Tiny
Documentation technique

v1.0

Sommaire

1. Introduction

1.1. Tiny en deux mots

Il existe aujourd'hui bien des outils de gestion de contenu Web sur le marché – OpenSource ou non. Tiny se distingue peut-être de ces outils par le fait qu'il est plus orienté « développeur » qu' « utilisateur final ». Une brève présentation des raisons qui ont motivé sa création précisera sans doute ce point.

C'est le projet de création de l'intranet de la Communauté Urbaine de Lille qui est à l'origine de la création de Tiny. Les éléments clés du cahier des charges de ce projet sont les suivants :

- outil de gestion de contenu permettant à chaque direction d'être autonome dans la mise en ligne du contenu, sans dépendre du savoir-faire d'un informaticien.
- édition de contenu via une interface WYSIWYG
- indépendance graphique totale de chaque site
- intégration aisée de modules métier dans chaque site
- mise en place éventuelle de parties privées/publiques dans chacun des sites
- impératif de réutilisabilité des modules : tout développement créé spécifiquement pour un site doit pouvoir être réutilisé dans n'importe quel autre, moyennant un rhabillage éventuel, mais sans duplication de code

L'équipe projet affectée à cette réalisation étant très réduite (un chef de projet fonctionnel, un chef de projet technique, et un développeur), il s'est donc avéré immédiatement nécessaire de s'appuyer sur :

- un outil de gestion de contenu à la fois multisite et très souple graphiquement et structurellement,
- un modèle de développement permettant une séparation traitement / affichage totale,
- un framework permettant d'accélérer la création et l'intégration de modules métiers.

Tiny est donc à la fois un environnement de développement et un outil de gestion de contenu :

- un développeur ayant des connaissances en HTML et XML peut créer très rapidement un site au sein d'une entité Tiny, y définir des utilisateurs, leur affecter des droits, définir une structure éditoriale, créer un environnement graphique spécifique … etc.
- un développeur PHP peut créer et intégrer aisément des modules réutilisables
- un non-informaticien sachant utiliser un traitement de texte peut saisir et mettre en ligne du contenu mis en forme (tableaux, images …).

1.2. Objectif du document

Ce document a pour objectif de donner les éléments techniques clés permettant à un développeur d'aborder Tiny dans de bonnes conditions. Il ne s'agit pas de décrire ici chaque classe de l'application, mais plutôt de détailler celles qui permettront de comprendre toutes les autres. Ce document est donc très complémentaire du code de l'application.

2. Architecture technique

2.1. Plateforme

Tiny s'installe sur une plateforme LAMP (Linux Apache MySql PHP). La version de PHP doit être supérieure à 4.3.

Exemples de configurations testées :

Linux :
- RedHat 7.3
- MySql 3.23.56
- PHP 4. 3
- Apache 1.3

Windows :
- EasyPHP 1.6 (ou +)

2.2. PEAR

2.2.1. Packages

Tiny utilise un certain nombre de packages de PEAR [http://pear.php.net]. Les plus utilisés sont :
- HTML_Template_PHPLIB : Moteur de templates
- PEAR_DB : Data Abstraction Layer
- Auth : Authentification
- HTML_QuickForm : Génération de formulaires
- HTML_Table : Génération de tables HTML

2.2.2. Normes de codage

L'ensemble des classes Tiny est rédigé conformément aux normes de codage de PEAR. Ces normes sont accessibles à l'adresse suivante :
http://pear.php.net/manual/fr/standards.php

Le seul ajout à cette norme est un préfixage simple des variables :
- **a :** array
- **o :** objet
- **i :** numérique
- **s :** chaîne de caractères
- **b :** booleen

Exemples :
- $iNbResultats
- $sPrenom
- …etc.

2.3. Modélisation

2.3.1. Formalisme objet

Tiny est une application entièrement objet – autant que faire se peut en PHP : il y a une page PHP par site, une page PHP pour l'outil d'administration, et le reste est entièrement codé dans des objets.

2.3.2. Modèle MVC2

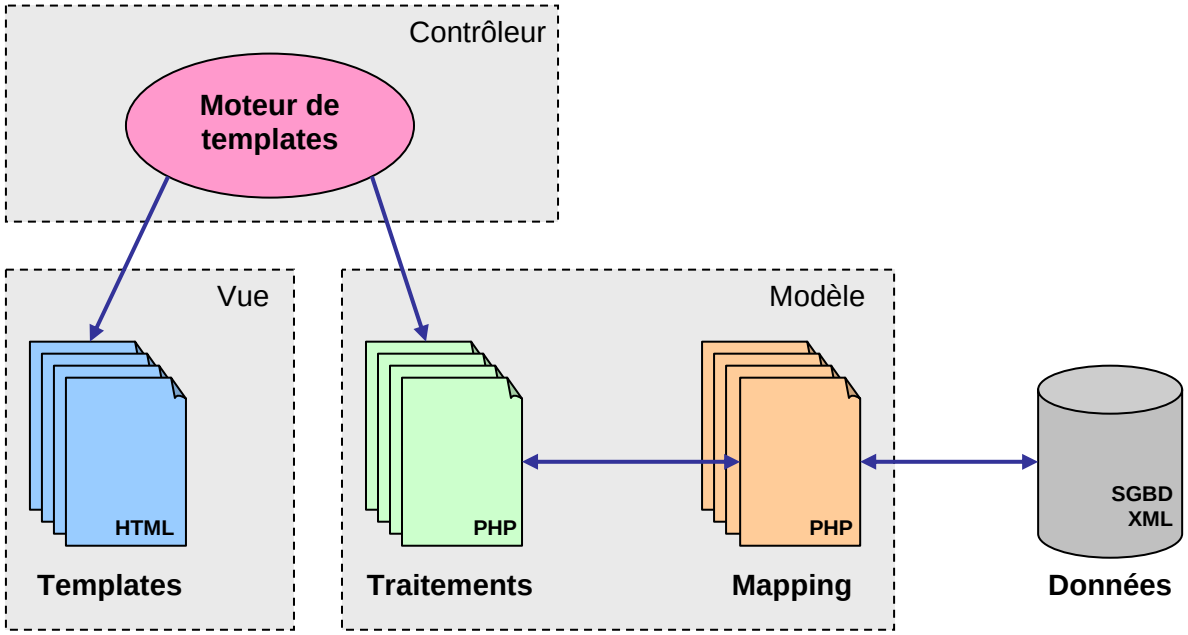
Pour mémoire, le modèle MVC [Model View Controller, ou « Modèle Vue Contrôleur » in french] propose une séparation du code en deux couches :

- **Modèle :** couche métier, dans lesquelles sont codés les règles de gestion et les traitements
- **Vue :** couche de présentation, qui assure le rendu des objets de type Modèle.

Un objet de type **Contrôleur** associe quant à lui un objet Modèle et l'objet Vue associé.
Le modèle MVC2 est identique au modèle MVC, à ceci près qu'un contrôleur unique est utilisé.

Tiny suit le modèle MVC2 :

- le rôle du contrôleur unique est assuré par une instance du moteur de templates HTML_Template_PHPLIB, fourni dans PEAR
- l'ensemble du code PHP constitue la couche modèle. Deux types d'objets sont représentés : des objets qui assurent le mapping des données (en base ou dans des fichiers XML), et des objets qui contiennent l'ensemble des règles de gestion de l'application
- la partie vue est constituée de l'ensemble des templates fournis au moteur et remplis par les objets Modèle



2.3.3. Remarque

Ce modèle rigoureux impose dans certains cas simples d'écrire beaucoup de code pour pas grand-chose. Dans un souci de simplification du code, et d'allègement de son exécution, certains écarts légers par rapport au modèle seront parfois observables dans le code source de Tiny.

3. Description fonctionnelle

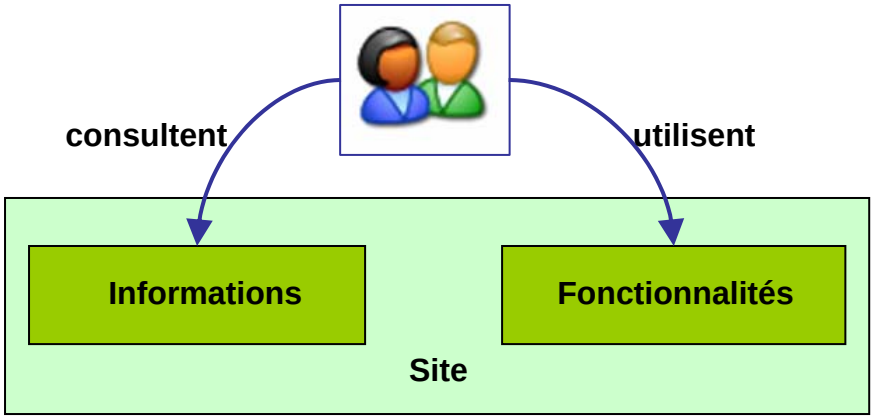
Ce chapitre ne constitue pas la documentation fonctionnelle de Tiny. Il se limite à préciser certains aspects de l'application, et à définir quelques éléments de terminologie.

3.1. Structure d'un site

Ce chapitre d'une trivialité déconcertante se limitera à présenter de façon schématique les concepts suivants :
- site
- information
- fonctionnalité
- utilisateur

3.1.1. Site

Un **site** internet contient de l'**information** et des **fonctionnalités** qu'il met à disposition d'**utilisateurs**.

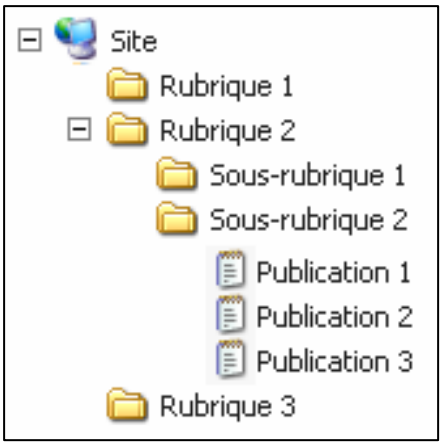


Il est important de bien distinguer informations et fonctionnalités :
- **informations :** ensemble des informations textuelles qui véhiculent le propos du site dans sa globalité
- **fonctionnalités :** modules autonomes utilisés pour répondre à un besoin précis

Par analogie avec un livre :
- l'information est l'ensemble du texte
- les fonctionnalités sont la table des matières, l'index, la numérotation des pages …etc.

3.1.2. Information

L'information se doit d'être adaptée au support internet. Elle doit être concise, et facile à trouver. Cela nécessite donc une structuration forte du contenu : les textes sont découpés en éléments simples, et classés dans une arborescence de thématiques. Dans toute la suite, on notera :
- **rubriquage** : l'arborescence des thématiques d'un site
- **rubrique** : un axe, un thème, …
- **article** (ou **publication**) : un texte (élément simple cité ci-dessus)



Tout l'aspect gestion de contenu dans l'outil d'administration Tiny consiste donc à :
- gérer le rubriquage
- gérer les articles à l'intérieur de chaque rubrique

3.1.3. Fonctionnalité

Chaque fonctionnalité doit pouvoir être intégrée graphiquement et hiérarchiquement à un site. L'intégration graphique consiste à pouvoir définir un habillage qui respecte les chartes graphiques et ergonomiques du site. L'intégration hiérarchique consiste à pouvoir définir un emplacement pour chaque fonctionnalité dans le plan du site, afin que les éléments de navigation mis en place permettent à l'internaute de les atteindre. L'outil d'administration permet donc d'associer une fonctionnalité à une rubrique.

3.1.4. Utilisateurs

Un internaute accède à l'information, et aux fonctionnalités. Un utilisateur de l'outil d'administration accède à cet outil, accède à la gestion d'un site en particulier, et accède ensuite à chacun des modules de gestion mis en place pour le site choisi (gestion des rubriques, des articles, et gestion des fonctionnalités du site). Le système de gestion des droits mis en place dans Tiny permet de régir l'ensemble de ces conditions d'accès. Il sera détaillé dans le chapitre 3.3.

3.2. Logique fonctionnelle d'affichage d'une page

Un site étant constitué d'information et de fonctionnalités, l'affichage d'une page d'un site repose uniquement sur le contexte d'utilisation :
- quel élément d'information l'internaute est-il en train de visualiser ?
- quelle fonctionnalité est-il en train d'utiliser ?

Aussi, dans Tiny, chaque site est constitué d'une seule page PHP. L'affichage de cette page est déterminé par les variables **rub** (rubrique courante), **art** (article courant) et **module** (module courant) :
- http://.../site.php : aucun contexte précisé : on affiche la page d'accueil
- http://.../site.php?rub=2 : l'internaute consulte le contenu de la rubrique n°2
- http://.../site.php?rub=2&art=3 : l'internaute consulte l'article n°3 de la rubrique n°2
- http://.../site.php?module=Glossaire : l'internaute utilise le glossaire

Le moteur se base donc sur l'analyse de ces trois paramètres pour déterminer ce qu'il doit afficher.

L'algorithme de décision (simplifié) est le suivant :

Si « module » est renseigné **Alors**
    Affichage du module
Sinon Si « rub » est renseigné **Alors**
    Si un module est associé à cette rubrique **Alors**
        Affichage du module associé
    Sinon Si « art » est aussi renseigné **Alors**
        Affichage de l'article
    Sinon
        Affichage de la rubrique
    Fin Si
Sinon
    Affichage de la page d'accueil
Fin Si

3.3. Gestion des droits

3.3.1. Identification des utilisateurs

L'utilisateur d'un site est repéré par un identifiant : c'est une chaîne d'une longueur maximale de 30 caractères. Tout utilisateur non authentifié utilise implicitement un identifiant par défaut : « **default** », accessible via la constante PHP « **C_DEFAULT_USER** ».
Les utilisateurs sont stockés dans la table « **personne** ».

3.3.2. Authentification

Par défaut, l'authentification est effectuée en base, par lecture dans la table « personne ». C'est le composant PEAR « Auth » qui se charge de la mise en œuvre de l'authentification. Il est possible de mettre en place très simplement un autre mode d'authentification (LDAP …) : se reporter pour cela à la documentation du composant « Auth ».

Remarque : si la table « personne » ne sert plus à authentifier les utilisateurs, elle doit néanmoins être remplie car elle est utilisée dans la gestion des droits des utilisateurs.

3.3.3. Privilèges des utilisateurs

Les privilèges sont stockés dans la table … « **privilège** » (ce qui dénote à la fois d'une logique implacable et d'un manque d'imagination flagrant).

