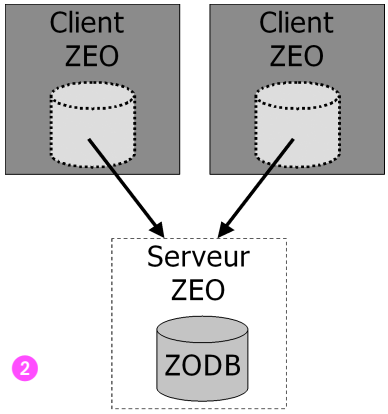


1

Le processus serveur Zope est « propriétaire » de la base de données objet ZODB.

Schéma du cas « Zope avec ZEO » 2

La base objet ZODB est portée par un serveur dédié, appelé Serveur ZEO. Ce serveur est utilisé par les clients ZEO, qui forment le serveur d'applications.



2

Supposons qu'on dispose de *N* machines à mettre en *cluster*. Ces serveurs seront utilisés de la façon suivante :

- 1 serveur de stockage : ZEO Server, contenant la ZODB ;
- *N*-1 clients ZEO. Ces machines se comportent comme des serveurs Zope standards. Ce sont ces machines qui servent les pages web.

On le voit, seule une machine est propriétaire de la base objet ZODB qui contient toutes les données. Toutes les autres instances Zope sont des clients ZEO, qui viennent chercher auprès du serveur ZEO les informations dont elles ont besoin. Grâce au système de cache de Zope, les conversations entre les clients et le serveur sont réduites au strict nécessaire.

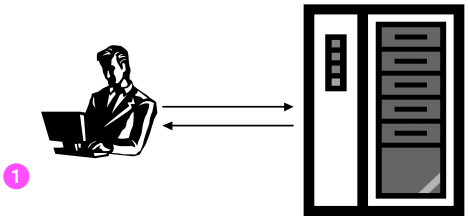
Mise en œuvre

Prérequis

Pour installer notre *cluster*, nous devons disposer de *N* machines. Si possible, et pour plus d'homogénéité, ces *N* machines utiliseront le même système d'exploitation, mais ce n'est absolument pas obligatoire. L'utilisation de machines Unix est fortement recommandée, même si ZEO fonctionne également sous MS-Windows.

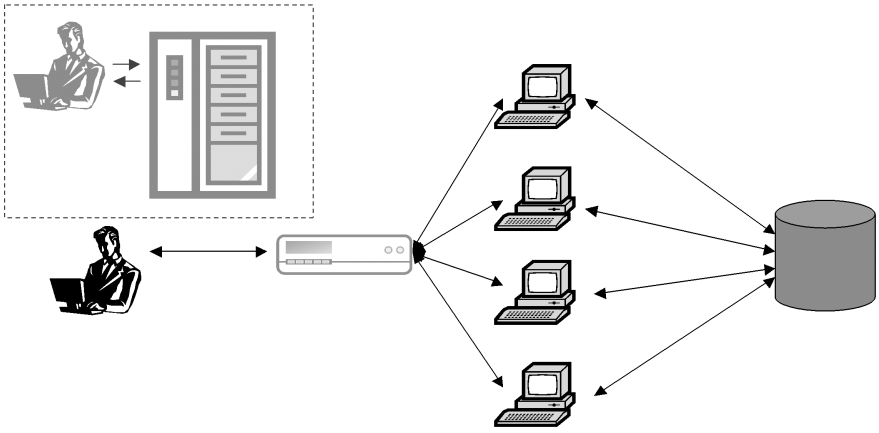
Schémas d'architecture

Un utilisateur se connecte à un serveur web 1.



Ce serveur web est en fait un *cluster* Zope/ZEO 2.

Voici le détail des échanges : en fait, la seule machine que l'utilisateur « voit » est la machine chargée de la distribution de la charge. Nous verrons plus loin les détails de son implémentation. Cette machine se charge de désigner le client ZEO qui va servir la page demandée par le client. Une des machines du cluster se charge donc du calcul de la page demandée, en se référant si nécessaire (ou dans son propre cache) aux données disponibles sur la machine serveur ZEO. Ainsi, la disponibilité est bien améliorée, et la charge que le système peut supporter est bien augmentée.



Sur ces *N* machines, Zope sera installé dans un répertoire que nous noterons `$ZOPE`.

Il faut installer strictement la même version de Zope sur ces différentes machines. ZEO ne vérifie pas ce point, et un non-respect de cette règle peut amener à des pertes de données massives.

Également, les mêmes produits Zope doivent être installés sur ces *N* machines.

Enfin, toutes les machines doivent avoir un accès identique aux ressources nécessaires à l'application : bases SQL, serveur mail, etc.

Le fichier `Data.fs` (image de la base ZODB) ne sera installé que sur la machine serveur ZEO. Les *N*-1 machines ZEO clientes n'ont pas de fichier `Data.fs`.

Installation

La dernière version de ZEO est maintenant livrée avec Zope, depuis la version 2.7. Il convient donc de suivre la documentation d'installation du chapitre précédent, pour tous les serveurs physiques. Seule l'installation du ZEO Server est un petit peu différente.

Démarrage ZEO Server

L'installation de ZEO Server se fait au moment de la création de l'instance Zope. Nous allons en créer une spécialement conçue pour être ce serveur.

En tant qu'utilisateur Zope, sur le serveur qui va être ZEO Server, entrez les commandes suivantes :

```
zope$ cd
zope$ mkdir instances
zope$ Zope2.7/bin/mkzeoinstance.py --dir=instances/zeoserver 8070
```

Pour démarrer le serveur, on utilise simplement (en tant qu'utilisateur zope) :

```
zope$ cd
zope$ instances/zeoserver/bin/zopectl start
```

Cette commande va démarrer le serveur ZEO, qui va attendre ses ordres sur le port 8070 de la machine serveur.

Démarrage ZEO client

Sur les machines clientes, la seule différence dans l'installation concerne le fichier `etc/zope.conf`, dans lequel il faut enlever la partie :

```
<zodb_db main>
    # Main FileStorage database
    <filestorage>
      path /home/zope/instances/plone1/var/Data.fs
    </filestorage>
    mount-point /
</zodb_db>
<zodb_db temporary>
    # Temporary storage database (for sessions)
    <temporarystorage>
      name temporary storage for sessioning
    </temporarystorage>
    mount-point /temp_folder
    container-class Products.TemporaryFolder.TemporaryContainer
</zodb_db>
```

pour la remplacer par :

```
<zodb_db main>
  mount-point /
  <zeoclient>
    server zeoserverhostname:8070
    storage 1
    name zeostorage
    var /home/zope/instances/plone1/var
  </zeoclient>
</zodb_db>
```

Enfin, le démarrage du (des) client(s) ZEO (et donc du serveur d'applications Zope), se fait « normalement » avec :

```
zope$ instances/plone1/bin/zopectl start
```

Voilà ! Vos clients ZEO communiquent avec votre serveur ZEO, de manière transparente. Un seul serveur est responsable du stockage, et *N*-1 machines sont disponibles pour calculer le rendu des pages.

Connexion à un client

Si on se connecte avec un navigateur à l'un des clients ZEO (qui est donc l'un de nos *N*-1 serveurs Zope maintenant), on constate que son `Control Panel` a un petit peu changé, notamment la rubrique `DataBase`.

En effet, on note que la machine qui porte la base ZODB et le numéro de port sont spécifiés : on est bien connecté à un serveur ZEO.

Le cas des serveurs multiprocesseurs

Si une des machines ZEO clientes est un serveur multiprocesseur, il est fortement conseillé de démarrer autant de processus clients que de CPU dans la machine. En effet, Python possède un *lock* global au processus, ce qui fait que Zope n'utilise pas pleinement les capacités d'un tel serveur. En utilisant *N* clients ZEO sur cette machine, on a la certitude que chaque CPU fera tourner une instance de Zope pleinement.

Limites et dangers de ZEO

La programmation d'un serveur avec ZEO impose tout de même quelques précautions…

Limites d'utilisation

En respect du modèle web, ZEO est très efficace si le site respecte le ratio 90/10 : 90 % de lecture d'information pour 10 % d'écriture. Au-delà, si le site doit écrire plus d'informations dans la ZODB, on pénalise le système en

augmentant le nombre de conflits d'accès concurrents. Dans ce cas, stocker plus de données dans une base SQL supportant bien les écritures multiples et concurrentes est une solution.

Dangers applicatifs

L'essentiel des dangers applicatifs se résume simplement : étant donné que deux requêtes consécutives de l'utilisateur n'utilisent par forcément le même serveur, il faut faire attention aux variables qui ne sont valables que dans le cadre d'un serveur.

Sessions

Un bon exemple de situation à risque est l'utilisation de sessions en RAM (cas du composant de session livré avec Zope). En effet, imaginons qu'une machine Z1 traite la première requête client et stocke des informations dans une session en RAM. Si la seconde requête du client est traitée par une machine Z2, alors celle-ci n'aura pas en RAM les informations nécessaires. Une façon simple de résoudre ce problème est d'utiliser un composant de gestion des sessions dans une base SQL commune par exemple.

B.A.-BA **Sessions**

Dans le contexte des sites web dynamiques, une session permet de transporter des informations liées à un internaute pendant la durée de sa navigation sur le site. Une session se définit donc par un identifiant unique, une durée de vie, et un ensemble de variables contenant les informations qu'elle transporte.

Temporary Folder

Comme pour les sessions, le composant `temp_folder` fourni avec Zope stocke ses informations dans la RAM du serveur qu'il exécute. Pour les mêmes raisons, son usage est déconseillé avec ZEO.

ATTENTION **Sécurité**

Le serveur ZEO n'est en rien sécurisé. Le port qu'il ouvre sur la base ZODB n'est validé par aucune règle de sécurité. Si, par malheur, un pirate venait à connecter un client ZEO sur votre serveur ZEO, il pourrait en faire ce que bon lui semble.
Pour éviter des désagréments majeurs, il faut utiliser la sécurité de votre réseau, par exemple avec un pare-feu en amont. Une solution habituelle est de laisser le répartiteur de charge dans la partie publique du réseau (ou dans la DMZ), et de placer les *N*-1 clients ZEO dans un partie protégée, sur une classe d'adresses non routables (sur un réseau local).

Fichiers sur disque

Un autre cas de danger pour l'application est de faire des manipulations de fichiers sur le serveur d'applications. En effet, à moins d'utiliser un système de fichiers distribué ou partagé (NFS par exemple), on ne retrouvera pas les mêmes fichiers d'une machine serveur à l'autre.

Ce point est problématique dans le cas d'utilisation de la base de données GadFly, et plus généralement des composants Zope qui stockent leurs informations sur le disque du serveur (souvent dans le répertoire `$ZOPE/var`).

Pour pallier ce problème, il faut soit utiliser un système de disque partagé, comme NFS sous Unix, soit trouver des composants qui n'utilisent pas ce principe.

B.A.-BA **Sécurité**

Pare-feu ou *firewall***.** Système informatique permettant de protéger un réseau local d'entreprise des agressions provenant d'Internet.
DMZ (*De-Militarized Zone*). Réseau ajouté entre le réseau interne (protégé) et Internet, la DMZ fournit une couche de sécurité supplémentaire.

Distribution de charge

Il est temps d'aborder les différentes solutions qui permettent de faire la répartition de la charge.

Il existe différentes réponses usuelles, plus ou moins simples à mettre en œuvre, et plus ou moins heureuses.

Choix du miroir par l'utilisateur

C'est la solution la moins chère : elle consiste à proposer à l'utilisateur de cliquer sur l'URL du client ZEO sur la page d'accueil. Cette solution est très limitée et n'apporte pas vraiment de réponse satisfaisante, mais elle permet de faire les premiers tests.

DNS Round Robin

Cette solution est assez courante, notamment pour les sites statiques. Elle consiste à placer la répartition de la charge au niveau du DNS. Ainsi, lorsque les utilisateurs demandent pour la première fois à voir la page www.mondomaine.com, le DNS va leur donner l'adresse IP de l'un des clients ZEO, directement. Les adresses distribuées par le serveur DNS sont équiréparties (distribution séquentielle). Cette solution est très simple, mais présente deux inconvénients majeurs.

- La répartition n'est pas une répartition de charge. Si la machine désignée par le DNS est déjà chargée par d'autres utilisateurs intensifs, la qualité du service rendu se dégrade.
- Et surtout, si la machine serveur tombe en panne, l'utilisateur ne pourra plus se connecter, son ordinateur ayant mis en cache l'adresse IP du client ZEO pour plusieurs heures.

Pour mettre en œuvre cette solution, il suffit d'utiliser un fichier DNS de la forme :

```
www 300 IN A 10.0.0.1
    300 IN A 10.0.0.2
    300 IN A 10.0.0.3
    300 IN A 10.0.0.4
```

Ainsi, l'IP demandée pour la machine `www` sera tour à tour `10.0.0.1`, `10.0.0.2`, `10.0.0.3` puis `10.0.0.4`, suivant l'ordre de requêtes des différents clients.

Serveur proxy web

Un serveur *proxy* est un service qui reçoit des requêtes web de clients, les passent à une tierce machine, pour retourner la réponse au client. C'est ce que fait Apache lorsqu'il est installé en frontal de Zope de manière classique.

C'est ce même principe, un peu élaboré, qui va nous servir pour réaliser un système de répartition de charge à la fois simple et économique.

Notre système est basé sur Apache, qui va recevoir les requêtes des clients, et se charger de les transmettre à l'un des *N*-1 clients ZEO.

B.A.-BA DNS

DNS (*Domain Name System*) est un service fondamental des réseaux TCP/IP en général, et d'Internet en particulier. C'est le service qui est responsable de la correspondance entre le nom de la machine (ex. : www.zopera.org) et l'adresse IP unique qui identifie la machine sur le réseau. Pour le service inverse, on parle de reverse-DNS.
Le serveur DNS le plus utilisé d'Internet est un logiciel libre nommé BIND.

▸ http://www.isc.org/products/BIND/

Pour ce faire, il faut commencer par ajouter le fichier `Balance.py`, dans le répertoire `/usr/local/bin` par exemple :

```
#!/usr/bin/env python
import sys, string
count=0
SERVERS=['www1','www2','www3']
NB_SERVERS=len(SERVERS)
URL="http://%s.mondomaine.com:8080/VirtualHostBase/http/www.
mondomaine.com:80/%s"
def translate(data):
  global count
  count=(count+1) % NB_SERVERS
  return URL % ( SERVERS[count], data )
if __name__=="__main__":
  data=string.strip(sys.stdin.readline())
  if not data:
    break
  sys.stdout.append(data)
  sys.stdout.flush()
```

Ce petit programme se contente de transformer les chemins qu'il reçoit en entrée (via `stdin`) en une URL sur un des serveurs Zope (un des *N*-1 clients ZEO). Il retourne le résultat sur `stdout`.

Exemple :

```
/about => http://www1.mondomaine.com:8080/VirtualHostBase/http/
  ➥www. mondomaine.com:80/about
/services/register=>http://www2.mondomaine.com:8080/VirtualHostBase/
  ➥http/www. mondomaine.com:80/services/register
```

On remarque que les clients sollicités sont alternativement `www1`, puis `www2`, puis `www3` et que le chemin est bien ajouté dans l'URL qui nous intéresse.

Le format de l'URL est le même que le format utilisé pour la programmation de `virtual host Zope` avec Apache, et pour cause !

Enfin, ce programme est donné à Apache, au travers de la modification suivante de `httpd.conf` :

```
<VirtualHost www.mondomaine.com>
  ServerAdmin webmaster@mondomaine.com
  ServerName www.mondomaine.com
  ErrorLog /var/log/www.error_log
  CustomLog /var/log/www.access_log
  RewriteEngine On
  RewriteMap balance prg:/usr/local/bin/balance.py
  RewriteRule ^/(.*)$ ${balance:$1} [P,L]
</VirtualHost>
```

On remarque que la règle de correspondance d'Apache est déléguée à notre petit programme `balance.py`.

Ainsi, les utilisateurs du site ne voient qu'un seul serveur web, qui est un serveur Apache, lequel sous-traite toutes ses requêtes aux *N*-1 clients ZEO.

Routeur « Soft »

On peut utiliser assez simplement des solutions à base de logiciels libres comme IPNAT, VRRPD, ou LinuxVirtualServer pour remplir cette tâche.

Les détails d'implémentation dépassent largement notre cadre.

Routeur « Hard »

Il est possible d'utiliser des commutateurs de niveau 4 pour effectuer les tâches de résilience (haute disponibilité) et de répartition de charge.

Les équipements du marché les plus souvent mis en place sont les modèles Alteon et LocalDirector de Cisco Systems.

POUR EN SAVOIR PLUS

Pour creuser le domaine des *clusters* Linux, on recommande la lecture de :

- http://www.linuxvirtualserver.org/

VRRPD pour FreeBSD est détaillé sur l'excellent site :

- http://www.bsdshell.net/hut_fvrrpd.html

En résumé…

Après l'application des différentes mesures présentées dans ce chapitre, nous disposons maintenant d'un site intranet particulièrement robuste, permettant d'offrir un service de qualité aux utilisateurs, ce qui est fondamental pour obtenir une bonne implication de leur part sur ce type de projet.

Annexes

Zope
Plone

Zope Pages Templates

A

Zope Plone

SOMMAIRE

MOTS-CLÉS

- Attribut
- Template
- Expression
- Espace de nommage
- Macro

Attribut | template | expression | espace de nommage | macro

```
<html·xmlns="http://www.w3.org/1999/xhtml"·xml:lang="en"·lang="en"
·····metal:use-macro="here/main_template/macros/master"
·····i18n:domain="plone">
<body>

<metal:main·fill-slot="main">
····<tal:main-macro·metal:define-macro="main"·tal:define="len_text·python:len(here.text);">
·········<h1·tal:content="here/title_or_id"·class="documentFirstHeading">
··········Title·or·id
·········</h1>
····
·········<div·metal:use-macro="here/document_actions/macros/document_actions">
···········Document·actions·(print,·sendto·etc)
·········</div>
····
·········<div·class="documentDescription"
···········tal:content="here/Description">
···········description
·········</div>
····
·········<div·class="stx"
···········tal:condition="len_text"
···········tal:attributes="class·python:test(here.text_format=='structured-text',·'stx',·'plain')">
···········<div·tal:replace="structure·python:here.CookedBody(stx_level=2)"·/>
·········</div>
····
·········<div·metal:use-macro="here/document_byline/macros/byline"·/>
····</tal:main-macro>
</metal:main>

</body>
</html>
```

ZPT est un mécanisme fournissant un objet `template`, et qui s'appuie sur trois langages :

- TAL (Template Attribute Language) ;
- TALES (TAL Expression Syntax) ;
- METAL (Macro Expansion Template Attribute Language).

TAL spécifie les opérations possibles, TALES le langage d'expression dans lequel les ordres sont exprimés et METAL est le système de macro de ZPT.

TAL

Syntaxe générale d'une opération TAL

```
<balise tal:commande="expression"> … </balise>
```

La grammaire de TAL est très simple : quelques « mots » seulement sont disponibles dans le vocabulaire de TAL. Chaque mot représente en fait le type de transformation portée par l'attribut `tal` : remplacement du contenu de la balise, de la balise elle-même, manipulation de ses attributs, etc. D'autres opérations, plus propres à un langage procédural sont également disponibles : boucles, conditions, définition de variables.

TAL propose donc les opérations suivantes :

- `content` – pour remplacer le contenu d'une balise ;
- `replace` – pour remplacer toute la balise ;
- `attributes` – pour manipuler les attributs de la balise ;
- `define` – pour définir une nouvelle variable ;
- `repeat`– pour faire une boucle ;
- `omit-tag` – pour ne pas produire la balise elle-même ;
- `on-error` – pour définir le comportement en cas d'erreur.

Premiers pas avec TALES

Une expression TALES permet de manipuler les objets de Zope, par exemple pour retrouver le titre de la page en cours :

```
here/title
```

Dans une expression TALES simple, la barre oblique « / » est l'équivalent du « . » dans les langages objets (dont Python).

Le préfixe `here` signifie « le sommet de la pile d'acquisition ». En d'autres termes, on demande à TALES de retrouver la propriété `title` dans le contexte courant. C'est exactement la même chose que d'écrire `context.title` dans un script Python.

Une deuxième écriture nous sera nécessaire pour bien comprendre les exemples qui vont suivre :

```
request/URL
```

Cette expression permet de retrouver la variable `URL` portée par le célèbre objet `REQUEST` de Zope.