On utilise 7 privilèges pour définir le niveau d'un utilisateur :
- **0** : Aucun privilège
- **1** : Utilisateur
- **2** : Tâches basiques
- **4** : Rédacteur
- **8** : Valideur
- **16** : Tâches sensibles
- **32** : Administrateur

Pour chaque site on peut accorder à chaque utilisateur un ou plusieurs privilèges. Dans tous les cas, le niveau d'un utilisateur sur un site est représenté par la somme des privilèges qu'il possède sur ce site. Une personne possédant tous les privilèges sauf Administrateur aura donc le niveau 1 + 2 + 4 + 8 + 16 = 31 : son niveau restera donc inférieur à celui d'un administrateur (32).

Remarque : Le libellé des privilèges est fortement teinté « gestion de contenu ». Ils sont néanmoins utilisables en dehors de ce contexte.

3.3.4. Droits des utilisateurs

Un droit est l'association d'un privilège sur un site pour une personne. Les droits sont stockés dans la table « **droit** ».

Pour limiter l'accès à une fonctionnalité ou à une action, on pourra mettre en place deux types de contrôle :
- accès limité aux utilisateurs qui possèdent exactement un privilège requis
- accès limité aux utilisateurs dont le niveau est supérieur à un seuil

3.3.5. Ventilation des droits sur les rubriques

Dans le cas de la gestion du contenu d'un site, admettons que l'on définisse la règle suivante : pour pouvoir créer un article, l'utilisateur doit posséder le privilège « Rédacteur ». C'est un peu rédhibitoire car cela autorise à priori n'importe quel rédacteur à créer des articles partout dans le site. Pour permettre de ventiler les privilèges des utilisateurs sur chaque rubrique, on introduit la notion d' « acteur de rubrique ». En clair, on définit pour chaque rubrique une liste d'utilisateurs parmi ceux qui possèdent des privilèges sur le site courant : seuls ces utilisateurs pourront exercer leurs privilèges sur la rubrique.

Exemple : soit le site des amateurs de pâtés de foie en boîte. Il possède deux rubriques : « Olida » et « Hénaf ». Deux utilisateurs, Georges et Robert, possèdent le privilège « Rédacteur » sur ce site. On veut que Robert ne puisse créer des articles que dans la rubrique « Olida », et que Georges puisse le faire partout.
Il suffit pour ce la que Robert ne soit acteur que de la rubrique « Olida », et que Georges soit acteur des deux.

Les acteurs sont stockés dans la table « **rubrique_acteur** », et les rubriques dans la table « **rubrique** » (ce modèle de données est décidément lumineux …).

3.3.6. Corollaire : création d'un espace privé

Pour créer un espace privé dans un site, c'est-à-dire une branche du site accessible seulement à un groupe de personnes distinctes, on utilise de la même façon le principe d'acteur de rubriques.

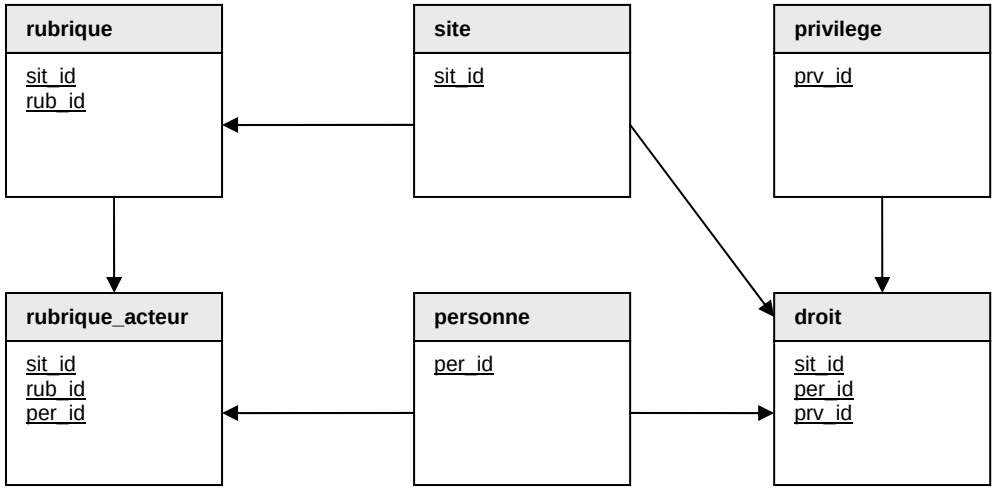
Par défaut, l'utilisateur « Tout le monde » (identifiant « default »), est utilisateur de chaque site Tiny. Il ne possède sur le site que le privilège « Aucun privilège » (0). Un utilisateur qui ne possède que ce privilège peut uniquement consulter l'information contenue dans ce site, et exécuter les fonctionnalités qui ne requièrent aucun droit.

Sur un site, pour atteindre du contenu, on spécifie une rubrique et éventuellement un article (http://.../site.php?rub=2&art=2). Avant d'afficher la page correspondante, le moteur vérifie

que l'utilisateur courant a le droit de la visualiser : le moteur vérifie donc que l'utilisateur courant est acteur de la rubrique demandée.

Par conséquent :
- pour rendre une rubrique **Publique** il faut que l'utilisateur « Tout le monde » en soit acteur.
- pour rendre une rubrique **Privée** il faut que l'utilisateur « Tout le monde » n'en soit pas acteur : seuls les acteurs de la rubrique pourront y accéder.

3.3.7. Modèle de données



4. Logique technique d'affichage d'une page

En développement PHP procédural, les pages HTML sont le plus souvent générées verticalement :

- Traitements
- Headers
- Bandeau haut (affichage successif de tous les éléments qui le composent)
- Contenu (affichage successif de tous les éléments qui le composent)
- Bandeau bas (affichage successif de tous les éléments qui le composent)

Pour éviter d'avoir des fichiers trop énormes, les éléments sont souvent séparés en plusieurs fichiers, et inclus au bon endroit de la page source. L'approche reste cependant verticale.

Avec Tiny, l'approche est différente. Au risque de contrarier le lecteur attentif et féru d'anticipation, il ne s'agit pas non plus d'une approche horizontale (bien que « l'approche horizontale de développement » soit un concept très attirant, il faut bien l'avouer).

Il s'agit plutôt d'une approche gigogne, comme les poupées russes. On définit une page comme un ensemble de blocs, chaque bloc pouvant lui-même contenir d'autres blocs.

4.1. Découpage en blocs

Prenons l'exemple d'une page (au hasard) :

Premier niveau

On considère que la page est constituée de trois zones :

- bandeau haut
- bandeau central
- bandeau bas

Deuxième niveau
Le bloc « bandeau haut » contient, à part du HTML statique (logo, liens fixes …) un bloc « menu DHTML »

Le bloc « bandeau central » contient 2 blocs :
- bandeau gauche
- zone centrale

Le bloc « bandeau bas » quant à lui est uniquement statique

Troisième niveau
Le bloc « bandeau gauche » contient
- un bloc de navigation par rubriques
- un bloc d'information contextuelle

Le bloc « zone centrale » contient
- un article de bienvenue
- un bloc d'information contextuelle en bas
- un bloc d'information contextuelle à droite

Ce qui nous donne en résumé :

L'approche par blocs, ou conteneurs, consiste à décrire l'arborescence de blocs qui constitue la page.

4.2. Qu'est-ce qu'un bloc

Chaque bloc est constitué de trois éléments :
- une classe PHP dans laquelle sont codés les traitements liés au bloc
- un fichier HTML : template d'affichage du bloc
- un fichier XML (facultatif) : qui décrit la liste des sous blocs et leur nature

4.2.1. Notion de template

Les templates utilisables avec Tiny doivent être compatibles avec le moteur de templates PHPLIB (voir description complète en Annexe). Pour rester simple, un template est un fichier HTML dans lequel on peut définir :
- des clés : zones que l'on pourra remplacer par des valeurs
- des blocs : portions de HTML dupliquables

Les clés sont des mots sans espace encadrés par des accolades (`{NOM_CLE}`, `{ELEMENT}`)
Les blocs sont encadrés par `<!-- BEGIN nom_bloc -->` et `<!-- END nom_bloc -->`.

Exemple : le template suivant permet de gérer l'affichage d'une liste :
- affichage d'un titre
- affichage des éléments de la liste par duplication d'un bloc « element »

```
<table>
 <tr>
  <td colspan=2><b>{TITRE}</b></td>
 </tr>
 <!-- BEGIN element -->
 <tr>
  <td><img src="puce.gif"></td>
  <td>{LIBELLE_ELEMENT}</td>
 </tr>
 <!-- END element -->
</table>
```

4.2.2. Classes PHP

(Ce chapitre est un peu ardu).
Le moteur travaille avec trois classes de base qui sont :
- **Container** : qui dispose des méthodes permettant de manipuler un template (remplacement de clés, duplication de blocs)
- **Feeder** : classe dont un attribut est une référence sur un objet Container, et qui est capable de remplir automatiquement le template associé à cet objet Container en fonction du fichier de définition XML fourni.
- **Portlet** : classe qui hérite de Container et qui peut mettre en relation un Feeder, un fichier XML de définition, et un template

Reprenons calmement :

4.2.2.1. Container

L'ancêtre Container dispose des méthodes permettant de manipuler le template :
- remplacement d'une clé par une valeur
- duplication d'un bloc

Pour utiliser ces méthodes, il faut donc lui fournir un template. On trouvera donc dans le code du template une ligne signifiant « je vais utiliser ce fichier HTML comme template ».

Exemple : Voici un exemple de conteneur permettant de remplir le template « liste.htm » du chapitre 4.2.1. Le conteneur utilise ce template pour afficher une liste de Georges célèbres.

```
class Georges extends Container
{
    /**
     * Constructeur
     */
    function Georges(&$oEnv)
    {
        // Association du template
        $this->setTemplate("liste.htm");
        // Affichage du titre
        $this->setElement("TITRE", "Liste des Georges célèbres");
        // Définition du bloc "element", qu'on nommera "elt"
        // dans la suite
        $this->setBlock($this->sTemplate, "element", "elt");
        // Ajout de Georges Petibidon
        $this->setVar("LIBELLE_ELEMENT", "Georges Petibidon");
        $this->parse("elt", "element", true);
        // Ajout de Georges de la Jungle
        $this->setVar("LIBELLE_ELEMENT", "Georges de la Jungle");
        $this->parse("elt", "element", true);
    }
}
```

NB : l'exemple de code ne fonctionne pas tel quel : quelques lignes de code qui nuiraient à la compréhension du chapitre ont été supprimées.

L'inconvénient majeur lié à l'utilisation directe d'un Container est que le template y est directement lié : le code PHP n'est donc pas réutilisable, puisqu'il est lié à une mise en forme.

D'où la necessité de séparer le code en deux :
- une classe dans laquelle on code le remplissage d'un template dont on ne connaît que la structure (nom des clés, et nom des blocs dupliquables) : classe de type **Feeder**
- l'instance d'une classe **Portlet** pour associer le Feeder et le fichier template

4.2.2.2. Feeder

Cette classe sera instanciée par une instance d'un Portlet. Le Feeder possède en attribut une référence sur l'objet qui l'a créé. Il passera donc par le Portlet pour remplir le template. On rappelle que l'objet Portlet hérite de Container : il dispose donc bien des méthodes adéquates.

Exemple : On reprend l'exemple des Georges célèbres. Deux choses sont à repérer : on passe par l'objet appelant pour écrire dans le template, et le code est placé dans une fonction init(), et non plus dans le constructeur (l'explication de ce point sera donnée au chapitre 4.2.2.4).

```php
class Georges extends Feeder
{
   /**
     * méthode init()
     */
    function init()
    {
        // Association du template
        $this->oCon->setTemplate("liste.htm");
        // Affichage du titre
        $this->oCon->setElement("TITRE", "Liste des Georges célèbres");
        // Définition du bloc "element", qu'on nommera "elt"
        // dans la suite
        $this->oCon->setBlock($this->oCon->sTemplate, "element", "elt");
        // Ajout de Georges Petibidon
        $this->oCon->setVar("LIBELLE_ELEMENT", "Georges Petibidon");
        $this->oCon->parse("elt", "element", true);
        // Ajout de Georges de la Jungle
        $this->oCon->setVar("LIBELLE_ELEMENT", "Georges de la Jungle");
        $this->oCon->parse("elt", "element", true);
    }
}
```

4.2.2.3. Portlet

Pour utiliser un Feeder, on passe par l'instanciation d'un objet Portlet, qui attend comme paramètres le nom d'un Feeder, et un fichier template. C'est dans le constructeur du Portlet que la classe Feeder va être instanciée, et que le template va être rempli.

```php
new Portlet($oEnv, "Georges", "liste.htm");
```

Nous avons donc bien dissocié le code PHP du fichier template, et donc réglé le problème de la réutilisabilité du code PHP :

```php
// Utilisation de la classe Georges avec le template "liste.htm"
... new Portlet($oEnv, "Georges", "liste.htm");

// Utilisation de la classe Georges avec le template "liste2.htm"
... new Portlet($oEnv, "Georges", "liste2.htm");
```

Tout ceci est bien joli, mais il reste un inconvénient assez lourd : toutes les classes que l'on utilise et les templates qu'on leur associe semblent être disséminées un peu partout dans le code PHP. Pour éviter cela, on automatise les choses en utilisant des fichiers de définition. Le principe de ces fichiers au format XML est de décrire par quoi il faut remplacer les clés d'un template.

4.2.2.4. Autoremplissage

Dans le paragraphe précédent nous avons vu que la classe Portlet permet d'associer un Feeder et un template. On peut en plus associer un fichier XML de définition des clés qui se trouvent dans le template.