Les opérations TAL

Opération « content »

L'opération `content` permet de remplacer le contenu de la balise par le contenu produit par l'expression passée en paramètre. Voici un exemple :

SYNTAXE

```
tal:content=
    "[structure] Expression TALES"
```

```
<h1 tal:content="here/title">Mon titre</h1>
Dans ce cas, le contenu de la balise H1, écrit par la personne qui a
fait la maquette graphique, est remplacé dynamiquement à l'exécution
par le titre de la page en cours :
<h1>Le CMS Plone</h1>
```

Contrairement au DTML, le texte produit est automatiquement converti au format HTML. Cela signifie que tous les contenus manipulés sont systématiquement transformés pour garantir un affichage fidèle dans le navigateur. Techniquement, cela signifie que le contenu est transformé par la fonction `html_quote` avant la substitution. C'est une des différences avec DTML : cette opération devait être explicite en DTML, avec l'ajout du mot-clé `html_quote` à la balise `dtml-var`.

Pour ne pas subir cette transformation, le préfixe `structure` doit être employé :

```
<div tal:content="structure here/maTable">
Ici le rendu de la table, avec ses balises
</div>
```

Dans ce cas, le code HTML de l'expression n'est pas modifié. Cette instruction est donc équivalente, en DTML, à `<dtml-var maTable>`.

Opération « replace »

L'opération `replace` permet de remplacer l'élément complet (balise et contenu) par le contenu produit par l'expression passée en paramètre. Voici un exemple :

SYNTAXE

```
tal:replace=
    "[structure] Expression TALES"
```

```
<p>Le titre de ce livre est <b tal:replace="here/title">Le titre</b>
</p>
```

En lisant trop rapidement ce code, on peut penser que la production sera :

```
<p>Le titre de ce livre est <b>Zope, 2ème edition</b></p>
```

C'est le comportement qui est obtenu par l'opération `content`, mais l'opération `replace` produit bien :

```
<p>Le titre de ce livre est Zope, 2ème edition</p>
```

On remarque que la balise porteuse de l'opération TAL a disparu. L'opération `replace` peut donc être utilisée lorsqu'un formatage prévu par le concepteur graphique pour mettre en évidence une portion de la page n'est pas nécessaire dans

le rendu final. Nous verrons plus loin d'autres cas d'utilisation de l'opération `replace`.

Opération « attributes »

SYNTAXE

```
tal:attributes=
    "ExpressionTALES1 ;
    ExpressionTALES2;
    ExpressionTALES3; …"
```

L'opération `attributes` permet de remplacer et/ou de modifier les valeurs des attributs de la balise. Voici un exemple :

```
<img src="logo.gif"
     tal:attributes="src here/getLogoImageURL; border string:1"/>
```

Cet exemple vient renseigner dynamiquement l'attribut `src` de la balise `img` pour spécifier l'emplacement de l'image à charger. On peut imaginer que `getLogoImageURL` est un script Python qui fournit l'adresse d'une image en fonction des paramètres du contexte (par exemple, la langue du navigateur de l'utilisateur).

Dans l'exemple, deux attributs sont renseignés : `src` avec l'URL de l'image et `border`, qui portera toujours la valeur `1`.

La production obtenue est, par exemple :

```
<img src="images/logos/francais.gif" border="1"/>
```

Opération « define »

SYNTAXE

```
tal:define=
    "[global] NomVariable
    ExpressionTALES1; [global]
    NomVariable ExpressionTALES2;
    [global] NomVariable
    ExpressionTALES3; …"
```

L'opération `define` permet de créer et de renseigner des variables temporaires dans la `Page Template` en cours. La portée d'une variable ainsi définie peut être :

- **Locale** : la variable est visible depuis la balise qui porte l'attribut `tal:define`, mais aussi pour toutes les balises qu'elle contient.
- **Globale** : la variable peut être visible pour tous les objets que la `Page Template` utilisée manipule. Cela signifie que non seulement la balise qui porte l'attribut `tal:define` et les balises contenues voient la variable, mais aussi les balises situées en dehors du bloc concerné et également les `Page Templates` et autres objets Zope utilisés dans le rendu de la page en cours. Bien entendu, la variable n'est visible que pour les objets évalués **après** la déclaration de la variable.

La portée par défaut est locale. Pour préciser une portée globale, il faut préfixer l'expression du mot-clé `global`. Prenons un exemple :

```
<p tal:define="utilisateur request/AUTHENTICATED_USER">
  Bienvenue <b tal:content="utilisateur">Nom de l'utilisateur</b> !
</p>
```

La variable `utilisateur` est définie dans toute la portée de la balise `p`. La variable est très simplement utilisée dans l'opération `tal:content` de la ligne qui salue l'utilisateur.

La production obtenue est :

```
<p>Bienvenue <b>user1</b></p>
```

Opération « repeat »

L'opération `repeat` permet d'effectuer des boucles ou des itérations dans le cadre d'une *template*. C'est une opération notamment très utile pour construire des tableaux, où les lignes sont la répétition du même modèle. Ainsi :

SYNTAXE

```
tal:repeat=
   "NomVariable ExpressionTALES"
```

```
<table border="1">
  <tr tal:repeat="fichier here/objectValues">
    <td tal:content="fichier/id">Nom du fichier</td>
    <td tal:content="fichier/title">Titre du fichier</td>
  </tr>
</table>
```

Cet exemple permet d'afficher la liste des fichiers présents dans le répertoire courant, en précisant leur identifiant et leur titre.

On remarque que le gabarit qui est reproduit est celui porté par la balise `tr` : il y aura dans la production autant de balises `<tr>…</tr>` que de fichiers présents dans le répertoire courant.

La production attendue est :

```
<table border="1">
  <tr>
    <td>mon_cv.doc</td>
    <td>Mon CV</td>
  </tr>
  <tr>
    <td>comptes_janvier.xls</td>
    <td>Comptes de Janvier</td>
  </tr>
<tr>
    <td>ludo.jpg</td>
    <td>Photo de Ludo</td>
  </tr>
</table>
```

Itérations imbriquées

Il est possible d'utiliser des opérations imbriquées. Par exemple, le code suivant permet d'afficher une table de multiplication :

```
<table border="1" tal:define="liste python:[1,2,3,4,5,6,7,8,9,10]">
  <tr tal:repeat="x liste">
    <td tal:repeat="y liste"tal:content="python:x*y">X * Y</td>
  </tr>
</table>
```

L'opération `define` portée par la table permet de construire une variable qui est une liste Python contenant les valeurs entières de 1 à 10.

Le `tr` porte une première itération, avec `x` qui prend pour valeurs successives les valeurs de la liste.

Enfin, le `td` est répété à nouveau pour tous les éléments de la liste, mais sur la variable d'itération `y`. On note que le résultat est obtenu par une opération `content`, qui utilise la formule `x*y` exprimée en Python.

Informations portées par les itérations

Pour accéder à des informations relatives à l'itération elle-même, une variable particulière, `repeat`, est accessible au sein des expressions TALES.

La variable `repeat` contient une entrée par boucle en cours (c'est-à-dire qu'il est possible d'accéder à des boucles imbriquées).

Voici un exemple d'utilisation :

```
<table border="1">
  <tr tal:repeat="item here/objectValues">
    <th tal:content="repeat/item/number">No de fichier</th>
    <td tal:content="item/title_or_id">Titre du fichier</td>
  </tr>
</table>
```

Cet exemple va créer un magnifique tableau avec la liste des objets contenus dans le dossier courant. On retrouve le numéro d'ordre de l'objet dans la première colonne (avec `repeat/item/number`), suivi du titre de l'objet dans la seconde.

Les variables d'itérations portent les informations suivantes :

- `index` – l'index de la répétition dans la séquence en cours (commence à partir de zéro) ;
- `number` – comme `index`, mais commence à partir de 1 ;
- `even` – retourne la valeur « vrai » si l'index en cours est pair (0, 2, 4, etc.) ;
- `odd` – retourne la valeur « vrai » si l'index en cours est impair (1, 3, 5, etc.) ;
- `start` – retourne la valeur « vrai » pour la première itération ;
- `end` – retourne la valeur « vrai » pour la dernière itération ;
- `first` – retourne la valeur « vrai » pour la première itération d'un groupe ;
- `last` – retourne la valeur « vrai » pour la dernière itération d'un groupe ;
- `length` – nombre total d'itérations ;
- `letter` – comme `index`, mais exprimé avec des lettres : `a-z`, `aa-az`, `ba-bz`, ..., `za-zz`, `aaa-aaz`, etc. ;
- `Letter` – comme `letter`, en majuscules ;
- `roman` – comme `item`, en lettres romaines minuscules : `i`, `ii`, `iii`, `iv`, `v`, etc. ;
- `Roman` – comme `roman`, en majuscules.

Opération « on-error »

L'opération `on-error` est appelée si la balise en cours lève une exception dans le traitement d'une opération TAL. Dans ce cas, l'exception est étouffée et la production utilisée est celle de l'expression TALES portée par la commande `on-error`.

L'objet exception est ajouté au *namespace* dans la variable `error`. Voici un exemple :

```
<p>Le prix de l'article est <b tal:content="here/prix"

   tal:on-error="string:N/A">0.00</b> Euros.</p>
```

Cet exemple affiche le prix d'un produit. Admettons que le prix soit une propriété prix, portée par les objets concernés. Dans le cas où l'objet n'a pas cette propriété (parce que l'administrateur a oublié de la saisir par exemple), la chaîne `N/A` est affichée à la place du prix.

SYNTAXE

tal:on-error="ExpressionTALES"

Opération « omit-tag »

L'opération `omit-tag` est utilisée pour ne pas produire la balise qui la porte, mais seulement son contenu. Ceci permet d'utiliser des balises pour porter de la logique, sans que celles-ci ne soient produites par le moteur de rendu.

```
<p>On compte de 1 à 3 :
  <span tal:repeat="cpt python:range(1,3)" tal:content="cpt">1</span>
</p>
```

Ce code produit :

```
<p>On compte de 1 à 3 :
  <span>1</span>
  <span>2</span>
  <span>3</span>
</p>
```

Si on utilise `tag-omit` :

```
<p>On compte de 1 à 3 :
  <span tal:repeat="cpt python:range(1,3)" tal:content="cpt" tal:tag-
omit="">1</span>
</p>
```

Le résultat est le suivant :

```
<p>On compte de 1 à 3 :
  1
  2
  3
</p>
```

La balise porteuse `span` n'a pas été produite.

SYNTAXE

tal:omit-tag=""
Associer `tal:content` à `tal:omit-tag` équivaut à utiliser une opération `tal:replace`.

Opérations multiples

Toutes les commandes peuvent être utilisées en combinaison dans le cadre de la même balise, comme dans l'exemple suivant :

```
<p tal:content="here/myContent"
   tal:attributes="align string:'center'">Le contenu ici</p>
```

Dans cet exemple, la méthode Python `myContent` est appelée pour fournir le contenu du paragraphe, et l'attribut `align` est positionné à `center`.