Exemple : Nous avons un template « liste.htm » qui nous permet d'afficher des listes, et un Feeder « Georges » qui connaît des Georges célèbres. Admettons que nous ayons créé sur le même modèle une classe « Robert » qui connaîsse des Roberts célèbres, et qui sache les afficher via le même template.
L'exemple suivant montre comment faire pour créer un composant « DeuxListes » qui sache afficher un titre, et les listes des Georges et des Roberts célèbres :

Liste de personnalités célèbres	
Liste des Georges célèbres ❖ Georges Petibidon ❖ Georges de la Jungle	**Liste des Robert célèbres** ❖ Robert Petibidon ❖ Robert de la Jungle

Le composant DeuxListes va utiliser les composants Georges et Robert pour afficher les deux listes : on réutilise le code déjà produit. Donc le savoir faire de l'objet DeuxListes se réduit à l'affichage d'un titre.

```
class DeuxListes extends Feeder
{
   /**
     * méthode drawTitre()
     */
    function drawTitre ()
    {
        // On renvoie le titre
        return "Liste de personnalités célèbres";
    }
}
```

Pour afficher un tableau avec un titre en haut, et les deux listes en dessous, on a besoin d'un template : « deuxlistes.htm » :

```
<table>
 <tr>
  <td colspan=2 align=center><b>{TITRE_DL}</b></td>
 </tr>
 <tr>
  <td>{LISTE_1}</td>
  <td>{LISTE_2}</td>
 </tr>
</table>
```

Pour remplir ce template, on doit donc :
- remplacer {TITRE_DL} par le titre, fourni par la méthode « drawTitre » de notre objet DeuxListes
- remplacer {LISTE_1} par le feeder « Georges » associé au template « liste.htm »
- remplacer {LISTE_2} par le feeder « Robert » associé au template « liste.htm »

C'est ce qu'on décrit en XML de la façon suivante (fichier « deuxlistes.xml »)

```xml
<?xml version="1.0"?>
<root>
 <bloc nom="TITRE_DL" type="methode" data="drawTitre" />
 <bloc nom="LISTE_1"  type="portlet" data="Georges" template="liste.htm" />
 <bloc nom="LISTE_2"  type="portlet" data="Robert"  template="liste.htm" />
</root>
```

Pour générer les deux listes, il suffit d'écrire :

```php
new Portlet($oEnv, "DeuxListes", "deuxlistes.htm", "deuxlistes.xml");
```

En écrivant cela, les opérations suivantes sont effectuées par le moteur (dans le constructeur de l'objet Portlet) :
- Instanciation de l'objet DeuxListes
- Exécution du constructeur de DeuxListes : il n'y en a pas
- Exécution du constructeur de Feeder (père de DeuxListes) : lancement de la méthode « init() »
- La méthode « init() » parse le fichier de définition « deuxlignes.xml » et traite chacun des blocs qui y sont définis :
- Traitement du 1er bloc : la clé {TITRE_DL} est remplacée par la valeur de retour de la méthode « drawTitre » de DeuxListes
- Traitement du 2ème bloc : la clé {LISTE_1} est remplacée par le HTML produit par l'instruction « `new Portlet($oEnv, "Georges", "liste.htm");` » (liste des Georges célèbres)
- Traitement du 3ème bloc : la clé {LISTE_2} est remplacée par le HTML produit par l'instruction « `new Portlet($oEnv, "Robert", "liste.htm");` » (liste des Robert célèbres)

Par conséquent, nous avons résolu le dernier problème : l'ensemble des noms de classe et des noms de templates utilisés ne sont plus disséminés dans le code PHP, mais centralisés dans un fichier XML.

Remarque : Dans le chapitre 4.2.2.2 (feeder Georges), le code de remplissage du template était placé dans une méthode « init » : nous avons donc surchargé la méthode « init » du parent pour débrayer le mécanisme d'autoremplissage. En d'autre termes, dans le constructeur du parent (Feeder) la méthode init exécutée est celle du fils (Georges).

4.2.3. Fichiers XML

Un fichier XML de définition sert donc à définir les blocs d'un template et la façon dont le moteur doit les remplir. Les blocs sont définis dans des tags <bloc>. Les attributs suivants peuvent être utilisés :

Attribut	Obligatoire	Signification
nom	oui	Nom de la clé dans le template
type	oui	Méthode de remplissage. Les valeurs admises sont : - string : chaîne de caractère - methode : méthode du Feeder associé - function : fonction globale - portlet : portlet
data	oui	Selon le type : - string : chaîne de caractères qu'il faut substituer à la clé - methode : nom de la méthode à appeler - function : nom de la fonction à appeler - portlet : nom du portlet à instancier
template	oui pour type portlet	Nom du template à associer au portlet
deffile	non	Nom du fichier de définition à associer au portlet
arguments	non	Arguments éventuels à fournir au constructeur du Portlet, ou à la méthode appelée

Remarques :

- On distingue « methode » définie pour un objet, et « fonction » définie globalement.
- Les arguments doivent être passés sous la forme « `nom1=>valeur1|nom2=>valeur2…` »

Exemple :

```
<?xml version="1.0"?>
<root>
 <bloc nom="CLE1" type="string"   data="Bonjour à tous !" />
 <bloc nom="CLE2" type="methode"  data="drawTruc" arguments="truc=>coucou" />
 <bloc nom="CLE3" type="fonction" data="getCurrentDateFr" />
 <bloc nom="CLE4" type="portlet"  data="Georges"  template="liste.htm" />
</root>
```

4.3. Affichage d'une page

L'affichage d'une page n'est ni plus ni moins qu'une mise en musique des mécanismes explicités dans le chapitre précédent. Dans le fichier PHP de chaque site, on se contente :

- d'inclure les classes nécessaire au moteur
- de définir des constantes applicatives
- d'initialiser un objet Environment (la mystérieuse variable $oEnv qui apparaît dans les exemples précédents, et qui sera largement détaillée plus loin)
- de créer un portlet de type « Page », associé au template d'affichage global de la page et à un fichier de définition

L'ensemble de la page est construit de proche en proche grâce au mécanisme d'auto-remplissage.

5. Gabarits

5.1. Notion de gabarit

Dans toute la suite on nommera « gabarit » l'association d'un template et d'un fichier de définition. Le gabarit décrit donc la mise en forme d'un bloc. Dans Tiny, on distingue trois familles de gabarits, qui reposent sur les mêmes mécanismes, mais qui sont utilisés à des fins différentes :

- les gabarits de portlets
- les gabarits de rubriques
- les gabarits d'articles

5.2. Gabarit de portlet

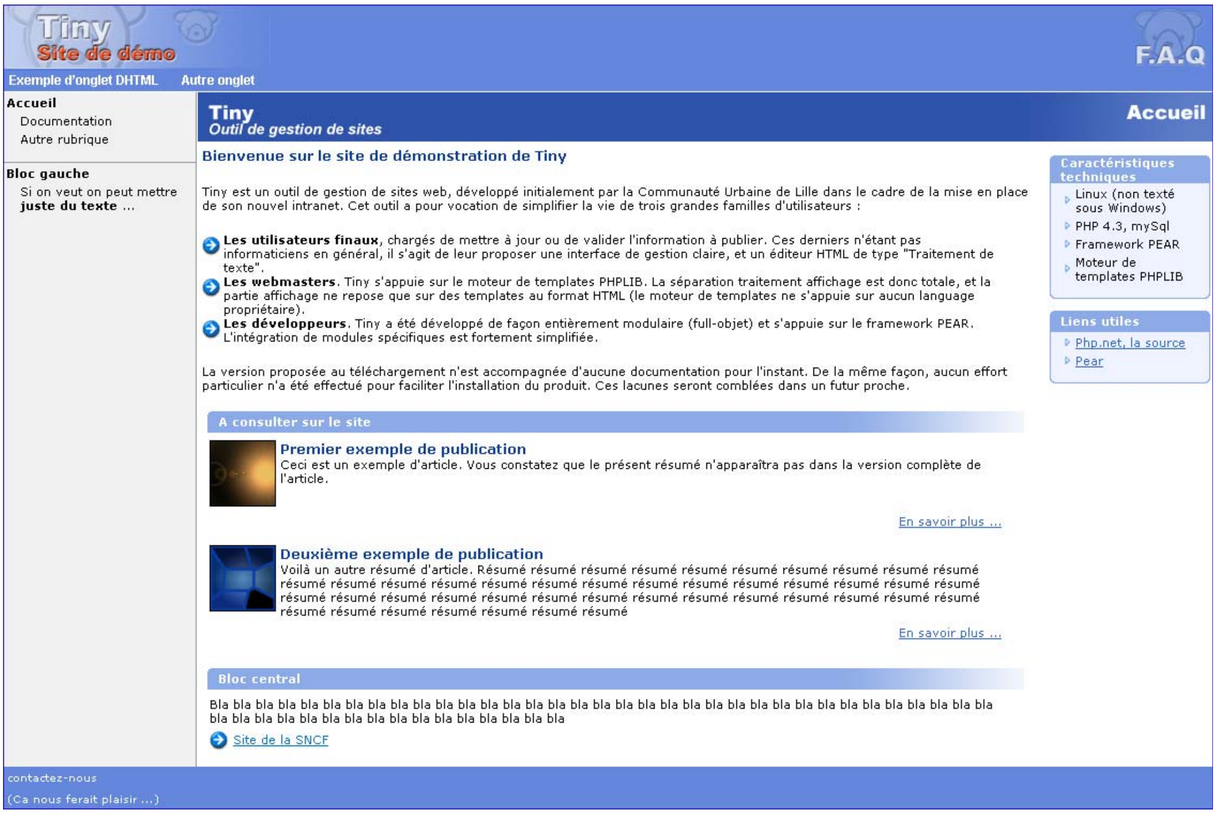
Les gabarits de portlets ont été largement détaillés dans le chapitre précédent. On notera simplement que par abus de langage, le terme Portlet désignera tantôt la classe Portlet elle-même, tantôt le bloc généré par un Feeder, un template, et éventuellement un fichier de définition.

5.3. Gabarit de rubrique

5.3.1. Principe

La logique d'affichage d'une page (paragraphe 3.2) décrit le fait que lorsque l'utilisateur se positionne sur une rubrique (http://.../site.php?rub=2) le moteur affiche le contenu de cette rubrique. Typiquement, une rubrique peut contenir des sous-rubriques et des articles. Un gabarit de rubrique définit les éléments à afficher lorsque l'on se positionne sur la rubrique. Chaque site possède donc son propre jeu de gabarits de rubriques, en fonction des différents comportements d'affichages qu'il nécessite : par exemple, pour le site de démo qui accompagne le code Tiny, on constate que structurellement la page d'accueil est différente des pages intérieures :

Le gabarit de rubrique associé à la rubrique « Documentation » propose l'affichage :
- à gauche : d'un bloc de navigation
- à droite : d'un sommaire constitué des articles en ligne de la rubrique
- au centre : du premier article de la rubrique ou de l'article sélectionné dans le sommaire



Pour la rubrique accueil, on affiche cette fois :
- à gauche : un bloc de navigation et les blocs contextuels de la rubrique définis en position gauche (*)
- à droite : les blocs contextuels de la rubrique définis en position droite (*)
- au centre : l'article en masterpage de la rubrique (*), les blocs contextuels de la rubrique définis en position centrale (*), et les raccourcis définis pour la rubrique (*)

(*) voir documentation fonctionnelle

5.3.2. Mise en place

Un gabarit de rubrique est donc constitué d'un template d'affichage (HTML correspondant à la partie centrale de l'écran) qui comporte autant de clés qu'il y a d'éléments fonctionnels à afficher, et d'un fichier XML qui décrit par quoi chaque clé doit être remplacée.

Chaque gabarit est identifié par un nom (en minuscules, sans espace ni caractères spéciaux, de 30 caractères au maximum) utilisé :
- en base de donnée dans la table « gabarit », champ « gab_id » (Exemple : « accueil »)
- pour le nom du template (exemple : « accueil.htm »)
- pour le nom du fichier de définition (exemple : « accueil.xml »)

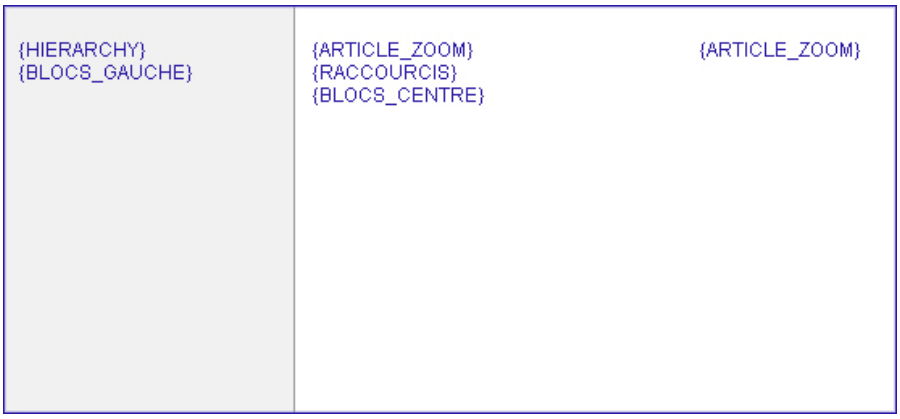
A noter que chaque site possède son propre jeu de gabarit.

5.3.3. Exemple

Le gabarit utilisé pour la rubrique Accueil du site de démo est identifié par « **accueil** ».

On trouve donc :

- dans la table « gabarit » un enregistrement dont l'identifiant est « accueil » pour le site « demo ».
- un fichier « accueil.htm » dont la structure est la suivante :



- un fichier de description « accueil.xml » :

```
<?xml version="1.0"?>
<root>
 <bloc nom="HIERARCHY" type="portlet" data="Hierarchy"
       template="hierarchy.htm" />
 <bloc nom="BLOCS_GAUCHE" type="portlet" data="BlocContextuel"
       template="bloccontextuel_gauche.htm"  arguments="pos=>g" />
 <bloc nom="ARTICLE_ZOOM" type="portlet" data="ArticleMasterpageOfRub"
       template="articlemasterpageofrub.htm"
       arguments="sit=>demo|rub=>1|short=>0" />
 <bloc nom="BLOCS_CENTRE" type="portlet" data="BlocContextuel"
       template="bloccontextuel_centre.htm"  arguments="pos=>c" />
 <bloc nom="BLOCS_DROITE" type="portlet" data="BlocContextuel"
       template="bloccontextuel_droite.htm"  arguments="pos=>d" />
 <bloc nom="RACCOURCIS"   type="portlet" data="ArticleRaccourcis"
       template="articleraccourcis.htm" />
</root>
```

5.4. Gabarit d'article

5.4.1. Principe

Les besoins éditoriaux des sites sont généralement tous différents. De même, au sein d'un même site, des structures d'articles différentes peuvent être nécessaires. Par exemple, un article de type « brève » sera constitué généralement d'un titre et d'un morceau de texte, alors qu'un article journalistique pourra être composé d'un surtitre, d'un titre, d'un sous-titre, d'un chapeau d'introduction, d'un corps de texte …etc. On doit donc, pour chaque site, pouvoir définir plusieurs structures d'articles différentes.
En outre, on a souvent besoin d'afficher un article en version courte et d'afficher un lien « en savoir plus » qui mène vers la version longue de l'article. On doit donc être capable de

définir, pour chaque structure d'article – ou gabarit – ce qui doit être affiché en version courte, et ce qui doit l'être en version longue.

Un gabarit d'article va donc être défini par :
- un fichier XML de description de la structure de l'article
- un template d'affichage de l'article en version courte
- un template d'affichage de l'article en version longue

5.4.2. Mise en place

Chaque gabarit est identifié par un nom (en minuscules, sans espace ni caractères spéciaux, de 30 caractères au maximum) utilisé :
- en base de donnée dans la table « gabarit », champ « gab_id » (Exemple : « edito »)
- pour le nom du template pour la version longue (exemple : « edito.htm »)
- pour le nom du template pour la version courte (exemple : « edito_short.htm »)
- pour le nom du fichier de définition (exemple : « edito.xml »)

Remarques :
- Le nom du template en version courte est toujours composé de l'identifiant suivi de « _short ».
- Les enregistrements de la table gabarit sont typés par le champ « gab_type » : « r » pour rubrique et « a » pour article.
- Comme pour les rubriques, chaque site possède son propre jeu de gabarits d'articles.

5.4.3. Fichier de définition

Les fichiers de définition des gabarits article sont différents ce ceux que l'on a vus jusqu'à maintenant. En effet, ils ne sont pas associés à un Feeder, mais ils décrivent simplement la nature de ce que le rédacteur va devoir saisir :
- texte non mis en forme
- texte mis en forme

C'est en fonction de cette description que l'on va pouvoir générer dynamiquement le formulaire de création d'un article.

Attribut	Obligatoire	Signification
nom	oui	Nom de la clé dans le template
libelle	oui, sauf pour type portlet	Libellé à afficher dans le formulaire de saisie/modification
type	oui	Type : - string : texte non mis en forme : la saisie s'effectuera dans un input de type text - html : texte mis en forme : la saisie s'effectuera via le miniword - portlet : portlet : aucun champ de saisie associé
data	oui pour type portlet	Nom du portlet à instancier
template	oui pour type portlet	Nom du template à associer au portlet
deffile	Non	Nom du fichier de définition à associer au portlet
arguments	Non	Arguments éventuels à fournir au constructeur du Portlet, ou à la méthode appelée

A la différence des autres gabarits, les tags <bloc> possèdent cette fois des valeurs (voir fichier exemple en page suivante). Ces valeurs serviront à présenter un aperçu du gabarit au rédacteur au moment de choisir un gabarit d'article.

5.4.4. Exemple

Prenons l'exemple du gabarit « default » du site de démonstration.

Le fichier de définition est le suivant (« default.xml ») :

```
<?xml version="1.0"?>
<root>
 <bloc nom="ART_TITRE" libelle="Titre" type="string">
  Titre de l'article
 </bloc>
 <bloc nom="ART_RESUME" libelle="Résumé" type="html">
  <![CDATA[<i>Résumé résumé résumé résumé résumé</i>]]>
 </bloc>
 <bloc nom="ART_IMAGE" libelle="Image pour le résumé" type="html">
  <![CDATA[IMAGE]]>
 </bloc>
 <bloc nom="ART_CHAPEAU" libelle="Chapeau" type="html">
  <![CDATA[chapeau chapeau chapeau chapeau chapeau]]>
 </bloc>
 <bloc nom="ART_CORPS" libelle="Corps" type="html">
  <![CDATA[corps corps corps corps corps corps corps]]>
 </bloc>
 <bloc nom="ART_PJ" libelle="Pièces jointes" type="portlet"
       data="Pj" template="pj.htm">
 </bloc>
</root>
```

On définit donc :
- un titre en texte brut
- quatre zones de texte mis en forme
- une zone de type portlet pour faire appel à un portlet d'affichage des pièces jointes

On décide d'afficher en version courte :
- titre
- résumé
- image

Et en version longue :
- titre
- chapeau
- corps
- pièces jointes

Le template de la version courte est donc (« default_short.htm ») :

```
<table cellspacing=0 cellpadding=0 border=0 width='100%'>
<tbody>
 <tr>
  <td width=60 valign=top>{ART_IMAGE}</td>
  <td width=10> </td>
  <td width=100% valign=top>
   <span class=art_titre>{ART_TITRE}</span><br>
   {ART_RESUME}
  </td>
 </tr>
 <tr>
  <td colspan=3><img height=5 src="demo/public/images/p.gif" width=1></td>
 </tr>
</tbody>
</table>
```

Le template de la version longue est (« default.htm ») :

```
<table cellspacing=0 cellpadding=0 border=0 width='100%'>
<tbody>
 <tr>
  <td><span class=art_titre>{ART_TITRE}</span></td>
 </tr>
 <tr>
  <td><img height=5 src="demo/public/images/p.gif" width=1></td>
 </tr>
 <tr>
  <td>{ART_CHAPEAU}</td>
 </tr>
 <tr>
  <td><img height=5 src="demo/public/images/p.gif" width=1></td>
 </tr>
 <tr>
  <td>{ART_CORPS}</td>
 </tr>
 <tr>
  <td><img height=5 src="demo/public/images/p.gif" width=1></td>
 </tr>
 <tr>
  <td>{ART_PJ}</td>
 </tr>
</tbody>
</table>
```

Remarques :

- Les articles saisis sont stockés dans le même format que le fichier de définition : les valeurs par défaut sont remplacées par les valeurs saisies par l'utilisateur.
- Dans un fichier de définition la valeur d'un bloc de type « html » est du HTML brut généré via le miniword. C'est pour cette raison que ces valeurs sont définies en CDATA (encadrées par « `<![CDATA` » et « `]]>` »)

6. Logique du moteur pour rechercher les fichier

Dans tout ce qui précède nous avons cité un certain nombre de fichiers : classes PHP, fichiers de définition XML, templates HTML … Le lecteur attentif (encore lui !) aura sans doute remarqué que :

- nulle part n'a été précisé où ils se trouvent dans l'arborescence du site
- lorsque l'on nomme une ressource au moteur (dans un fichier XML) on utilise un nom de classe ou de fichier sans en préciser le chemin d'accès

L'objet de ce chapitre est donc de préciser à la fois où placer ces fichiers, et comment le moteur va les rechercher. Ce dernier point est important car il précise la logique de partage d'un même portlet sur plusieurs sites.

6.1. Description de l'arborescence

6.1.1. Vue globale

tiny	Pages principales (demo.php, back.php …)
core	**Moteur**
private	
classes	Conteneurs + classes du moteur
portlets	Portlets (classes héritant de Feeder)
config	Fichiers de configuration du moteur
inc	Includes
lib	(inutilisé)
templates	Templates des conteneurs
portlets	Templates des portlets
public	
images	Images
scripts	Fichiers javascripts (.js)
styles	Feuilles de styles
back	**Backoffice**
private	
classes	Conteneurs + classes du site courant
portlets	Portlets (classes héritant de Feeder)
config	Fichiers de configuration du site
inc	(inutilisé)
templates	Templates des conteneurs
portlets	Templates des portlets
public	
images	Images
scripts	Fichiers javascripts (.js)
styles	Feuilles de styles
spaw	Source du miniword
demo	**Site**
data	Articles
private	
classes	Conteneurs + classes du site courant
portlets	Portlets (classes héritant de Feeder)
config	Fichiers de configuration du site
templates	Templates des conteneurs
articles	Templates des articles
formulaires	Templates des formulaires
portlets	Templates des portlets
rubriques	Templates des rubriques
public	
images	Images
scripts	Fichiers javascripts (.js)
styles	Feuilles de styles

Le répertoire Tiny (où le répertoire racine d'apache) contient une page PHP par site de l'instance Tiny (« **demo.php** » …), et une autre pour l'outil d'administration (« **back.php** »). Il contient en outre :

- un dossier « **core** » qui contient les fichiers du moteur Tiny, et les fichiers communs à tous les sites de l'instance
- un dossier « **back** » qui contient les fichiers spécifiques à l'outil d'administration des sites
- un dossier par site (« **demo** ») qui contient les fichiers spécifiques à ce site

Chacun de ces dossiers contient :

- un dossier « **data** » pour le stockage des articles (absent pour « core » et « back »)
- un dossier « **private** » pour le stockage des fichiers PHP et des gabarits
- un dossier « **public** » pour le stockage des images, des feuilles de style et des fichiers javascript

6.1.2. Dossier « data »

Ce dossier contient :

- un dossier « **sandbox** » dans lequel sont stockés les fichiers joints aux articles
- un dossier par rubrique

Le nom de chaque dossier rubrique correspond à l'identifiant en base de la rubrique correspondante (numérique). Le classement des dossiers des rubriques correspond à l'arborescence hiérarchique des rubriques (si « 4 » est une sous rubrique de « 2 », le répertoire « 2 » aura un sous répertoire « 4 »).

Chaque dossier rubrique contient les fichiers XML des articles de la rubrique. Le nom de ces fichiers est composé de l'identifiant de l'article en base suivi de « _ » puis de la date de création de l'article au format « AAAAMMJJHHMMSS ».

6.1.3. Dossier « private/classes »

Ce dossier contient les fichiers source des classes PHP (autres que Feeder) spécifiques au site courant :

- classes métier
- classes de type Container

A noter que l'on créé un fichier PHP par classe, et que le nom du fichier est identique à celui de la classe (casse comprise).

6.1.4. Dossier « private/classes/portlets »

Ce dossier contient les fichiers source des classes portlets (classes héritant de Feeder) du site courant.

6.1.5. Dossier « private/classes/config»

Ce dossier contient les fichiers de configuration du site courant. Celui du répertoire « core » contient quatre fichiers communs à tous les sites :

- **host.inc** : constantes spécifiques à l'environnement d'installation

- **constantes.inc** : constantes applicatives
- **classes.inc** : effectue l'inclusion des classes utiles à tous les sites
- **include.inc** : fichier inclus dans chaque page PHP : effectue les includes des fichiers nécessaires

Le moteur inclut par défaut tous les fichiers de configuration du noyau (« core »). Il inclut ensuite (si existence) les fichiers « **classes.inc** » et « **constantes.inc** » du répertoire du site courant. Dans ces fichiers sont définies les classes et les constantes spécifiques au site courant.

6.1.6. Dossier « private/inc »

Ce dossier contient deux fichiers particuliers :
- **error.inc** : positionne les error_handlers PHP et PEAR
- **function.inc** : définit des fonctions globales (principalement traitement de dates)

6.1.7. Dossier « private/template »

Ce dossier contient les templates d'affichage des objets de type Container (fichiers htm uniquement).

6.1.8. Dossier « private/template/articles »

Ce dossier contient les gabarits article du site courant : on rappelle que pour un gabarit dont l'identifiant en base est « gab », les trois fichiers qui constituent le gabarit sont :
- **gab.xml** : définition de la structure de l'article
- **gab.htm** : affichage de l'article en version longue
- **gab_short.htm** : affichage de l'article en version courte

6.1.9. Dossier « private/template/rubriques »

Ce dossier contient les gabarits rubrique du site courant : on rappelle que pour un gabarit dont l'identifiant en base est « gab », les deux fichiers qui constituent le gabarit sont :
- **gab.xml** : définition des blocs du template
- **gab.htm** : template d'affichage

6.1.10. Dossier « private/template/portlets »

Ce dossier contient les gabarits des portlets utilisés dans le site courant : on rappelle que ces gabarit sont constitués :
- d'un ou plusieurs fichiers templates (plusieurs si dans le site courant on utilise le même portlet avec des habillages différents)
- d'un fichier de définition XML optionnel (si on utilise le mécanisme d'autoremplissage).

Le jeu de fichiers constituant le gabarit d'affichage d'un portlet est placé dans un sous-répertoire dont le nom est identique à celui de la classe Feeder associée, mais **en minuscules**.

Exemple : pour le portlet « Hierarchy » constitué à partir de la classe Feeder « Hierarchy », le gabarit est constitué par un seul fichier « hierarchy.htm ». Ce fichier se trouve dans le répertoire « private/templates/portlets/hierarchy ».

6.2. Recherche d'un Portlet

Lors de la création d'un objet Portlet, on fournit le nom d'un objet Feeder (« MaClasse »), le nom d'un template (« maclasse.htm »), et éventuellement le nom d'un fichier de définition (« maclasse.xml »). La logique de recherche de ces fichiers dans l'arborescence est décrite dans ce chapitre. Pour plus d'informations, se reporter au code qui se situe dans le constructeur de l'objet Portlet (« core/private/classes/Portlet.class »)

6.2.1. Recherche de la classe

Le système va d'abord rechercher le fichier « MaClasse.class » dans le répertoire des portlets du site courant (« **site**/private/classes/portlets »). S'il le trouve il l'utilise, sinon il va le rechercher dans le répertoire des portlets du noyau (« **core**/private/classes/portlets »). S'il ne le trouve pas non plus, une erreur est déclenchée.

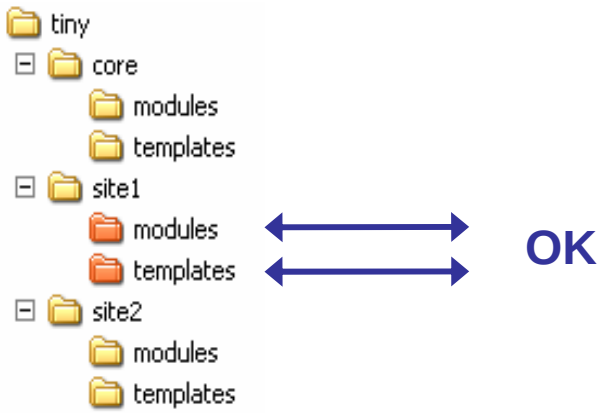
6.2.2. Recherche du gabarit

Le système recherche les fichiers qui constituent le gabarit (« maclasse.htm » et « maclasse.xml ») uniquement dans le répertoire des gabarits du portlet « MaClasse » du site courant (« **site**/private/classes/portlets/**maclasse** »). S'il les trouve il les utilise, sinon une erreur est déclenchée.

6.2.3. Corollaire : partage des portlets

Cette logique de recherche permet de mutualiser les portlets.