La seule combinaison interdite est le mélange des opérations `content` et `replace`, qui sont mutuellement exclusives.

Cependant, une même commande ne peut apparaître qu'une seule fois dans le corps d'une balise.

Si plusieurs commandes TAL sont disposées sur une même balise, l'ordre de priorité dans lequel elle sont évaluées est le suivant :

- `define` ;
- `condition` ;
- `repeat` ;
- `content` ou `replace` ;
- `attributes` ;
- `omit-tag`.

TALES

TALES est le langage dans lequel sont écrites les expressions TAL portées par les attributs, par exemple :

```
<i tal:content=" ExpressionTal">bla bla</i>
```

Une expression TALES est généralement formatée de la façon suivante :

```
préfixe:expression
```

où `préfixe` permet de préciser le type de l'expression.

L'expression est interprétée grâce au type précisé. Le préfixe peut être omis. Dans ce cas, c'est le type par défaut (`path`) qui est utilisé.

TALES propose principalement les types d'expression suivants :

- `path`,
- `not`,
- `string`,
- `nocall`,
- `exists`,
- `python`.

Expressions de type « path »

Elles permettent d'insérer ou de remplacer la valeur d'un objet ou attribut disponible dans la pile d'acquisition de la *template*. Ce sont les expressions simples qui ont été le plus utilisées jusqu'ici (dans les exemples) et où l'on se sert de la barre oblique « / » à la place de la notation « . » usuelle des langages tels que Python. Le préfixe utilisé est `path`, mais il est généralement omis.

TALES fournit plusieurs variables implicites telles que `here` et `request`, qui facilitent l'écriture des expressions. Les variables implicites sont présentées en détail plus loin.

Voici quelques exemples simples avec ce type d'expression :

```
<div tal:define="fichiers container/dossier/objectValues" >
  <div tal:repeat="fichier fichiers"
       tal :content="fichier/title_or_id" >
    Fichier
  </div>
</div>
<span tal:content="user/getUserName" > jdupont </span>
Date de modification :
<span tal:replace="here/bobobase_modification_time" >
2003/07/08 14:35:10 </span>
```

On peut également combiner plusieurs expressions simples alternatives comme dans l'exemple suivant :

```
<span tal:content="request/form/custname | here/custname | nothing" >
 Nom du client </span>
```

Expressions de type « not »

Les expressions de ce type permettent d'effectuer une négation sur une expression de type `path`. Son utilisation est très simple : il faut la voir comme une extension triviale du type `path`. En effet, en ajoutant simplement le mot-clé `not:` devant une expression `path`, la valeur booléenne de celle-ci est inversée.

Cela s'avère fort pratique pour simuler une condition de type `else`, comme dans l'exemple ci-dessous, qui permet d'afficher le résultat d'un formulaire dans lequel une variable peut être cochée ou pas :

```
<div tal:condition="request/maVariable">
    Vous avez coché la case
</div>
<div tal:condition="not:request/maVariable">
    Vous n'avez pas coché la case
</div>
```

Expressions de type « string »

Elles permettent de combiner une expression simple avec du texte, ou plusieurs petites expressions simples, dans la même opération. La combinaison se fait par concaténation des chaînes de caractères, d'où le nom de ce type d'expressions. Le préfixe utilisé est `string`.

Voici quelques exemples simples avec ce type d'expression :

```
<span tal:content="string:Bienvenue ${user/getUserName} !" >
Bienvenue jdupont !</span>
<span tal:content="string:Date de modification : ${here/
                ➥ bobobase_modification_time}" >
2003/07/08 14:35:10 </span>
```

Expressions de type « nocall »

Une expression de type `path` *rend* l'objet qu'elle évalue. Dans certains cas rares, le résultat de l'expression est un objet Zope exécutable et vous ne voulez pas qu'il soit rendu. Le préfixe utilisé est `nocall`. Le cas typique est celui d'un objet de type `DTML Document` auquel on veut accéder simplement pour récupérer et afficher ses propriétés.

Voici un exemple :

```
<div tal:define="doc nocall:here/mondoc"
     tal:content="string:${mondoc/getId}: ${doc/title}">
  <a href="" tal:attributes="href doc/absolute_url"
     tal:content="here/title">Titre du doc </a>
</div>
```

Expressions de type « exists »

Une expression de type `exists` est similaire à une expression de type `path`, sauf qu'au lieu de retourner une valeur (la valeur de l'objet ou attribut auquel on accède), elle teste simplement son existence.

Ce type d'expression est peu utilisé, mais est prévu par le langage pour apporter de la souplesse dans des cas où l'on doit faire plusieurs traitements sur la même balise, dont le test du résultat de l'un de ces traitements.

En voici un exemple :

```
<p tal:condition="exists:request/form/notitymessage"
   tal:content="request/form/notitymessage"> Notification </p>
```

Expressions de type « python »

Elles permettent d'évaluer du code Python (l'équivalent des « expressions Python » dans le contexte du DTML) pour une opération TAL. Le préfixe utilisé est `python`.

Voici quelques exemples simples avec ce type d'expression :

```
<span tal:content="python:len(container.dossier.objectValues(
                ➥ ['File', 'Image']))" >12</span> fichiers
Date de modification :
<span tal:replace="python:here.bobobase_modification_time.
                ➥ strftime('%d/%m/%Y')">08/07/2003</span>
```

Ce type d'expression est celui qui apporte le plus de flexibilité pour traiter des opérations complexes, puisque l'on a accès à la puissance de Python. De plus, il est possible d'utiliser un autre type d'expression tel que `path` ou `string` au sein d'une expression `python`. Ceci est possible parce qu'une fonction Python est disponible pour chaque type d'expression que l'on voudrait utiliser au sein d'une expression `python`. Ce sont les fonctions qui portent le même nom : `path()`, `string()`, `exists()` et `nocall()`.

On peut donc utiliser des expressions de type `python` comme dans les exemples suivants :

```
<span tal:replace="python:path('here/%s/title_or_id' %
                ➥ dossiercourant)" />
<span tal:replace="python:path('request/form/membername |
                ➥ here/membername | nothing') or 'Invité'" />
```

Espace de nommage

L'espace de nommage des `Pages Templates` contient quelques variables implicites dont la liste est présentée ci-après.

`user`	Cette variable représente l'objet correspondant à l'utilisateur connecté. Elle fournit ainsi l'accès au nom de l'utilisateur et à ses rôles et permissions. Dans le monde DTML, cette variable s'appelle `AUTHENTICATED_USER`. Ex. : `Bienvenue <b tal:content="user/getUserName">Nom utilisateur</b>`
`here`	Cette variable correspond au contexte d'exécution de la page en cours. C'est sur cette variable que l'on retrouve les variables de la pile d'acquisition par exemple. Cette variable est très proche de la variable `context`, que nous rencontrerons dans les *Python Scripts*. Ex. : `Le titre de cette page est <span tal:replace="here/title">titre du répertoire en cours</span>`.
`request`	La variable `REQUEST` de Zope est ainsi accessible. Elle porte les informations relatives à la connexion HTTP en cours.
`template`	Cette variable représente l'objet `page template` lui-même. Elle peut être utile dans des cas rares, comme pour afficher le nom de la `template` dans le titre.
`nothing`	Cette valeur spéciale permet d'effacer une balise ou son contenu selon qu'elle est utilisée dans un `tal:replace` ou un `tal:content`, ou un `tal:attributes`. Ex : `<p tal:replace="nothing">` Commentaire : cette information est susceptible de changer.</p>

default	Cette valeur spéciale n'affecte aucune modification à la balise lorsqu'elle est utilisée dans un tal:replace, tal:content, ou tal:attributes. Elle conserve intact le texte auquel s'applique la balise. Ex. : Statut : N/A.
options	Si des arguments de type keyword sont envoyés à la template après l'exécution d'un script, ceux-ci sont disponibles dans la variable options. Ex. : Erreur : Le message d'erreur
attrs	Cette variable est le dictionnaire des attributs de la balise courante dans la template. Les clés du dictionnaire sont les noms des attributs et ses valeurs correspondent aux valeurs d'origine des attributs (avant le rendu de la template). Elle est rarement utilisée.
root	Cette variable représente l'objet racine de Zope. Elle peut être utile pour accéder à des objets donnés de l'arborescence de Zope.
container	Cette variable représente le conteneur (généralement un répertoire) de l'objet template. Ex. : Cette template est hébergée dans le dossier : titre du conteneur de la template.
modules	Cette variable contient la collection des modules Python que l'on peut utiliser au sein d'une template Ex. :

METAL

METAL (*Macro Expansion Template Attribute Language*) est le système qui permet de réutiliser des blocs entiers de code d'une template donnée au sein d'une autre template. C'est le mécanisme des macros. En proposant au développeur une « approche objet » dans la manière de mettre en œuvre les *skins* (couches de présentation du site), les macros apportent un gain en qualité et en maintenabilité pour des sites ou des systèmes de gestion de contenu complexes.

Principe d'une macro

Une macro regroupe un certain nombre d'éléments de code HTML d'une template afin de les réutiliser dans d'autres templates. Elle s'applique à la balise qui délimite le bloc de code concerné, et elle porte un nom.

La macro contient généralement un (sous-)bloc vide (ou plusieurs) appelé *slot* (littéralement : tiroir), que l'on remplit avec du code spécifique dans chaque template où l'on utilise la macro.

Les fonctions du langage METAL

La grammaire de METAL est similaire à celle de TAL, mais elle ne fournit que les quatre commandes suivantes :

- `define-macro`, qui permet de définir une macro ;
- `define-slot`, qui permet de définir un bloc de code personnalisable au sein d'une macro ;
- `use-macro`, qui permet d'utiliser une macro déjà définie ;
- `fill-slot`, qui permet de personnaliser une *slot* au sein de la macro utilisée.