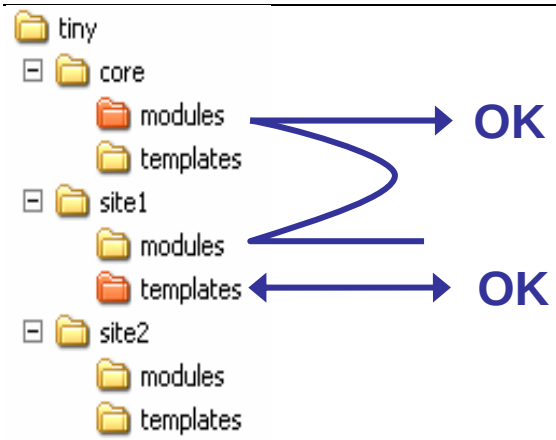
1ère étape : création d'un portlet spécifique à un site : les fichiers sont localisés dans le répertoire du site en question



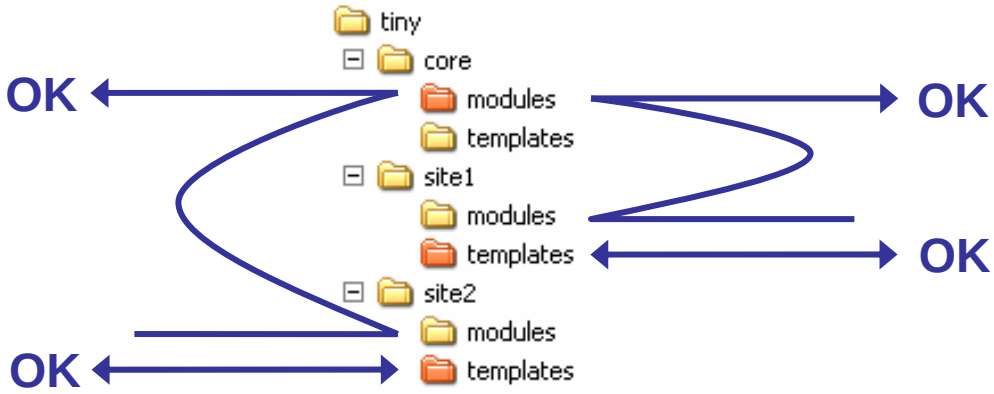
Si l'un des fichiers du site fait référence à ce portlet, le système peut le retrouver et l'afficher.

2ème étape : rendre ce portlet générique, et donc utilisable dans les autres sites de l'instance Tiny. Il suffit de déplacer le fichier PHP du répertoire des portlets du site d'origine vers le répertoire des portlets du noyau.

On vérifie sur le schéma suivant que ce déplacement n'impacte pas le bon fonctionnement du portlet dans le site d'origine :



3ème étape : utilisation du même portlet sur plusieurs site. C'est possible moyennant la présence d'un template d'affichage dans chaque site qui utilise le portlet :



6.3. Recherche d'un Container

L'exécution d'un module peut être appelée de deux façons :

- explicite : http://.../site.php?module=MonModule : on spécifie que le contenu de la page est codé dans le Container « MonModule.class »
- implicite : http://.../site.php?rub=2, alors qu'on a dans l'outil d'administration associé la rubrique 2 avec le module « MonModule ».

Dans les deux cas, le moteur sait qu'il doit afficher le module « MonModule ». Il doit donc rechercher le fichier « MonModule.class ».

Pour plus d'informations, se reporter au code qui se situe dans la méthode « getContent() » du portlet générique Page (« core/private/classes/portlets/Page.class »)

NB : on rappelle que dans le cas des objets de type Container, le template est spécifié directement dans le code : ce n'est pas au moteur d'aller le rechercher.

6.3.1. Recherche de la classe

Le système recherche le fichier source du conteneur uniquement dans le répertoire des classes du site courant (« **site**/private/classes »). S'il le trouve il l'utilise, sinon une erreur est déclenchée.

6.3.2. Corollaire

Le système ne prévoit donc pas de mise en commun des modules (Container), au contraire des Portlets. C'est logique puisque le template, et donc la mise en forme, est lié au Container.

6.4. Recherche d'un gabarit de rubrique

Lorsque l'on demande au système d'afficher une rubrique (« http://.../site.php?rub=2 »), il va d'abord rechercher en base l'identifiant du gabarit qui lui est associé. Cet identifiant (« gab ») lui permet de déduire le nom des fichiers qui constituent le gabarit (« gab.htm » et « gab.xml »). Le système va rechercher ces fichiers dans le répertoire des templates des rubriques du site courant (« **site**/private/templates/rubriques »). S'il les trouve il les utilise, sinon une erreur est déclenchée.

6.5. Recherche d'un Article

Lorsque l'on demande au système d'afficher un article (« http://.../site.php?rub=22&art=53 »), il va rechercher en base :

- l'identifiant du gabarit qui lui est associé. Cet identifiant (« gab ») lui permet de déduire le nom des fichiers qui constituent le gabarit (« gab.htm », « gab_short.htm » et « gab.xml »).
- le nom du fichier XML contenant les données de l'article (« 53_AAAAMMDDHHMMSS »)
- le chemin d'accès aux fichiers de la rubrique à partir du répertoire « data » (« 2/17/22 »)

6.5.1. Recherche du gabarit

Le système va rechercher les fichiers dans le répertoire des templates des articles du site courant (« **site**/private/templates/articles »). S'il les trouve il les utilise, sinon une erreur est déclenchée.

6.5.2. Recherche des données

Le système va reconstituer le chemin d'accès au fichier en concaténant :

- le répertoire des articles du site courant (« **site**/data »)
- le chemin d'accès aux fichiers de la rubrique à partir du répertoire « data » (« 2/17/22 »)
- le nom du fichier XML contenant les données de l'article (« 53_AAAAMMDDHHMMSS »)

S'il trouve le fichier il l'utilise, sinon une erreur est déclenchée.

7. Mise en cache des Portlets

7.1. Remarque préliminaire

Revenons sur l'exemple du Portlet « DeuxListes » présenté au chapitre 4.2.2.4. L'affichage de ce Portlet demande au système d'ouvrir les fichiers suivants :

- DeuxListes.class : source du feeder
- deuxlistes.htm : template du portlet DeuxListes
- deuxlistes.xml : fichier de définition du portlet
- Robert.class : source du feeder de la liste des Roberts
- liste.htm : template d'affichage de la liste des Roberts
- Georges.class : source du feeder de la liste des Georges
- liste.htm : template d'affichage de la liste des Georges

Les fichiers XML doivent être parsés, les fichiers PHP exécutés, les templates manipulés … Bref, c'est du boulot, et donc du temps CPU … D'où l'intérêt de pouvoir mettre en cache le code HTML généré par l'appel de DeuxListes.

7.2. Principe de la mise en cache

Mettre en cache quelque chose, c'est facile : pour poursuivre sur l'exemple de DeuxListes, il suffit de le stocker quelque part le HTML généré, et de dire au système d'utiliser ce HTML s'il existe au lieu d'instancier un portlet DeuxListes quand on lui demande.
Les choses sont un peu plus délicates lorsqu'on veut s'assurer que si on régénérait le portlet DeuxListes, on retomberait bien sur le même code HTML que celui qui est en cache.

Pour prendre un peu de distance par rapport à notre exemple, le rendu d'un portlet peut dépendre de six choses selon les cas :

- contenu d'une ou plusieurs tables de la base de données (si les donnée affichées proviennent d'une lecture en base)
- template d'affichage du portlet
- contexte d'utilisation
- articles
- rubriques
- arguments

Pour préciser les choses :

- si on change les données d'une table sur laquelle s'appuie un portlet, il ne faut plus utiliser sa version en cache
- si on modifie le template d'affichage d'un portlet, il ne faut plus utiliser sa version en cache
- si un portlet dépend du contexte d'utilisation, il faut pour ce portlet mettre en cache autant de versions qu'il y a de contextes d'utilisation (par exemple, le portlet Hierarchy (bloc gauche de navigation) dépend de la rubrique courante : on met en cache une version du portlet pour l'accueil, une version pour la rubrique 1, etc…)
- si un portlet dépend du contenu d'un ou plusieurs articles, et que des modifications sont apportées à cet article sans que la base soit impactée (modification du gabarit par exemple), il ne faut plus utiliser la version en cache
- si un portlet dépend du contenu d'une ou plusieurs rubriques, et que des modifications sont apportées à cette rubrique sans que la base soit impactée (modification du gabarit par exemple), il ne faut plus utiliser la version en cache
- si un portlet s'initialise avec des arguments, il faut pour ce portlet mettre en cache autant de versions qu'il y a de valeurs possibles pour ces arguments

En plus de ces dépendances, comme on le verra ultérieurement dans une analyse détaillée, un portlet peut s'appuyer sur des feuilles de styles et/ou des fichiers javascripts. C'est alors l'exécution du code PHP du portlet qui se charge d'inclure ces CSS et ces JS. On doit donc s'assurer de sauvegarder en plus du HTML la liste des fichiers client à inclure.

Ces contraintes font de la mise en cache des portlets une opération assez difficile à gérer …

7.3. Mise en œuvre

On utilise le composant PEAR « Cache » comme support de la gestion de cache Tiny. Ce composant peut travailler avec plusieurs conteneurs pour stocker les éléments mis en cache. Nous travaillons ici en base de donnée (cache stocké dans un champ blob). Seul ce support permet d'associer simplement des données descriptives à une donnée en cache.

7.3.1. Modèle de données

Deux tables sont utilisées dans le cadre de la mise en cache des portlets :
- table « **cache** » : stockage
- table « **cache_portlet** » : description des portlets à mettre en cache et de leurs dépendances

7.3.1.1. Table « cache »

Il s'agit de la table utilisée par le composant PEAR pour le stockage des données.

Champ	Type	
id	varchar(32)	Id de l'enregistrement
cachegroup	varchar(127)	Groupe d'appartenance
cachedata	blob	Données mises en cache
userdata	varchar(255)	Données complémentaires
expires	int	Date d'expiration
changed	timestamp	Date de modification

Le champ « id » est généré à partir d'un encodage md5 des données suivantes :
- id du site
- nom du portlet (classname)
- nom du fichier template
- nom et valeur des arguments
- nom et valeur du contexte (variables postées en GET ou POST)

Cet identifiant correspond donc à un cas d'utilisation précis du portlet.

Le champ « cachegroup » contient une chaîne constituée du nom du site et du nom du portlet. Ce champ permet d'effectuer des opérations de nettoyage telles que :
- vidage intégral du cache pour un site
- vidage intégral du cache pour un portlet sur un site

Le champ « userdata » est généré par un checksum md5 effectué sur les fichiers du répertoire contenant le gabarit d'affichage du portlet. Ce champ est utilisé pour déterminer si les fichiers du gabarit ont été modifiés depuis la mise en cache du portlet.

Le champ « expires » contient la date d'expiration du cache : les donnée sont utilisées si cette date n'est pas atteinte, effacées sinon.

7.3.1.2. Table « cache_portlet »

Cette table permet de spécifier les portlets à mettre en cache, ainsi que leurs dépendances.

Champ	Type	
prt_id	varchar(50)	Nom du portlet (classname)
prt_cacheable	tinyint	Flag permettant de débrayer la mise en cache d'un portlet
prt_cache_expires	int	Durée de validité du cache, en secondes
prt_formvars_dependance	varchar(128)	Variables de formulaire (contexte) dont le portlet dépend
prt_tables_dependance	varchar(128)	Tables dont le portlet dépend
prt_css_needed	varchar(128)	Feuilles de style dont le portlet dépend
prt_js_needed	varchar(128)	Javascripts dont le portlet dépend

Le champ « id » est le nom de la classe Portlet (sensible à la casse).

Le champ « prt_cacheable » est un flag permettant de débrayer la mise en cache du portlet correspondant :
- « 0 » : mise en cache désactivée
- « 1 » : mise en cache activée.

Le champ « prt_formvars_dependance » contient la liste des variables de contexte (variables de formulaires postées en GET ou en POST) dont le portlet dépend. Les noms de variables doivent être séparés par des « | ». La liste doit commencer et finir par un « | ».
Exemple : pour un portlet dépendant de la rubrique courante, de l'article courant, et de la valeur d'une variable « mode », on trouvera dans le champ prt_formvars_dependance la valeur : « |**rub**|**art**|**mode**| ».

Le champ « prt_tables_dependance » contient la liste des tables dont le portlet dépend. Les noms des tables doivent être séparés par des « | ». La liste doit commencer et finir par un « | ».
Exemple : pour un portlet dépendant des tables « rubrique », « article » et « piece_jointe », on trouvera dans le champ prt_tables_dependance la valeur : « |**article**|**rubrique**|**piece_jointe**| ».

Le champ « prt_css_needed» contient la liste des fichiers CSS dont le portlet a besoin. Cette liste est constituée sur le même modèle que ci-dessus (noms séparés par des « | », liste commence et finit par un « | »).

Le champ « prt_js_needed» contient la liste des fichiers JS dont le portlet a besoin. Cette liste est constituée sur le même modèle que ci-dessus (noms séparés par des « | », liste commence et finit par un « | »).

7.3.2. Paramétrage supplémentaire

La mise en cache des portlets peut être activée ou désactivée site par site. C'est le champ « sit_portlet_caching » de la table « site » qui détermine le mode courant :
- « 0 » : mise en cache désactivée
- « 1 » : mise en cache activée.

7.3.3. Objet PortletCache

Sur la page de chaque site, en début de code, on initialise un objet PortletCache. Cet objet mappe la table « cache_portlet » afin d'éviter de faire une lecture en base à chaque demande d'affichage d'un portlet.

Cet objet centralise toutes les lectures/écritures du cache. Il est utilisé à une endroit unique du code : dans la méthode « factory() » de l'objet Feeder.
Le code étant relativement bien commenté, l'algorithme étant facile à déduire des éléments ci-dessus, et le lecteur étant – comme je l'ai fait largement remarquer – particulièrement attentif, c'est sans aucun remords que j'arrête ici la documentation relative à la mise en cache des portlets.