Syntaxe générale d'une fonction METAL

```
<balise-de-bloc metal:commande=
    "chemin ou nom">
…
</balise-de-bloc>
```

L'API de Plone

B

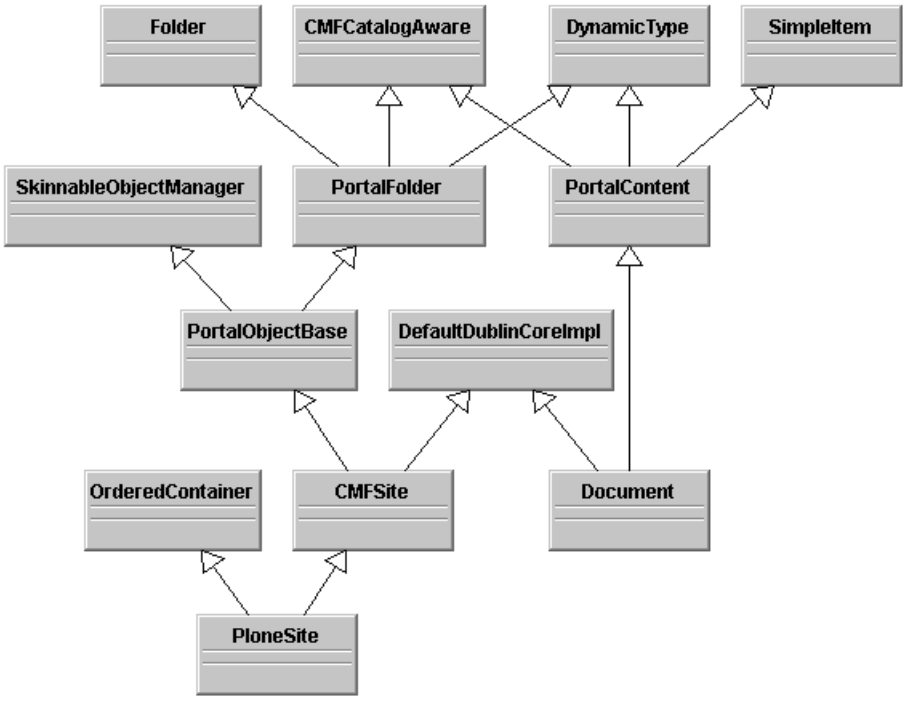
Zope Plone

SOMMAIRE

MOTS-CLÉS

- Classe
- Héritage
- Méthode
- Fonction
- Variable

Classe | héritage | méthode | fonction | variable



Nous présentons ici les éléments les plus utilisés de l'API de Plone : les principales classes de contenu, les classes de composants (ou Tools), et les variables *builtins* qui permettent d'avoir accès aux services du site Plone.

Objets de contenu

Module Products.CMFPlone.PloneContent

Classe PloneContent

Classes de base

`Products.CMFCore.PortalContent.PortalContent`

Méthodes et fonctions

`failIfLocked(self)`

`SearchableText(self)`

`view(self)`

Module Products.CMFCore.PortalContent

Classe PortalContent

Classes de base

`Products.CMFCore.DynamicType.DynamicType`

`Products.CMFCore.CMFCatalogAware.CMFCatalogAware`

`OFS.SimpleItem.SimpleItem`

Méthodes et fonctions

`failIfLocked(self)`

`SearchableText(self)`

`view(self)`

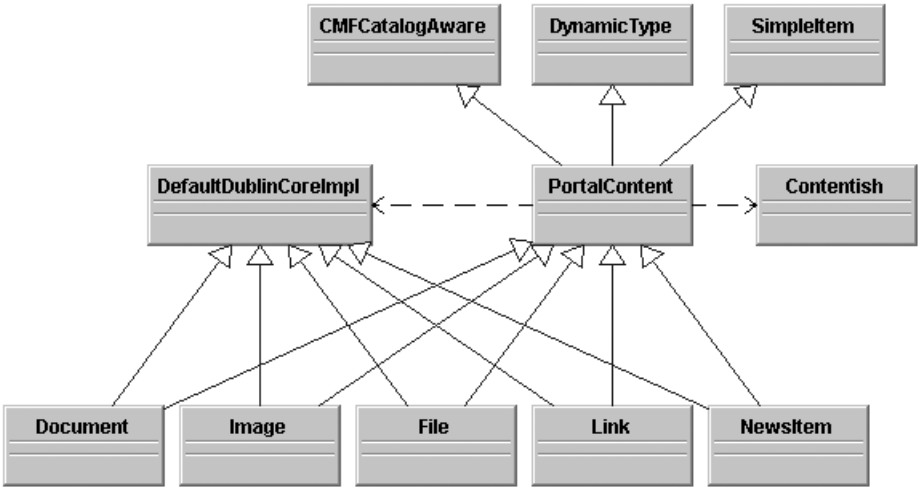


Figure B–1
Les classes de contenu CMF

Module Products.CMFPlone.PloneFolder

Classe PloneFolder

Classes de base

Products.CMFPlone.PloneFolder.BasePloneFolder

Products.CMFPlone.PloneFolder.OrderedContainer

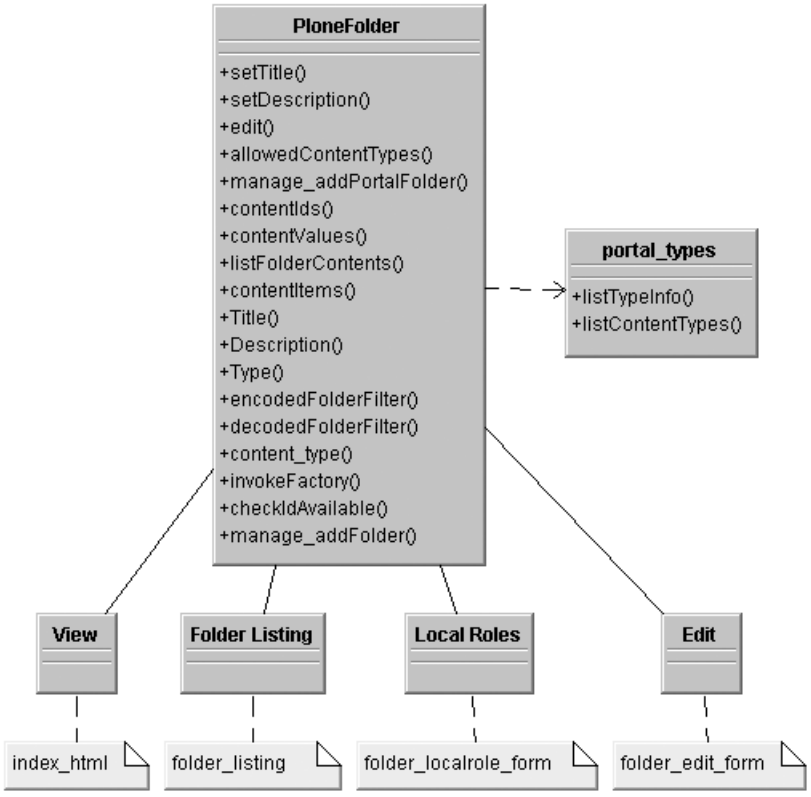


Figure B–2
La classe PloneFolder

Classe BasePloneFolder

Classes de base

Products.CMFDefault.SkinnedFolder.SkinnedFolder

Products.CMFDefault.DublinCore.DefaultDublinCoreImpl

Méthodes et fonctions

```
contentValues(self, spec=None, filter=None, sort_on=None, reverse=0)
folderlistingFolderContents(self, spec=None, contentFilter=None,
   suppressHiddenFiles=0)
invokeFactory(self, type_name, id, RESPONSE=None, *args, **kw)
```

*args est un tuple d'arguments non nommés.
**kw est un dictionnaire d'arguments nommés.

```
listFolderContents(self, spec=None, contentFilter=None,
   suppressHiddenFiles=0)
manage_addPloneFolder(self, id, title='', REQUEST=None)
manage_delObjects(self, ids=[], REQUEST=None)
manage_renameObject(self, id, new_id, REQUEST=None)
```

Classe OrderedContainer

Classes de base

```
OFS.Folder.Folder
```

Méthodes et fonctions

```
getObjectPosition(self, id)
manage_renameObject(self, id, new_id, REQUEST=None)
moveObject(self, id, position)
moveObjectToPosition(self, id, position)
moveObjectsByDelta(self, ids, delta)
moveObjectsDown(self, ids, delta=1, RESPONSE=None)
moveObjectsToBottom(self, ids, RESPONSE=None)
moveObjectsToTop(self, ids, RESPONSE=None)
moveObjectsUp(self, ids, delta=1, RESPONSE=None)
orderObjects(self, key, reverse=None)
```

Composants de service

Module Products.CMFPlone.CatalogTool

Classe CatalogTool

Classes de base

```
Products.CMFCore.CatalogTool.CatalogTool
```

Méthodes et fonctions

```
enumerateIndexes(self)
indexObject(self, object, idxs=[])
migrateIndexes(self)
```

Module Products.CMFCore.CatalogTool

Classe CatalogTool

Classes de base

Products.CMFCore.utils.UniqueObject

Products.ZCatalog.ZCatalog.ZCatalog

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

searchResults(self, REQUEST=None, **kw)

catalog_object(self, object, uid, idxs=[])

enumerateColumns(self)

enumerateIndexes(self)

indexObject(self, object)

reindexObject(self, object, idxs=[])

unindexObject(self, object)

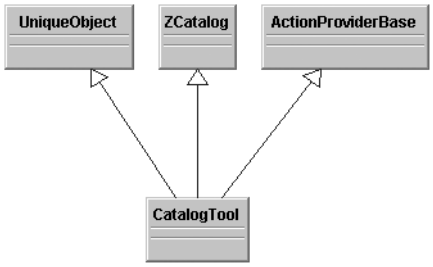


Figure B–3 La classe CatalogTool

Classe IndexableObjectWrapper

Méthodes et fonctions

allowedRolesAndUsers(self)

Module Products.CMFPlone.ActionsTool

Classe ActionsTool

Classes de base

Products.CMFCore.ActionsTool.ActionsTool

Méthodes et fonctions

listFilteredActionsFor(self, object=None)

Module Products.CMFCore.ActionsTool

Classe ActionsTool

Classes de base

Products.CMFCore.utils.UniqueObject

OFS.Folder.Folder

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

```
addActionProvider(self, provider_name)
deleteActionProvider(self, provider_name)
listActionProviders(self)
listFilteredActionsFor(self, object=None)
manage_aproviders(self, apname='', chosen=(), add_provider=0,
   del_provider=0, REQUEST=None)
```

Module Products.CMFPlone.DiscussionTool

Classe DiscussionTool

Classes de base

```
Products.CMFDefault.DiscussionTool.DiscussionTool
```

Méthodes et fonctions

```
cookReply(self, reply, text_format=None)
```

Module Products.CMFCore.DiscussionTool

Classe DiscussionTool

Classes de base

```
Products.CMFCore.utils.UniqueObject
OFS.SimpleItem.SimpleItem
```

Méthodes et fonctions

```
getDiscussionFor(self, content)
isDiscussionAllowedFor(self, content)
listActions(self, info)
```

Module Products.CMFPlone.FactoryTool

Classe FactoryTool

Classes de base

```
Products.CMFCore.utils.UniqueObject
OFS.SimpleItem.SimpleItem
```

Méthodes et fonctions

```
doCreate(self, obj, id=None, **kw)
```

fixRequest(self)
getFactoryTypes(self)
getTempFolder(self, type_name)
isTemporary(self, obj)
manage_setPortalFactoryTypes(self, REQUEST=None, listOfTypeIds=None)

Classe TempFolder

Classes de base

Products.CMFPlone.PloneFolder.PloneFolder

Méthodes et fonctions

getOwner(self, info=0, aq_get=, UnownableOwner=[], getSecurityManager=)
getPhysicalPath(self)
get_local_roles_for_userid(self, userid)
get_valid_userids(self)
has_local_roles(self)
manage_renameObject(self, id1, id2)
owner_info(self)
userCanTakeOwnership(self)
userdefined_roles(self)
valid_roles(self)
validate_roles(self, roles)

Module Products.CMFPlone.GroupDataTool

Classe GroupDataTool

Classes de base

Products.GroupUserFolder.GroupDataTool.GroupDataTool

Module Products.CMFPlone.GroupsTool

Classe GroupsTool

Classes de base

Products.GroupUserFolder.GroupsTool.GroupsTool

Module Products.CMFPlone.MemberDataTool

Classe MemberData

Classes de base

Products.CMFCore.MemberDataTool.MemberData

Classe MemberDataTool

Classes de base

Products.CMFCore.MemberDataTool.MemberDataTool

Méthodes et fonctions

pruneMemberDataContents(self)

Module Products.CMFCore.MemberDataTool

Classe MemberData

Classes de base

OFS.SimpleItem.SimpleItem

Méthodes et fonctions

getDomains(self)

getId(self)

getMemberId(self)

getPassword(self)

getProperty(self, id, default=[])

getRoles(self)

getRolesInContext(self, object)

getTool(self)

getUser(self)

getUserName(self)

has_role(self, roles, object=None)

notifyModified(self)

setMemberProperties(self, mapping)

setProperties(self, properties=None, **kw)

setSecurityProfile(self, password=None, roles=None, domains=None)

Classe MemberDataTool

Classes de base

Products.CMFCore.utils.UniqueObject
OFS.SimpleItem.SimpleItem
OFS.PropertyManager.PropertyManager
Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

getMemberDataContents(self)
pruneMemberDataContents(self)
registerMemberData(self, m, id)
searchMemberDataContents(self, search_param, search_term)
wrapUser(self, u)

Module Products.CMFPlone.MembershipTool

Classe MembershipTool

Classes de base

Products.CMFDefault.MembershipTool.MembershipTool

Méthodes et fonctions

changeMemberPortrait(self, portrait, member_id=None)
createMemberarea(self, member_id=None)
createMemberarea(self, member_id=None)
deletePersonalPortrait(self, member_id=None)
getAuthenticatedMember(self)
getPersonalFolder(self, member_id=None)
getPersonalPortrait(self, member_id=None, verifyPermission=0)
listMemberIds(self)
listMembers(self)
searchForMembers(self, REQUEST=None, **kw)
setPassword(self, password, domains=None)
testCurrentPassword(self, password, username=None)

Module Products.CMFCore.MembershipTool

Classe MembershipTool

Classes de base

Products.CMFCore.utils.UniqueObject

OFS.SimpleItem.SimpleItem

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

addMember(self, id, password, roles, domains, properties=None)

checkPermission(self, permissionName, object, subobjectName=None)

createMemberarea(self, member_id) credentialsChanged(self, password)

deleteLocalRoles(self, obj, member_ids, reindex=1)

getAuthenticatedMember(self)

getCandidateLocalRoles(self, obj)

getHomeFolder(self, id=None, verifyPermission=0)

getHomeUrl(self, id=None, verifyPermission=0)

getMappedRole(self, portal_role)

getMemberById(self, id)

getMemberareaCreationFlag(self)

getPortalRoles(self)

isAnonymousUser(self)

listMemberIds(self)

listMembers(self)

searchMembers(self, search_param, search_term)

setLocalRoles(self, obj, member_ids, member_role, reindex=1)

setMemberareaCreationFlag(self)

setPassword(self, password, domains=None)

setRoleMapping(self, portal_role, userfolder_role)

wrapUser(self, u, wrap_anon=0)

Module Products.CMFPlone.MetadataTool

Classe MetadataTool

Classes de base

Products.CMFDefault.MetadataTool.MetadataTool

Module Products.CMFDefault.MetadataTool

Classe MetadataTool

Classes de base

Products.CMFCore.utils.UniqueObject

OFS.SimpleItem.SimpleItem

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

```
editProperties(self, publisher=None, REQUEST=None)
addElementPolicy(self, element, content_type, is_required,
   supply_default, default_value, enforce_vocabulary,
   allowed_vocabulary, REQUEST=None)
removeElementPolicy(self, element, content_type, REQUEST=None)
updateElementPolicy(self, element, content_type, is_required,
   supply_default, default_value, enforce_vocabulary,
   allowed_vocabulary, REQUEST=None)
listElementSpecs(self)
getElementSpec(self, element)
addElementSpec(self, element, is_multi_valued, REQUEST=None)
removeElementSpec(self, element, REQUEST=None)
listPolicies(self, typ=None)
getFullName(self, userid)
getPublisher(self)
listAllowedVocabulary(self, element, content=None, content_type=None)
listAllowedSubjects(self, content=None, content_type=None)
listAllowedFormats(self, content=None, content_type=None)
listAllowedLanguages(self, content=None, content_type=None)
listAllowedRights(self, content=None, content_type=None)
setInitialMetadata(self, content)
validateMetadata(self, content)
```

Module Products.CMFPlone.RegistrationTool

Classe RegistrationTool

Classes de base

Products.CMFDefault.RegistrationTool.RegistrationTool

Méthodes et fonctions

```
generatePassword(self)
generateResetCode(self, salt, length=14)
getPassword(self, length=5, s=None)
isValidEmail(self, email)
```

Module Products.CMFCore.RegistrationTool

Classe RegistrationTool

Classes de base

```
Products.CMFCore.utils.UniqueObject
OFS.SimpleItem.SimpleItem
Products.CMFCore.ActionProviderBase.ActionProviderBase
```

Méthodes et fonctions

```
addMember(self, id, password, roles=('Member',), domains='',
   properties=None)
afterAdd(self, member, id, password, properties)
generatePassword(self)
isMemberIdAllowed(self, id)
isRegistrationAllowed(self, REQUEST)
mailPassword(self, forgotten_userid, REQUEST)
testPasswordValidity(self, password, confirm=None)
testPropertiesValidity(self, new_properties, member=None)
```

Module Products.CMFPlone.SkinsTool

Classe SkinsTool

Classes de base

```
Products.CMFCore.SkinsTool.SkinsTool
```

Module Products.CMFCore.SkinsTool

Classe SkinsTool

Classes de base

```
Products.CMFCore.utils.UniqueObject
Products.CMFCore.SkinsContainer.SkinsContainer
```

OFS.Folder.Folder

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

PUT_factory(self, name, typ, body)

addSkinSelection(self, skinname, skinpath, test=0, make_default=0)

clearSkinCache(self)

clearSkinCookie(self)

getAllowAny(self)

getCookiePersistence(self)

getDefaultSkin(self)

getRequestVarname(self)

getSkinByName(self, name)

getSkinPath(self, name)

getSkinPaths(self)

getSkinSelections(self)

manage_afterAdd(self, item, container)

manage_afterClone(self, item)

manage_beforeDelete(self, item, container)

manage_properties(self, default_skin='', request_varname='',
   allow_any=0, chosen=(), add_skin=0, del_skin=0, skinname='',
   skinpath='', cookie_persistence=0, REQUEST=None)

manage_reloadSkins(self, REQUEST=None)

manage_skinLayers(self, chosen=(), add_skin=0, del_skin=0, skinname='',
   skinpath='', REQUEST=None)

reloadDVS(self, dvs, parent='')

testSkinPath(self, p)

updateSkinCookie(self)

Module Products.CMFPlone.TypesTool

Classe TypesTool

Classes de base

Products.CMFCore.TypesTool.TypesTool

Module Products.CMFCore.TypesTool

Classe FactoryTypeInformation

Classes de base

Products.CMFCore.TypesTool.TypeInformation

Méthodes et fonctions

constructInstance(self, container, id, *args, **kw)

isConstructionAllowed(self, container)

Classe ScriptableTypeInformation

Classes de base

Products.CMFCore.TypesTool.TypeInformation

Méthodes et fonctions

constructInstance(self, container, id, *args, **kw)

isConstructionAllowed(self, container)

Classe TypeInformation

Classes de base

Products.CMFCore.utils.SimpleItemWithProperties

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

Description(self)

Metatype(self)

Title(self)

Type(self)

allowDiscussion(self)

allowType(self, contentType)

getActionById(self, id, default=[])

getIcon(self)

getId(self)

globalAllow(self)

listActions(self, info=None)

Classe TypesTool

Classes de base

Products.CMFCore.utils.UniqueObject

OFS.Folder.Folder

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

constructContent(self, type_name, container, id, RESPONSE=None, *args, **kw)

getTypeInfo(self, contentType)

listActions(self, info=None)

listContentTypes(self, container=None, by_metatype=0)

listDefaultTypeInformation(self)

listTypeInfo(self, container=None)

manage_addFactoryTIForm(self, REQUEST)

manage_addScriptableTIForm(self, REQUEST)

manage_addTypeInformation(self, add_meta_type, id=None, typeinfo_name=None, RESPONSE=None)

Products.CMFPlone.URLTool

Classe URLTool

Classes de base

Products.CMFCore.URLTool.URLTool

Module Products.CMFCore.URLTool

Classe URLTool

Classes de base

Products.CMFCore.utils.UniqueObject

OFS.SimpleItem.SimpleItem

Products.CMFCore.ActionProviderBase.ActionProviderBase

Méthodes et fonctions

getPortalObject(self)

getPortalPath(self)

getRelativeContentPath(self, content)

getRelativeContentURL(self, content)

Products.CMFPlone.WorkflowTool

Classe WorkflowTool

Classes de base

Products.CMFCore.WorkflowTool.WorkflowTool

Méthodes et fonctions

doActionFor(self, ob, action, wf_id=None, *args, **kw)
flattenTransitions(self, objs, container=None)
getChainForPortalType(self, pt_name, managescreen=0)
getTransitionsFor(self, obj=None, container=None, REQUEST=None)
getWorklists(self)
listWorkflows(self)
workflows_in_use(self)