8. Fichiers de configuration

8.1. Fichier « core/private/config/host.inc »

Ce fichier comprend deux parties :
- des constantes à redéfinir lors de l'installation de Tiny sur une nouvelle plateforme
- des constantes relatives à l'installation qu'il ne faut pas modifier

Ces deux parties sont clairement commentées.
Les constantes sont les suivantes (celles en bleu sont à redéfinir) :

Constante	Description
C_MAX_FILE_SIZE_MO	Taille maximale des fichiers uploadables (en Mo)
SITE_HOST	Nom de l'hôte (ce qu'il y a entre "http://" et le répertoire de l'application).
SITE_DIR	Répertoire de base de l'application
C_BASE_ROOT	Chemin complet du répertoire de base de l'application
DB_DEFAULT_HOST	Host sur lequel est installée la base mysql
DB_DEFAULT_BASE	Nom de la base
DB_DEFAULT_USER	Nom d'utilisateur
DB_DEFAULT_PASS	Mot de passe
DB_DEFAULT_TYPE	Type (mysql)
C_MAX_FILE_SIZE	Taille maximale des fichiers uploadables en bytes (déduite de C_MAX_FILE_SIZE_MO)
C_SPAW_URL	Url du miniword
C_DEFAULT_USER	Utilisateur par défaut
C_BACKOFFICE_SID	Identifiant du site « Outil d'administration »
C_BASE_URL	Url de base de l'application
C_CORE_CLASSES_PATH	Suffixe du répertoire des classes du noyau (core/private/classes/)
C_CORE_TPL_PATH	Suffixe du répertoire des templates du noyau (core/private/templates)
C_ADO_PATH	Chemin vers la bibliothèque ADODB (inutilisée jusqu'à maintenant)
SUFIX_CLASSES_PATH	Constantes décrivant les noms de répertoires utilisés (templates, portlets, sandbox …etc.). Ces constantes permettent de modifier les chemins définis dans cette documentation.
SUFIX_CONFIG_PATH	
SUFIX_ACTIONS_PATH	
SUFIX_ROOT_SCRIPTS	
SUFIX_IMG_PATH	
SUFIX_TPL_PATH	
SUFIX_DEF_PATH	
SUFIX_TBL_PATH	
SUFIX_JSC_PATH	
SUFIX_CSS_PATH	
SUFIX_UPL_PATH	
SUFIX_CACHE_PATH	
SUFIX_DATA_PATH	
SUFIX_ARTICLES	
SUFIX_RUBRIQUES	
SUFIX_FORMULAIRES	

8.2. Fichier « core/private/config/include.inc »

Fichier dans lequel on effectue tous les includes nécessaire. Dans la page de chaque site, c'est le seul fichier que l'on inclut …

8.3. Fichier « xxx/private/config/classes.inc »

Ce fichier permet d'inclure les classes nécessaires au bon fonctionnement de l'application.

8.3.1. Fichier « core/private/config/classes.inc »

Inclusion des classes mères Tiny, des classes PEAR nécessaires, et des classes métier nécessaires à tous les sites (Article.class, Site.class …etc.).

8.3.2. Fichier « site/private/config/classes.inc »

Inclusion des classes métier spécifiques à un site

8.3.3. Remarque

Les classes des modules et des portlets sont incluses dynamiquement : elles n'ont pas à l'être explicitement.

8.4. Fichier « xxx/private/config/constantes.inc »

Ce fichier permet de définir des constantes applicatives.

8.4.1. Fichier « core/private/config/constantes.inc »

Ce fichier permet de définir les constantes applicatives utilisées par l'ensemble des sites de l'instance. Ces constantes sont les suivantes

Constante	Description
C_PRV_AUCUN	0 : aucun privilège
C_PRV_UTILISATEUR	1 : utilisateur
C_PRV_BASIQUE	2 : modules basiques
C_PRV_REDACTEUR	4 : rédacteur
C_PRV_VALIDEUR	8 : valideur
C_PRV_SENSIBLE	16 : modules sensibles
C_PRV_ADMINISTRATEUR	32 : administrateur
C_ETAT_REDACTION	« redaction » : état « en rédaction » pour un article
C_ETAT_WORKFLOW	« workflow » : état « en validation » pour un article
C_ETAT_ONLINE	« online » : état « en ligne » pour un article
C_PROG_NAME	Nom de l'application
C_DATE_MOINS_INFINI	Date considérée comme moins l'infini (14/12/1901)
C_DATE_PLUS_INFINI	Date considérée comme plus l'infini (19/01/2038)
C_REPLACEMENT_TPL	Template de remplacement
C_REPLACEMENT_KEY	Clé de remplacement
C_MSG_WORKFLOW	« workflow » : Message de type « workflow de validation »
C_MSG_FO	« fo » : Message de type « front-office »
C_MSG_BO	« bo » : Message de type « back-office »
C_MSG_EMIS	« E » : Statut d'un message « émis »
C_MSG_REPONDU	« R » : Statut d'un message « répondu »
C_MSD_NON_LU	« N » : Statut d'un message « le destinataire n'a pas lu »
C_MSD_LU	« L » : Statut d'un message « le destinataire a lu »
C_MSD_REPONDU	« R » : Statut d'un message « le destinataire a répondu »

.

8.4.2. Fichier « site/private/config/constantes.inc »

Ce fichier permet de définir des constantes applicatives spécifiques à un site.

9. Informations techniques importantes

Ce chapitre détaille quelques éléments importants à saisir pour faciliter la compréhension générale du fonctionnement technique de l'outil. Une lecture du code des quelques objets cités ici est vivement conseillée pour compléter le propos qui suit.

9.1. Objets clés

9.1.1. Objet Environment

9.1.1.1. Principe

Tiny est une application « presque full objet ». « Presque », parce qu'on utilise une page PHP pour instancier un premier objet, « full » parce que tous les mécanismes internes, les modules que vous réutiliserez, et ceux que vous écrirez seront encapsulés dans des objets.

Une caractéristique de ce formalisme est qu'à l'intérieur d'un objet on ne « voit » que l'objet. C'est un peu contraignant dans le cadre d'une application web, car on a souvent besoin d'avoir une vision de l'extérieur (contexte applicatif, session, objets instanciés ailleurs …etc.).

Une solution consiste à utiliser la directive PHP « global » dès que le besoin d'utiliser une donnée extérieure à un objet se fait sentir. Cette solution possède au moins deux désavantages :

- c'est un peu lourdingue
- si en PHP 4.0 on avait écrit 622 fois « global $HTTP_POST_VARS ; » dans 622 scripts PHP, on doit pour migrer en PHP 4.3 remplacer cette ligne 622 fois par « global $_POST; », tout en redoutant à l'avance la future migration en PHP 5 qui nous obligera à supprimer ces 622 lignes, sachant que le tableau $_POST deviendra superglobal.

Une seconde solution consiste :

- à créer un objet dans lequel on stocke tout le contexte applicatif
- à initialiser cet objet une fois en début de page
- à fournir une référence sur cet objet au constructeur de chaque objet créé
- à enregistrer cette référence en attribut de chaque objet

Ainsi tous les objets peuvent accéder à tout le contexte applicatif via cette référence.

Je vous le donne Emile : l'objet « Environment » est la concrétisation de cette deuxième solution, dont l'élégance n'aura pas manqué d'émouvoir même le lecteur le plus attentif.

Remarque : le rédacteur de la présente documentation à l'air d'être content de lui lorsqu'il évoque « l'élégance » de la solution. Il n'en est rien : l'idée n'est pas nouvelle, et le principe sera d'ailleurs proposé nativement dans la version 5 de PHP. C'est d'ailleurs la lecture des fonctionnalités attendues de cette prochaine version qui a motivé la création de l'objet Environment. Voilà pour rendre à César ce qui appartient à César.

9.1.1.2. Attributs

L'objet Environment possède les attributs suivant :

Attribut	Description
aFormVars	Tableau associatif comprenant toutes les variables postées (fusion des tableaux $_POST, $_GET, et $_FILES)
aObjects	Tableau associatif d'objets transversaux (Session, Article …etc.)
aJavascripts	Tableau listant les fichiers javascripts à inclure dans la page
aStyles	Tableau listant les feuilles de style à inclure dans la page
aConstantes	Tableau de constantes dépendant du site courant (chemins …)
aLog	Tableau de log applicatif
sSid	Site courant (raccourci)
sAct	Action courante (raccourci)
sMod	Module courant (raccourci)
iArt	Identifiant de l'article courant (raccourci)
iRub	Identifiant de la rubrique courante (raccourci)
bPreview	Flag : est-on en mode « preview » ?
bDebug	Flag : est-on en mode « debug » ?
bOnline	Flag : le site est-il ouvert ou pas
bUseCache	Flag : la mise en cache des portlets est-elle activée

Il est conseillé, comme d'habitude, de ne pas manipuler ces méthodes directement, mais d'utiliser les accesseurs.

9.1.1.3. Méthodes

On décrit ici quelques méthodes parmi les plus utilisées. Une lecture du code de l'objet Environment est vivement conseillée en complément.

getFormVar
Cette méthode permet de récupérer une variable d'url ou de formulaire. Dans le code exécuté dans un objet lors de l'appel de la page http://.../site.php?rub=22&art=53, on doit récupérer la rubrique courante en tapant : « `$this->oEnv->getFormVar("rub")` »

getObject
Cette méthode permet de récupérer un objet depuis le tableau aObjects. Par exemple, on utilise un objet Session pour manipuler la session PHP. Cet objet est initialisé et enregistré dans l'environnement en début de page. Pour récupérer cet objet de n'importe quel endroit du code, on doit écrire : « `$oSession = this->oEnv->getObject("session")` », et manipuler la session depuis l'objet obtenu.

logMsg
Cette méthode permet d'enregistrer dans le tableau de log un message de débug. Ce message peut-être d'information, un warning, ou un message d'erreur.
Les logs ne sont affichés en bas de page que si le site est en mode debug.

Etc…

9.1.2. Objet Portlet

Nous avons déjà largement parlé de cet objet dans tout ce qui précède. Il semble toutefois important de préciser deux ou trois choses

9.1.2.1. Abus de langage

Le terme « Portlet » ne fait pas référence à l'API java homonyme. Il s'agit d'un terme utilisé commercialement avec l'outil Interligo proposé par la société SQLI (qui m'employait dans une autre vie). Le site internet de la Communauté Urbaine de Lille s'appuie sur l'outil de gestion de contenu Interligo. Le terme « Portlet » fait donc partie de notre vocabulaire commun : il fait référence à un « bloc affiché dans une page ».

Dans le contexte Tiny, on parlera aussi dans un souci de simplicité de « créer des portlets ». Il s'agit là encore d'un abus de langage, car on créé une classe Feeder et un gabarit (HTML/XML). **On ne créé jamais de classe fille de Portlet**.

9.1.2.2. Mais au fait, comment ça marche ?

A la lecture du code, il peut être surprenant de constater qu'à l'exception du portlet Page, on ne demande jamais à un portlet de s'afficher. On se contente de créer les objets, sans vraiment les utiliser. Ce comportement est lié au fonctionnement du moteur de templates. En début de page on créé une instance du moteur, qu'on enregistre dans l'environnement. Les Portlets et Containers créés récursivement tout au long de l'exécution de la page vont simplement transmettre des données au moteur de template (via la référence qu'ils possèdent en attribut sur l'objet Environment). Les Portlets et Containers n'écrivent donc rien directement : ils fournissent des templates au moteur et lui disent comment les remplir.

Donc quand on écrit « `$content = new Portlet($this->oEnv, ...);` », l'objet reçu (`$content`) importe peu : c'est dans son instanciation que tout se passe.

9.1.3. Objet Container

Par rapport au Portlet, le Container fait peut-être figure de parent pauvre :
- son code n'est pas partageable au sein d'une instance Tiny
- les templates qu'il utilise sont codés en dur

Il serait dommage de rester sur cette idée fausse. Prenons le cas de l'outil d'administration des sites, et en particulier le module de gestion des rubriques. Ce module est assez riche, puisqu'il propose :
- de choisir une rubrique de travail
- le détail de la rubrique sélectionnée
- la gestion des acteurs de la rubrique
- …etc.

Ce module utilise des objets métiers pour gérer l'accès aux données de la base ou du système de fichiers. Mais les règles de gestion et l'affichage de tous les modes d'utilisation du module sont codées dans un même objet : le Container RubManager. C'est excessivement pratique puisque :
- le code est très centralisé
- le code est très lisible (Cf. « back/private/classes/RubManager.class »)
- le même conteneur utilise des templates différents suivant le mode d'affichage demandé

On peut noter en outre qu'un Container est plus facile à coder qu'un Portlet a priori.

Conclusion : un Container c'est bien aussi.

9.1.4. Portlet ou Container ?

Le choix est simple : lorsque l'on aborde un nouveau développement, il convient d'abord de se demander s'il pourra être réutilisé ou non. Si c'est le cas, on créera des Portlets, sinon des Containers.

9.2. Portlets clés

9.2.1. Portlet Page

Ce portlet est important puisque c'est le point d'entrée de la construction de la Page. Voilà un bref aperçu de ses méthodes :

getContent
C'est dans cette méthode qu'est codée la logique d'affichage des pages en fonction du contexte.

getPageJs – getPageCss
Ces méthodes génèrent les inclusions HTML des fichiers javascripts et css pour l'ensemble de la page. Prenons l'exemple du portlet MenuDhtml. Comme son nom l'indique, il sert à générer un menu DHTML. Ce menu s'appuie lui-même sur une feuille de style spécifique et sur un fichier javascript. Lorsque le portlet MenuDhtml est instancié, il enregistre les références de ces fichiers dans l'environnement. Ces enregistrements sont traités par les méthodes « getPageJs » et « getPageCss » pour générer les lignes de HTML adéquates.

getPageTitle
Cette méthode génère le titre de la page en fonction du contexte (« Site – Rubrique – Article »).

updateHits
Cette méthode met à jour les statistiques d'utilisation d'un site.