Products.CMFCore.WorkflowTool

Classe WorkflowTool

Classes de base

Products.CMFCore.utils.UniqueObject
OFS.Folder.Folder

Méthodes et fonctions

doActionFor(self, ob, action, wf_id=None, *args, **kw)
getActionsFor(self, ob)
getCatalogVariablesFor(self, ob)
getChainFor(self, ob)
getDefaultChainFor(self, ob)
getHistoryOf(self, wf_id, ob)
getInfoFor(self, ob, name, default=[], wf_id=None, *args, **kw)
getStatusOf(self, wf_id, ob)
getWorkflowById(self, wf_id)
getWorkflowIds(self)
getWorkflowsFor(self, ob)
listActions(self, info)
manage_addWorkflow(self, workflow_type, id, RESPONSE=None)
manage_addWorkflowForm(self, REQUEST)
manage_changeWorkflows(self, default_chain, props=None, REQUEST=None)

```
manage_selectWorkflows(self, REQUEST, manage_tabs_message=None)
notifyBefore(self, ob, action)
notifyCreated(self, ob)
notifyException(self, ob, action, exc)
notifySuccess(self, ob, action, result=None)
setChainForPortalTypes(self, pt_names, chain)
setDefaultChain(self, default_chain)
setStatusOf(self, wf_id, ob, status)
updateRoleMappings(self, REQUEST=None)
wrapWorkflowMethod(self, ob, method_id, func, args, kw)
```

Les builtins de l'interface utilisateur

Variables globales des templates

Pour faciliter le travail des développeurs, de nombreux objets ou services sont directement accessibles via des variables globales qui sont définies au sein de `templateglobal_defines` appelée au début du `template` principal de Plone (`main_template`).

Ces variables sont disponibles dans tous les `templates` de l'interface utilisateur.

Variables d'accès aux composants de service (Tools)

Variable	Définition
utool	l'objet URL Tool
portal_properties	l'objet Properties Tool
mtool	l'objet Membership Tool
gtool	l'objet Groups Tool
gdtool	l'objet Groupdata Tool
atool	l'objet Actions Tool
aitool	l'objet ActionIcons Tool
putils	l'objet Plone (Utils) Tool
wtool	l'objet Workflow Tool

Autres variables de templates

Variable	Définition
portal	le site Plone
portal_url	l'adresse URL absolue du site

Variable	Définition
member	le membre authentifié
checkPermission	la méthode permettant de vérifier que le membre authentifié possède une permission donnée
isAnon	s'il existe, l'utilisateur n'est pas encore authentifié (anonyme)
actions	la liste de toutes les actions contextuelles
user_actions	la liste des actions personnelles
workflow_actions	la liste des actions exposées par le workflow
folder_actions	la liste des actions portées par le dossier courant
portal_tabs	la liste des actions exposées sous forme d'onglets de type portal_tab
site_properties	la fiche de propriétés stockant les paramètres généraux
isFolder	s'il existe, l'objet courant est un dossier (Folder)
here_url	l'adresse URL absolue de l'objet courant
default_language	la langue par défaut du site

Variables pour les expressions de paramétrage d'une action

Le composant ActionsTool ajoute les variables suivantes pour les expressions TALES utilisées dans les champs Action et Condition d'une action :

Variable	Définition
portal	le site Plone
portal_url	l'URL absolue du site
folder	le dossier courant (dans lequel on se trouve)
folder_url	l'URL absolue du dossier courant
object	l'objet courant (sur lequel on se trouve)
object_url	l'URL absolue de l'objet courant
member	le membre authentifié (s'il y a eu authentification d'un utilisateur)
request	la requête HTTP envoyée au serveur
modules	le conteneur des modules Python autorisés dans une Page Template (ex. string, ZTUtils, etc.)

Aide-mémoire Archetypes

C

Zope Plone

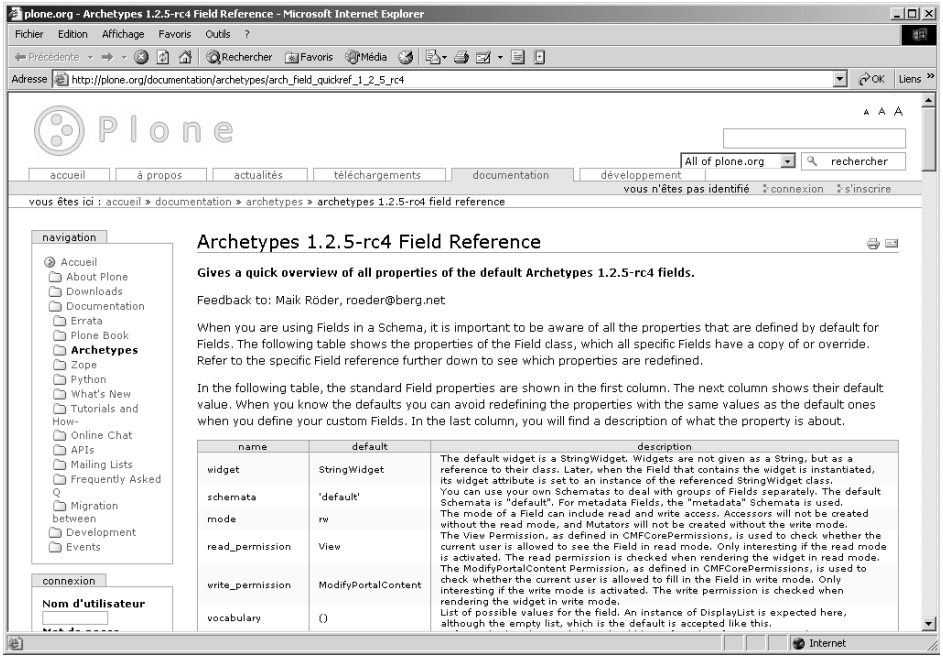
Widget | Field | propriétés | schéma

SOMMAIRE

- Aide-mémorie pour les champs (field)
- Aide-mémoire pour les widgets

MOTS-CLÉS

- Widget
- Field
- Propriétés
- Schéma



Cette annexe décrit les champs et widgets Archetypes, déjà rencontrés au chapitre 7. Elle tiendra lieu de référence au développeur souhaitant exploiter les possibilités de cette technologie.

Aide-mémoire pour les champs (field)

- http://plone.org/documentation/archetypes/arch_field_quickref_1_2_5_rc4

Maik Röder, roeder@berg.net

Pour utiliser des champs dans la définition d'un schéma, il est important de connaître les propriétés définies par défaut pour un champ. Le tableau ci-dessous contient toutes les propriétés de la classe `Field`. Lorsqu'on connaît les valeurs par défaut, on peut définir des schémas plus compacts. Une propriété n'a besoin d'être spécifiée que si sa valeur est différente de la valeur par défaut.

Nom de la propriété	Valeur par défaut	Notes
widget	StringWidget	Le widget par défaut est de type `StringWidget`. L'attribut `widget` n'est pas spécifié par une chaîne, mais par une référence vers sa classe. Au moment de l'initialisation d'un champ, le widget est instancié.
schemata	'default'	Un `schemata` permet de travailler avec plusieurs groupes de champs. Le schéma par défaut est `default`. Pour les métadonnées, le `schemata` est `metadata`.
mode	rw	Par défaut, les modes écriture (w) et lecture (r) sont activés pour les champs. En l'absence du mode lecture, aucun `accessor` n'est créé. Aucun `mutator` n'est créé en l'absence du mode écriture.
read_permission	View	La permission `View` est définie dans `CMFCorePermissions`. Elle sert à vérifier si un utilisateur peut voir un champ en mode lecture. Ce n'est intéressant que si le mode lecture est actif. La permission est vérifiée au moment où le widget est restitué.
write_permission	ModifyPortalContent	La permission `ModifyPortalContent` est aussi définie dans `CMFCorePermissions` et sert à vérifier le droit d'accès à un champ lors de la restitution en mode écriture.
vocabulary	()	La propriété `vocabulary` définit le vocabulaire d'une liste de choix. Par défaut, c'est un tuple vide , sinon c'est une instance de la classe `DisplayList`.
enforceVocabulary	0	La propriété `enforceVocabulary` permet de contrôler la bonne utilisation des valeurs d'une liste de choix définie par la propriété `vocabulary`.
searchable	0	Indique si un champ est indexé par la méthode `SearchableText`. Par défaut, le champ ne peut pas être recherché via le Web. Attention : si un champ protégé est indexé, n'importe quel utilisateur ayant accès au document et faisant une recherche sur un mot contenu dans ce champ, pourra en déduire que le mot recherché est contenu dans l'un des champs auxquels il n'a pas accès
index	None	Syntaxe : `<index_type>` ou `<index_type>:schema`. Si un type d'index est défini, il est automatiquement créé dans `portal_catalog`. Si l'index est spécifié avec `:schema`, le champ fera partie des métadonnées de `portal_catalog`, et sera donc accessible dans les résultats d'une recherche.
required	0	Un champ obligatoire doit être rempli dans un formulaire. Par défaut, un champ peut être vide.
validators	()	Il s'agit d'une liste de méthodes de validation. Par défaut, la propriété est un tuple vide. S'il n'y a qu'une seule méthode de validation, elle peut être spécifié directement sans tuple.
default	None	Il s'agit de la valeur par défaut du champ.
default_method	None	Méthode appelée pour obtenir la valeur par défaut.
storage	AttributeStorage()	La propriété `storage` définit le type de stockage d'une valeur. Par défaut, la valeur est stockée en tant qu'attribut.

Nom de la propriété	Valeur par défaut	Notes
mutator	None	Méthode d'altération qui permet de modifier un champ d'une instance. Si le nom d'une méthode n'est pas fourni, Archetypes crée une méthode d'altération par défaut.
accessor	None	Méthode d'accès qui permet d'accéder au champ d'une instance sans le modifier. Si le nom d'une méthode n'est pas fourni, Archetypes crée une méthode d'accès par défaut.
edit_accessor	None	Une méthode qui retourne la valeur brute (raw value) d'un champ peut être définie ici.
multiValued	0	Définit si un champ peut contenir des valeurs multiples. C'est le cas lorsqu'on utilise des listes de choix dans lesquelles on peut sélectionner plusieurs valeurs.
isMetadata	0	Utilisé pour les champs de métadonnées. Pour le moment, cette information n'est utilisée que dans la méthode `filterFields` de `Schema`.
type	None	Le type est spécifié pour chaque classe dérivée de `Field`. Cette propriété ne doit pas être modifiée.
generateMode	veVc	Si la valeur de la propriété `generateMode` contient le caractère `c`, les méthodes d'accès et d'altération du champ sont définies dans la classe. Les modes **v**, **e** et **V** ne sont pas utilisés dans Archetypes. Cette propriété ne doit pas être modifiée.
force	''	Non utilisée pour l'instant.

Les propriétés de `Field` sont réutilisées ou redéfinies dans tous les champs qui héritent de `Field`. Les différences pour chaque type de champs sont données ci-dessous.

Champ de type CMF ObjectField

Utilise des objets de contenu (CMF), dont le type de contenu, et éventuellement un workflow associé, a été défini. Ne peut servir que pour des types de contenu fondés sur `BaseFolder`.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	FileWidget	Valeur possible : • LabelWidget
storage	storage	<Storage ObjectManagedStorage>	
type		object	

Propriétés spécifiques

Nom	Type	Défaut	Notes
portal_type		File	
workflowable		1	
default_mime_type		application/octet-stream	

Champ de type DateTimeField

Sert à stocker des dates.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	CalendarWidget	Valeurs possibles : • LabelWidget • CalendarWidget (défaut)
default	datetime		
type		datetime	

Champ de type ReferenceField

Sert à stocker des références vers d'autres objets Archetypes.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	ReferenceWidget	Valeur possible : • LabelWidget
type		reference	

Propriétés spécifiques

Nom	Type	Défaut	Notes
destination			L'endroit où créer l'objet si la propriété `addable` est active.
relationship			Définit le type de relation à établir pour ce champ. Exemples : `KnowsAbout` ou `Owns`.
allowed_types		()	La liste définit tous les types d'objets que l'utilisateur peut référencer. Indique aussi les types qui peuvent être ajoutés directement depuis un widget `reference`, lorsque la propriété `addable` est active.
addable		0	Indique si les types de la propriété `allowed_types` peuvent être `addable` et pas seulement `referencable`
allowed_type_column		portal_type	

Champ de type TextField

Sert à stocker du texte qui peut être transformé.

Propriétés modifiées

Nom	Type	Défaut	Notes
default	string		
type		text	

Propriétés spécifiques

Nom	Type	Défaut	Notes
primary		0	
default_content_type		text/plain	
allowable_content_types		('text/plain',)	Utilisé dans les widgets `TextArea` et `Visual` pour proposer à l'utilisateur le choix du format à appliquer au contenu saisi.
default_output_type		text/plain	Utilisé seulement dans la méthode get, pour choisir le type MIME à appliquer au contenu s'il n'est pas précisé.

Champ de type ImageField

Sert à stocker des images.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	ImageWidget	Valeurs possibles : • LabelWidget • ImageWidget (défaut)
default	string		
type		image	

Propriétés spécifiques

Nom	Type	Défaut	Notes
original_size			
max_size			
default_content_type		image/gif	
sizes		{'thumb': (80, 80)}	
image_class			
allowable_content_types		('image/gif', 'image/jpeg')	

Champ de type FloatField

Sert à stocker des valeurs décimales.

Propriétés modifiées

Nom	Type	Défaut	Notes
default	string	0.0	
type		float	

Champ de type StringField

Sert à stocker une chaîne de caractères.

Propriétés modifiées

Nom	Type	Défaut	Notes
default	string		
type		string	

Propriétés spécifiques

Nom	Type	Défaut	Notes
default_content_type		text/plain	

Champ de type BooleanField

Sert à stocker des valeurs booléennes.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	BooleanWidget	Représenté par une case à cocher. Valeurs possibles : • LabelWidget • BooleanWidget (défaut)
default	boolean		
type		boolean	

Champ de type FileField

Sert à stocker des fichiers.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	FileWidget	Valeurs possibles : • FileWidget (défaut) • LabelWidget
type		file	

Propriétés spécifiques

Nom	Type	Défaut	Notes
primary		0	
default_content_type		text/plain	

Champ de type IntegerField

Sert à stocker des nombres entiers.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	IntegerWidget	Valeurs possibles : • ReferenceWidget • IntegerWidget (défaut) • LabelWidget
default	integer	0	
type		integer	

Propriétés spécifiques

Nom	Type	Défaut	Notes
size		10	La taille maximale d'un champ Entier est de 10 chiffres par défaut.

Champ de type FixedPointField

Sert à stocker des valeurs décimales.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	DecimalWidget	Valeurs possibles : • DecimalWidget (défaut) • LabelWidget
validators	validators	isDecimal	
default	string	0.0	
type		fixedpoint	

Propriétés spécifiques

Nom	Type	Défaut	Notes
precision		2	

Champ de type LinesField

Sert à stocker des lignes de texte.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	LinesWidget	Valeurs possibles : • SelectionWidget • KeywordWidget • LabelWidget • LinesWidget (défaut) • MultiSelectionWidget
default	string	[]	
multiValued	boolean	0	
type		lines	

Champ de type ComputedField

Champ qui ne peut être utilisé qu'en mode lecture et qui affiche une valeur calculée par une expression Python.

Propriétés modifiées

Nom	Type	Défaut	Notes
widget	widget	ComputedWidget	Valeurs possibles : • LabelWidget • ComputedWidget (défaut)
storage	storage	<Storage ReadOnlyStorage>	
type		computed	
mode	string	r	

Propriétés spécifiques

Nom	Type	Défaut	Notes
expression			Evalué sur l'objet pour calculer la valeur.

Aide-mémoire pour les widgets

▸ http://plone.org/documentation/archetypes/arch_widget_quickref_1_2_5_rc4

Widget de type LabelWidget

Affiche un label.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/label
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type StringWidget

Boîte de saisie d'une chaîne de caractères sur une ligne.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/string
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
maxlength	integer	255
size		30
show_content_type		0
description		

Widget de type IntegerWidget

Boîte pour la saisie de valeurs entières (Integer).

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1

Nom	Type	Valeur par défaut
macro		widgets/integer
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
maxlength		255
size		5
show_content_type		0
description		

Widget de type DecimalWidget

Boîte de saisie de nombres décimaux.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/decimal
postback	boolean	1
label		
thousands_commas		0
visible		{'edit': 'visible', 'view': 'visible'}
whole_dollars		0
maxlength		255
dollars_and_cents		0
size		5
show_content_type		0
description		

Widget de type BooleanWidget

Case à cocher.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/boolean
postback	boolean	1
label		

Nom	Type	Valeur par défaut
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type LinesWidget

Boîte de saisie d'une liste de valeurs, chacune étant placée sur une ligne.

Propriétés

Nom	Type	Valeur par défaut
rows	integer	5
modes		('view', 'edit')
populate	boolean	1
macro		widgets/lines
postback	boolean	1
cols	integer	40
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type SelectionWidget

Liste de choix avec un seul choix possible, proposée sous forme de liste déroulante (`format=select`), ou sous forme de menu (`format=radio`). Par défaut (`format=flex`), une liste déroulante est utilisée s'il y a plus de trois options, sinon c'est un menu.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
format		flex (autres choix : select et radio)
macro		widgets/selection
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type MultiSelectionWidget

Liste de choix avec plusieurs choix possibles, proposée sous forme de liste déroulante (`format=select`), ou sous forme de menu (`format=checkbox`).

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
format		select (autre choix : checkbox)
macro		widgets/multiselection
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
size		5
show_content_type		0
description		

Widget de type TextAreaWidget

Boîte de saisie de texte.

Propriétés

Nom	Type	Valeur par défaut
rows	integer	5
modes		('view', 'edit')
populate	boolean	1
format		0
macro		widgets/textarea
postback	boolean	1
cols	integer	40
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type VisualWidget

Boîte de saisie utilisant un éditeur WYSIWYG.

Propriétés

Nom	Type	Valeur par défaut
rows	integer	25
modes		('view', 'edit')
populate	boolean	1
format		0
macro		widgets/visual
postback	boolean	1
cols	integer	80
label		
visible		{'edit': 'visible', 'view': 'visible'}
width		507px
height		400px
show_content_type		0
description		

Widget de type RichWidget

Boîte de saisie utilisant un éditeur WYSIWYG. Offre le choix entre plusieurs formats de texte et permet de télécharger un fichier texte.

Propriétés

Nom	Type	Valeur par défaut
rows	integer	5
modes		('view', 'edit')
populate	boolean	1
format		1
macro		widgets/rich
postback	boolean	1
cols	integer	40
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type EpozWidget

Boîte de saisie utilisant l'éditeur WYSIWYG Epoz.

Propriétés

Nom	Type	Valeur par défaut
rows		5
modes		('view', 'edit')
populate	boolean	1
format		0
macro		widgets/epoz
postback	boolean	1
cols		40
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type FileWidget

Boîte de saisie utilisée pour télécharger un fichier.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/file
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		1
description		

Widget de type ImageWidget

Boîte de saisie utilisée pour télécharger et afficher une image.

Propriétés

Nom	Type	Valeur par défaut
display_threshold		102400
modes		('view', 'edit')
populate	boolean	1
macro		widgets/image

Nom	Type	Valeur par défaut
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		1
description		

Widget de type CalendarWidget

Boîte de saisie d'une date.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
format		
macro		widgets/calendar
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type PasswordWidget

Boîte de saisie d'un mot de passe.

Propriétés

Nom	Type	Valeur par défaut
modes		('edit',)
populate	boolean	0
macro		widgets/password
postback	boolean	0
label		
visible		{'edit': 'visible', 'view': 'visible'}
maxlength		255
size		20
show_content_type		0
description		

Widget de type KeywordWidget

Boîte de saisie pour un mot-clé.

Propriétés

Nom	Type	Valeur par défaut
vocab_source		portal_catalog
modes		('view', 'edit')
populate	boolean	1
macro		widgets/keyword
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
size		5
show_content_type		0
roleBasedAdd		1
description		

Widget de type ComputedWidget

Affiche une valeur calculée, donc sans boîte de saisie.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/computed
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type IDWidget

Boîte de saisie pour le nom court d'un objet.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
display_autogenerated		1
macro		widgets/zid

Nom	Type	Valeur par défaut
postback	boolean	1
is_autogenerated		isIDAutoGenerated
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Widget de type ReferenceWidget

Boîte de saisie pour les références entre les objets Archetypes.

Propriétés

Nom	Type	Valeur par défaut
modes		('view', 'edit')
populate	boolean	1
macro		widgets/reference
postback	boolean	1
label		
visible		{'edit': 'visible', 'view': 'visible'}
show_content_type		0
description		

Index