9.3. Analyse site.php

La structure de la page PHP d'un site est la suivante :

1er temps : initialisations
- mesure du top départ (pour affichage des temps de réponse)
- positionnement de l'error_reporting
- inclusion des fichiers utiles
- création d'un objet Auth (pour repérer l'utilisateur courant)
- création d'un objet Environment
- initialisation de l'objet Environment
- création d'un objet Session et enregistrement dans l'objet Environment
- création du moteur de templates et enregistrement dans l'objet Environment
- enregistrement dans l'objet Environment des feuilles de styles et javascripts communs
- enregistrement dans l'objet Environment de l'objet Auth créé plus haut
- interception des traitements de mise à jour (voir annexe 12.1)

2ème temps : construction de la page
- en une ligne : instanciation d'un portlet Page

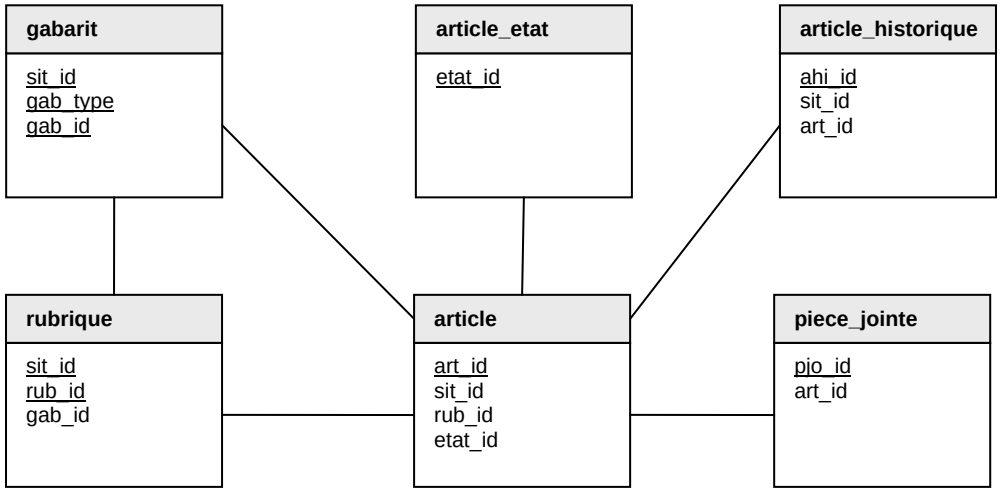
3ème temps : affichage

- mesure du temps de traitement
- affichage de ce qu'a généré le moteur de template
- mesure du temps d'affichage
- affichage des temps, du contexte d'exécution et des logs applicatifs

10. Modèle de données

10.1. Contenu

10.1.1. Modèle



10.1.2. Table « rubrique »

Définition d'une rubrique

Champ	Type	Description
sit_id	varchar(10)	Id civilité
rub_id	int	Id rubrique
gab_id	varchar(30)	Id gabarit
rub_id_parent	int	Id rubrique mère
rub_libelle	varchar(128)	Libellé rubrique
rub_description	varchar(255)	Description de la rubrique
rub_chemin	varchar(255)	Chemin d'accès à la rubrique, à partir de « data/ »
rub_module	varchar(30)	Module associé à la rubrique
rub_niveau	tinyint	Niveau de la rubrique (0 pour accueil, 1 pour niv 1 …etc.)
rub_ordre	tinyint	Ordre d'affichage de la rubrique
rub_hits	int	Nombre de fois où la rubrique a été consultée sur le site
rub_date_creation	varchar(12)	Date de création de la rubrique (format AAAAMMJJHHMM)
rub_user_creation	varchar(30)	Utilisateur qui a créé la rubrique
rub_date_modification	varchar(12)	Date de modification de la rubrique (AAAAMMJJHHMM)
rub_user_modification	varchar(30)	Utilisateur qui a modifié la rubrique

10.1.3. Table « article_etat »

Contient la liste des états définis pour un article (« en rédaction » …).

Champ	Type	Description
etat_id	varchar(10)	Id état
etat_libelle	varchar(50)	Libellé état
etat_description	varchar(255)	Description état
etat_ordre	tinyint	Ordre d'affichage

10.1.4. Table « article »

Définition d'un article

Champ	Type	Description
art_id	int	Id article
rub_id	int	Id rubrique d'appartenance
sit_id	varchar(30)	Id site
gab_id	varchar(30)	Id gabarit
art_chemin	varchar(255)	Chemin vers l'article au format XML (pareil que rub_chemin)
art_filename	varchar(50)	Nom du fichier XML
art_titre	varchar(50)	Titre de l'article
art_masterpage	tinyint	Flag article en masterpage de rubrique (0 : non, 1 : oui)
art_ordre	tinyint	Ordre d'affichage de l'article
art_date_debut	varchar(12)	Date de début de validité (format AAAAMMJJHHMM)
art_date_fin	varchar(12)	Date de fin de validité (format AAAAMMJJHHMM)
etat_id	varchar(10)	Id état
art_hits	int	Nombre de fois où l'article a été affiché sur le site
art_prints	int	Nombre de fois où l'article a été imprimé (non utilisé)
art_sends	int	Nombre de fois où l'article a été transféré (non utilisé)
art_date_creation	varchar(12)	Date de création de l'article (format AAAAMMJJHHMM)
art_user_creation	varchar(30)	Utilisateur qui a créé l'article
art_date_modification	varchar(12)	Date de modification de l'article (AAAAMMJJHHMM)
art_user_modification	varchar(30)	Utilisateur qui a modifié l'article

10.1.5. Table « article_historique »

Liste des civilités d'une personne

Champ	Type	Description
ahi_id	int	Id enregistrement
sit_id	varchar(30)	Id site
art_id	int	Id article
art_titre	varchar(50)	Titre de l'article
per_id	varchar(30)	Id utilisateur
ahi_action	varchar(10)	Code action (create : création, delete : suppression, update : mise à jour, state : changement d'état)
etat_id	varchar(10)	Id état
ahi_date	varchar(12)	Date d'enregistrement de l'action (format AAAAMMJJHHMM)

10.1.6. Table « gabarit »

Définition d'un gabarits (d'article ou de rubrique) pour un site

Champ	Type	Description
sit_id	varchar(10)	Id site
gab_type	char(1)	Type de gabarit (a : article, r : rubrique)
gab_id	varchar(30)	Id gabarit
gab_libelle	varchar(50)	Libellé gabarit
gab_online	tinyint	Flag gabarit utilisable (0 : non, 1 : oui)
gab_ordre	tinyint	Ordre d'affichage du gabarit
gab_date_creation	varchar(12)	Date de création du gabarit (format AAAAMMJJHHMM)
gab_user_creation	varchar(30)	Utilisateur qui a créé le gabarit
gab_date_modification	varchar(12)	Date de modification du gabarit (AAAAMMJJHHMM)
gab_user_modification	varchar(30)	Utilisateur qui a modifié le gabarit

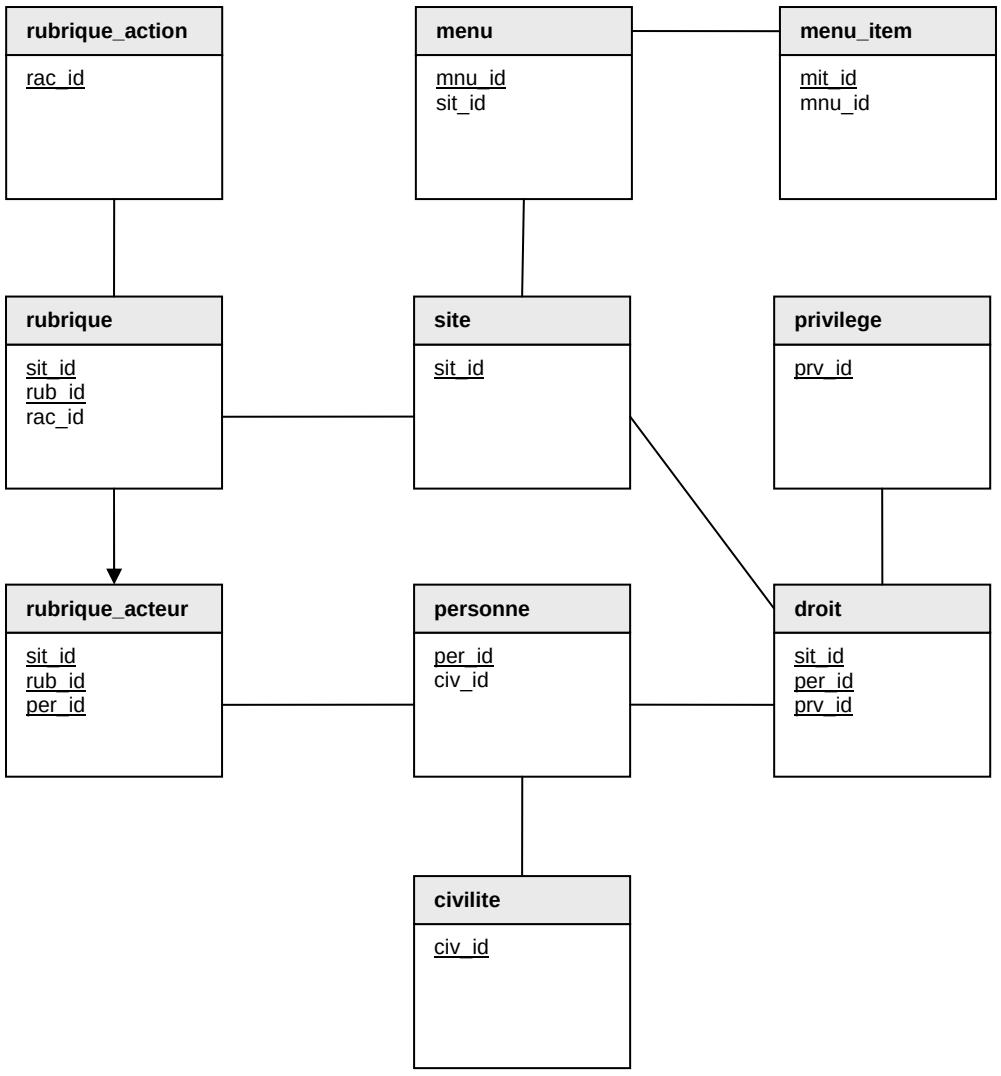
10.1.7. Table « piece_jointe »

Définition d'une pièce jointe d'un article

Champ	Type	Description
pjo_id	int	Id pièce jointe
art_id	int	Id article
pjo_libelle	varchar(50)	Libellé associé (non affiché)
pjo_liblink	varchar(50)	Libellé du lien de téléchargement de la pièce jointe
pjo_downloads	int	Nombre de fois où la PJ a été téléchargée (non-utilisé)
pjo_ordre	tinyint	Ordre d'affichage de la pièce jointe

10.2. Utilisateurs, droits et sites

10.2.1. Modèle



10.2.2. Table « site »

Liste des sites de l'instance

Champ	Type	Description
sit_id	varchar(10)	Id civilité
sit_libelle	varchar(50)	Libellé civilité
sit_description	varchar(255)	Description du site
sit_en_travaux	tinyint	Flag site en travaux (0 : non, 1 : oui)
sit_debug	tinyint	Flag affichage messages debug (0 : non, 1 : oui)
sit_portlet_caching	tinyint	Flag mise en cache des portlets (0 : non, 1 : oui)

10.2.3. Table « privilege »

Liste des privilèges définis dans l'application

Champ	Type	Description
prv_id	tinyint	Id privilège
prv_libelle	varchar(50)	Libellé privilège
prv_description	varchar(255)	Description privilège

10.2.4. Table « personne »

Utilisateurs définis dans l'instance

Champ	Type	Description
per_id	varchar(30)	Id utilisateur (login)
per_pwd	varchar(32)	Mot de passe
civ_id	tinyint	Id civilité
per_nom	varchar(50)	Nom
per_prenom	varchar(50)	Prénom
per_email	varchar(255)	Email

10.2.5. Table « civilite »

Liste des civilités d'une personne

Champ	Type	Description
civ_id	tinyint	Id civilité
civ_libelle	varchar(30)	Libellé civilité

10.2.6. Table « droit »

Liste des privilèges des personnes sur les sites

Champ	Type	Description
sit_id	varchar(10)	Id site
per_id	varchar(30)	Id personne
prv_id	tinyint	Id privilège

10.2.7. Table « menu »

Bloc dans le menu gauche de l'outil d'administration pour un site

Champ	Type	Description
mnu_id	int	Id bloc menu
sit_id	varchar(10)	Id site dans lequel le bloc doit apparaître
mnu_libelle	varchar(50)	Libellé du bloc
mnu_description	varchar(255)	Description du bloc (affichée sous le titre)
mnu_ordre	int	Ordre du bloc

10.2.8. Table « menu_item »

Item d'un bloc dans le menu gauche de l'outil d'administration pour un site. Chaque item est un lien vers un module du back-office. Chaque lien est affiché si l'utilisateur courant a les privilèges suffisants.

Champ	Type	Description
mit_id	int	Id item
mnu_id	int	Id bloc menu
mit_libelle	varchar(50)	Libellé de l'item
mit_module	varchar(50)	Module associé
prv_minimum	tinyint	Privilège minimum pour l'exécution du module
mit_ordre	tinyint	Ordre de l'item

10.2.9. Table « rubrique_acteur »

Liste des acteurs d'une rubrique

Champ	Type	Description
sit_id	varchar(10)	Id site
rub_id	int	Id rubrique
per_id	varchar(30)	Id personne

10.2.10. Table « rubrique_action »

Liste des actions effectuables sur les rubriques et des privilèges requis

Champ	Type	Description
rac_id	int	Id action
rac_libelle	varchar(50)	Libellé action
rac_description	varchar(255)	Description action
prv_minimum	tinyint	Privilège minimum pour pouvoir exécuter l'action

10.3. Cache

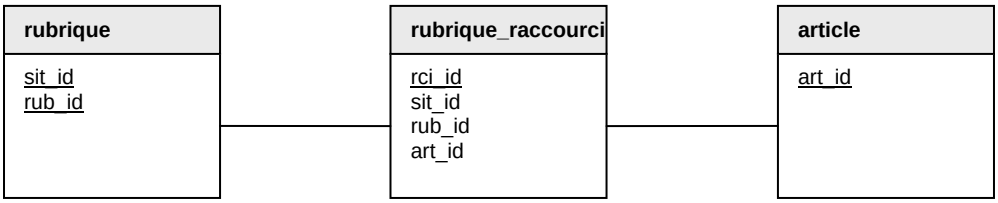
Le chapitre 7 est consacré à la gestion du cache des portlets. Le modèle et les tables (« cache » et « cache_portlet ») y sont explicités.

10.4. Module « Raccourci vers un article »

10.4.1. Rappel du principe

Il s'agit de pouvoir accrocher à une rubrique une liste d'articles du même site, en vue de les afficher en version réduite accompagnée d'un lien vers la version complète (dans une autre rubrique en général).

10.4.2. Modèle



10.4.3. Table « rubrique_raccourci »

Définition d'un raccourci vers un article pour une rubrique

Champ	Type	Description
rci_id	int	Identifiant du raccourci
sit_id	varchar(10)	Id site
rub_id	int	Id rubrique
art_id	int	Id article
rci_ordre	tinyint	Ordre d'affichage du raccourci

10.5. Module « Blocs contextuels »

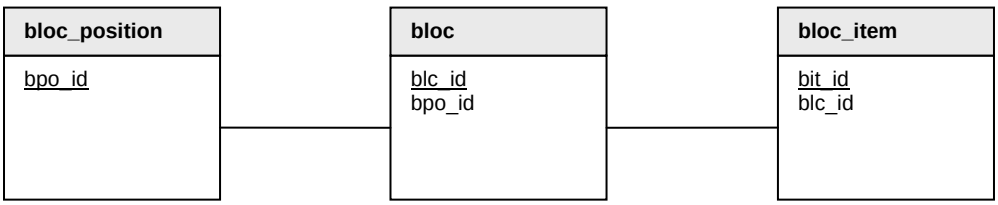
10.5.1. Rappel du principe

Il s'agit de pouvoir accrocher à une rubrique un ensemble d'informations contextuelles. Ces informations sont présentées sous la forme de blocs, un bloc étant constituée de tout ou partie des éléments suivants :

- icône
- titre
- texte descriptif
- liste de liens

A chaque rubrique on peut associer plusieurs blocs. Chaque bloc peut être affiché à gauche, à droite ou au centre de l'écran.

10.5.2. Modèle



10.5.3. Table « bloc_position »

Liste des positions de blocs définies

Champ	Type	Description
bpo_id	char(1)	Identifiant position
bpo_libelle	varchar(30)	Id site

10.5.4. Table « bloc »

Définition d'un bloc

Champ	Type	Description
blc_id	int	Identifiant bloc
sit_id	varchar(10)	Id site
rub_id	int	Id rubrique
blc_libelle	varchar(50)	Titre du bloc
blc_description	text	Texte descriptif du bloc
blc_icone	varchar(50)	Icône du bloc (nom du fichier)
bpo_id	char(1)	Id position
blc_ordre	tinyint	Ordre d'affichage du bloc

10.5.5. Table « bloc_item »

Définition d'un item (lien) à l'intérieur d'un bloc

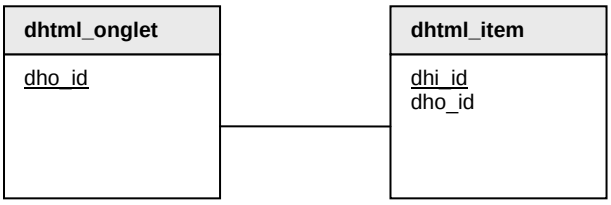
Champ	Type	Description
bit_id	int	Identifiant item
blc_id	int	Identifiant bloc
bit_libelle	varchar(50)	Libellé du lien
bit_url	varchar(255)	Href du lien
bit_flag_fichier	tinyint	Flag lien vers un fichier (0 : non, 1 : oui)
bit_flag_popup	tinyint	Flag ouvrir le lien dans une popup (0 : non, 1 : oui)
bit_popup_width	smallint	Largeur de la popup à ouvrir
bit_popup_height	smallint	Hauteur de la popup à ouvrir
Bit_ordre	tinyint	Ordre d'affichage du bloc

10.6. Module « Menu DHTML »

10.6.1. Rappel du principe

Ce module permet de créer un menu DHTML sur un site. Un menu DHTML est constitué d'onglets. A chaque onglet sont associés des éléments (liens) qui apparaissent dans un menu déroulant.
Pour des raisons de souplesse, le menu DHTML n'est pas synchronisé sur le rubriquage.

10.6.2. Modèle



10.6.3. Table « dhtml_onglet »

Définition d'un onglet du menu

Champ	Type	Description
dho_id	Int	Identifiant onglet
sit_id	varchar(10)	Id site auquel l'onglet appartient
dho_destsit	varchar(10)	Id site de la rubrique de destination (lien)
dho_destrub	int	Id rubrique de destination (lien)
dho_extlink	varchar(255)	Url de destination (si la destination n'est pas une rubrique)
dho_imgname	varchar(50)	Nom de l'image (*)
dho_width	smallint	Largeur de l'image en pixels (*)
dho_height	smallint	Hauteur de l'image en pixels (*)
dho_imgalt	varchar(50)	Tag alt de l'image (*)
dho_haschildren	tinyint	Flag l'onglet à des éléments (0 : non, 1 : oui)
dho_menuwidth	smallint	Largeur du menu déroulant en pixels
dho_offsetx	smallint	Décalage latéral en px par rapport au coin haut gauche de l'image
dho_offsety	smallint	Décalage vertical en px par rapport au coin haut gauche de l'image
dho_ordre	tinyint	Ordre d'affichage de l'onglet

10.6.4. Table « dhtml_item »

Définition d'un item (lien) à du menu déroulant

Champ	Type	Description
dhi_id	int	Identifiant item
dho_id	int	Identifiant onglet
dhi_destsit	varchar(10)	Id site de la rubrique de destination (lien)
dhi_destrub	Int	Id rubrique de destination (lien)
dhi_extlink	varchar(255)	Url de destination (si la destination n'est pas une rubrique)
dhi_libelle	varchar(128)	Libellé
dhi_ordre	tinyint	Ordre d'affichage de l'item

10.7. Module « Faq »

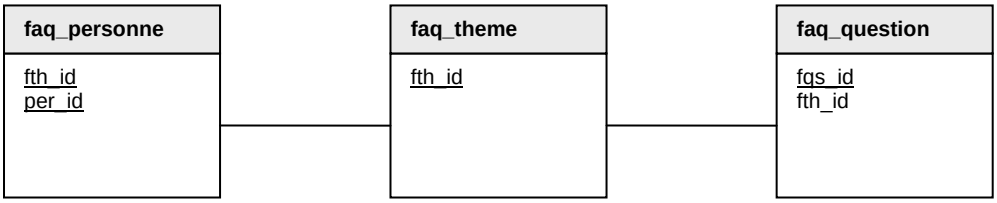
10.7.1. Rappel du principe

Ce module permet de gérer une liste de FAQ :
- une FAQ est une question assortie d'une réponse
- les FAQ sont réparties dans des thèmes.
- à chaque thème est associée une liste de destinataires

Le module FaqFront permet à l'internaute de naviguer dans les thèmes existants, d'afficher la liste des questions de chaque thème. Il permet en outre à l'internaute de poser une

nouvelle question dans un des thèmes : la question est alors adressée à l'ensemble des destinataires du thème.

10.7.2. Modèle



10.7.3. Table « faq_question »

Définition d'une FAQ

Champ	Type	Description
fqs_id	int	Identifiant question
fth_id	int	Identifiant thème
fqs_libresume	varchar(100)	Libellé court de la question
fqs_libelle	varchar(255)	Libellé complet de la question
fqs_reponse	text	Réponse
fqs_ordre	tinyint	Ordre d'affichage de la FAQ
fqs_date_creation	varchar(12)	Date de création (format AAAAMMJJHHMM)
fqs_user_creation	varchar(30)	Créateur
fqs_date_modification	varchar(12)	Date de modification (format AAAAMMJJHHMM)
fqs_user_modification	varchar(30)	Modifieur

10.7.4. Table « faq_theme »

Définition d'un thème de FAQ

Champ	Type	Description
fth_id	int	Identifiant thème
sit_id	varchar(10)	Id site auquel le thème appartient
fth_libelle	varchar(100)	Libellé du thème
fth_ordre	tinyint	Ordre d'affichage du thème

10.7.5. Table « faq_personne »

Table d'association faq_theme / personne

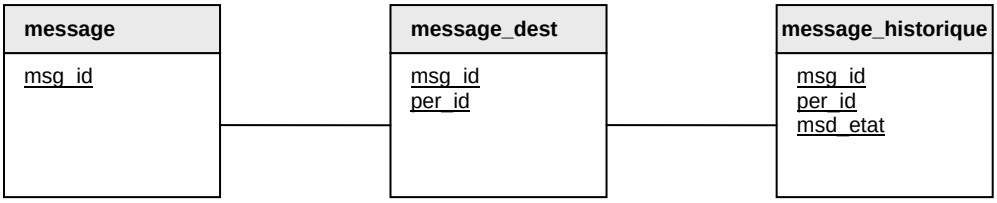
Champ	Type	Description
fth_id	int	Identifiant thème
per_id	varchar(30)	Id personne

10.8. Module « Messagerie »

10.8.1. Rappel du principe

Ce module permet de gérer la mini messagerie utilisée dans l'outil d'administration du site.

10.8.2. Modèle



10.8.3. Table « faq_question »

Définition d'un message

Champ	Type	Description
msg_id	int	Identifiant message
sit_id	varchar(10)	Id site d'origine du message
msg_type	varchar(10)	Type du message : - **workflow** : message généré par le workflow de validation - **fo** : message généré par un utilisateur du frontoffice - **bo** : message généré par un utilisateur du backoffice
msg_etat	char(1)	Etat du message : - **E** : le message a été émis - **R** : le message a été répondu
msg_emetteur	varchar(128)	Emetteur du message (email si depuis le frontoffice, per_id sinon)
msg_objet	varchar(255)	Objet du message
msg_corps	text	Corps du message
msg_reponse	text	Réponse apportée au message
msg_date	varchar(12)	Date d'envoi du message (format AAAAMMDDHHMM)

10.8.4. Table « message_dest »

Définition d'un destinateur du message

Champ	Type	Description
msg_id	int	Identifiant message
per_id	varchar(30)	Id personne
msd_etat	char(1)	Etat du message vis-à-vis du destinataire: - **N** : le destinataire n'a pas lu le message - **L** : le destinataire a lu le message - **R** : le destinataire a répondu au message

10.8.5. Table « message_historique »

Historique des actions effectuées sur un message

Champ	Type	Description
msg_id	int	Identifiant message
per_id	varchar(30)	Id personne
msd_etat	char(1)	Etat du message vis-à-vis du destinataire: - **N** : le destinataire n'a pas lu le message - **L** : le destinataire a lu le message - **R** : le destinataire a répondu au message

msg_date_action	varchar(12)	Date de l'action (format AAAAMMDDHHMM)

10.9. Module « Gestion des tables de référence »

10.9.1. Rappel du principe

Ce module est un mini phpMyAdmin permettant de gérer le contenu de tables simples. La liste des tables prises en compte par ce module est stockée dans la table « table_reference ».

10.9.2. Table « table_reference »

Champ	Type	Description
tbr_id	int	Identifiant table
sit_id	varchar(10)	Id site
tbr_nom	varchar(50)	Libellé associé à la table

10.10. Module « Statistiques »

10.10.1. Rappel du principe

On peut effectuer des statistiques de consultation des rubriques et des articles. La table work_statistique est une table de travail utilisée dans le calcul des statistiques de consultation des rubriques (en cumulé)

10.10.2. Table « work_statistique »

Champ	Type	Description
sit_id	varchar(10)	Id civilité
rub_id	int	Id rubrique
rub_id_parent	int	Id rubrique mère
rub_libelle	varchar(128)	Libellé rubrique
rub_niveau	tinyint	Niveau de la rubrique (0 pour accueil, 1 pour niv 1 …etc.)
rub_hits	int	Nombre de fois où la rubrique a été consultée sur le site
rub_hits_cumul	int	Cumul des hits sous-rubriques

11. Conclusion

11.1. Evolutions prévues

11.1.1. Gestion des utilisateurs

A l'heure actuelle il existe un module permettant de gérer la base des utilisateurs. Ce module ne permet de gérer que la table personne : la suppression d'un utilisateur avec ce module n'impacte donc pas le reste des tables de la base de données.
Un module dédié de gestion des utilisateurs sera donc développé pour pallier à cela.

11.1.2. Gestion des gabarits

Finalement un gabarit est un jeu de fichiers XML et HTML ; le XML est simple à générer, et nous disposons d'un éditeur HTML Wysiwyg. On prévoit donc de créer une interface de gestion de ces gabarits permettant leur création en ligne depuis l'outil d'administration du site. C'est d'ailleurs le fait d'avoir envisagé cette fonctionnalité au départ qui a conduit au choix du moteur de templates PHPLIB : en effet il possède l'avantage de manipuler des templates en HTML pur, sans code propriétaire (au contraire de Modelixe ou Spip par exemple).

11.1.3. Identifiants articles

La clé primaire de la table article est constituée par le champ « art_id ». Il s'agit je pense d'une légère erreur de conception. Si la clé avait été constituée d'un identifiant ET de l'identifiant du site d'appartenance de l'article, le transfert d'un site d'une instance tiny à une autre aurait été grandement facilité. Dans un avenir proche on se permet donc de mettre en œuvre cette évolution …

11.2. Evolutions pas prévues mais intéressantes

11.2.1. Editeur HTML crossbrowser

L'éditeur HTML utilisé dans l'outil d'administration est Spaw (http://www.solmetra.com). Cet éditeur ne fonctionne qu'avec Internet Explorer 5.5 (ou +). Il existe des solutions alternatives, s'appuyant sur des applets Java ou des contrôles ActiveX, qui rendrait possible l'édition HTML sur tous types de navigateurs.

11.2.2. Modification du gestionnaire de cache

Le gestionnaire du cache des portlets est le module PEAR « Cache ». Ce module travaille avec une connexion MySql persistante. Certains hébergeurs n'autorisent pas ce mode de connexion, ce qui pose donc un problème. On pourra donc se pencher sur l'utilisation d'un module de cache ne nécessitant pas une connexion persistante.

11.3. Conclusion générale

D'une façon générale il est assez rare que les gens s'astreignent à lire les conclusions des documentations techniques qui leur tombent entre les mains. De plus ma capacité à conclure ne s'améliore pas avec le temps – fait que je constate avec une amertume non feinte. Je me

contenterai donc ici d'espérer que vous avez pu comprendre, au travers cette fastidieuse lecture, quelques uns des mécanismes qui régissent Tiny.

Pour toute précision complémentaire, vous pouvez vous référer à la documentation technique de Tiny, téléchargeable depuis le site de l'Adullact.

Je voudrais remercier tout particulièrement le lecteur attentif pour sa persévérance et son opiniâtreté : il aura sans doute compris que la méthode exposée dans la phrase précédente avait ses limites, et il pourra donc, lui seul et personne d'autre, adresser ses questions pertinentes à l'adresse mail qui figure à chaque pied de page de ce document.