FRONT OFFICE

Le *front office* est la partie publique du site, celle à laquelle tous les utilisateurs peuvent accéder librement sans contrainte particulière (sans mot de passe). Cette partie du site est fabriquée à partir de la base de données de SPIP, à partir de fichiers-types (formats) en HTML fabriqués par le webmestre du site.

Important **Avant de pouvoir travailler sur le front office, il faut déjà avoir installé une structure dans le back-office, et y avoir installé des articles (que l'on a passé dans le mode «publié en ligne»). En effet, sans ces articles et cette première structure de rubriques, il est impossible pour le front-office de récupérer la moindre information.**

Les fichiers prédéfinis de SPIP

Outre le dossier /ecrire, vous trouverez dans la racine du dossier SPIP plusieurs fichiers,aux noms évocateurs : article.php3, rubrique.php3, article.html, rubrique.html. Ces fichiers permettent de configurer la présentation du front office.

Les fichiers .php3 comportent peu d'informations, mais ils sont indispensables : ce sont eux qui seront «appelés» par les utilisateurs, ce sont ces fichiers qui vont réellement fabriquer le site. Mais ils ne peuvent fonctionner seuls : ils utilisent en effet pour fonctionner des fichiers-types contenant la description des pages présentées aux visiteurs. Ces fichiers-types sont les fichiers en .html.

Remarque **Le choix de la terminaison .html pour les fichiers-types peut sembler paradoxal, ainsi que l'installation de ces fichiers dans la racine du site (au même niveau que les fichiers .php3). Il se justifie par la façon dont les webmestres vont concevoir ces pages: en fabriquant des fichiers HTML, puis en les installant directement à la racine du site. La terminaison se justifie par le fait que ce sont réellement des fichiers HTML qui seront utilisés comme tels pendant la phase de conception, l'installation dans la racine par la nécessité de ne pas compliquer outre mesure la gestion des liens internes (notamment pour les images).**

Ces fichiers-types (ou formats) sont, à la base, de simples fichiers HTML. Mais, afin d'effectuer le lien entre ces formats et la base de données, ils utilisent un méta-langage de description de structure de page. La compréhension de ce méta-langage n'est pas indispensable, mais elle permet de réellement personnaliser son propre site.

«Visiter» le site public

À moins d'une personnalisation, l'appel des pages du site public se fait par des URL sur les fichiers .php3, suivis d'une variable correspondant à l'article ou à la rubrique souhaitée.

Par exemple, pour visiter l'article 5, l'URL est : article.php3?id_article=5

Pour visiter la rubrique 3, l'URL est : rubrique.php3?id_rubrique=3

On verra plus loin comment configurer une navigation avec des adresses plus claires sur certains serveurs [pas encore rédigé].

Adapter graphiquement des formats prédéfinis

Le plus simple, pour le webmestre qui ne souhaite pas entrer dans les subtilités du méta-langage de SPIP, consiste à récupérer des formats prédéfinis et à simplement modifier les images.

Au fur et à mesure, nous mettrons à la disposition des utilisateurs une banque de formats-types, accompagnés à chaque fois de la procédure (la plus simple possible) pour les personnaliser.

Modifier et créer des formats

La méthode la plus efficace pour personnaliser entièrement son site Web passe par l'apprentissage du méta-langage de SPIP. Au départ, on se contentera d'adapter les pages des formats prédéfinis, mais nous verrons qu'il est possible de créer ses propres formats-types *ab initio*.

Pour les trolls **Une autre méthode n'est pas à exclure: vous pouvez si vous en avez les compétences, programmer ab initio votre propre interface de consultation de la base de données, en PHP/mySQL ou par n'importe quelle passerelle. Le back-office (le dossier /ecrire) et le front-office fonctionnent en effet de manière totalement indépendante; vous pouvez donc vous contenter d'utiliser l'interface du back-office pour créer les rubriques et les articles, et programmer une interface de consultation totalement personnelle, sans que cela perturbe le fonctionnement du back-office.**

[À venir] **Vous trouverez en annexe la structure des tables mySQL utilisée par SPIP, qui vous per-mettra de comprendre le fonctionnement du système et de programmer votre propre interface.**

Le fichier .php3

Les fichiers .php3 sont très simples et très courts. En voici un exemple type:

```
<?
$fond="article";
$delais=10;
include ("inc-public.php3");
?>
```

Il n'y a que deux variables sur lesquelles le webmestre peut intervenir.

$fond (ici « article ») indique le nom du fichier (format) qui sera utilisé pour créer la page correspondante: ici, article.html.

Remarque **Prochainement, on pourra personnaliser les pages en fonction des rubriques, sans modification du fichier .php3. Il suffira de créer des formats numérotés: ainsi rubrique60.html sera appelé pour toutes les rubriques contenues (hiérarchiquement) dans la rubrique 60 (idem pour les articles…).**

Remarque **Grâce à cette variable $fond, on peut nommer différement le fichier .php3 et le fichier .html. Par exemple, on peut fabriquer un fichier index.php3 (qui sera donc affiché en racine du site), qui lui-même utilise un format accueil.html ($fond="accueil"), pour éviter de créer un fichier index.html qui entrerait en conflit lors de l'arrivée à la racine du site.**

La variable $delais est la durée en secondes qui sépare deux mises à jour du fichier. En effet, afin de soulager le serveur de bases de données, les fichiers présentés au public sont stockés (dans un dossier /CACHE) pendant une durée qui est fixée par $delais.

Important **Un délais court (10 correspond à 10 secondes) provoque le calcul de des pages fréquemment, c'est-**

à-dire que le serveur sera fortement sollicité (d'expérience, si le site est très visité, cela provoque des plantages). Un délais long (par exemple: 36000 correspond à une mise à jour toutes les heures) sollicite beaucoup moins le serveur.

On jouera donc sur cette valeur: pendant le développement du format, on passera cette valeur à un valeur très faible (10), de façon à voir immédiatement les modifications que l'on porte sur le format. Une fois ce format adopté définitivement, on donnera une valeur beaucoup plus longue (quelques heures par exemple).

Technique

En réalité, les pages ne sont pas forcément recalculées à la fin du délais. Ce sont les visites des utilisateurs qui provoquent, pour chaque page, le calcul. Si deux visites sur une même page sont espacées d'un délais inférieur à cette valeur, alors c'est la version stockée en cache qui est affichée; si ce délais est supérieur, la page est recalculée et stockée.

Ainsi, le serveur est nettement moins sollicité: une page qui serait très peu visitée par les utilisateurs seraient peu recalculée, même si le temps entre ces visites excède la valeur $delais.

Le fichier .html

Le fichier format (à terminaison en .html) est à la base un fichier HTML simple. Il est complété par un méta-langage propre à SPIP permettant de décrire l'interface entre le site public et la base de données. Par exemple, il faut utiliser un code spécifique pour indiquer que l'on veut afficher, à tel endroit, le TITRE de l'article tel qu'il est enregistré dans la base de données.

Conseil

Pour créer ses propres formats, voici donc la marche à suivre la plus simple:

- fabriquez d'abord un fichier HTML «normal» (soit avec un éditeur spécialisé, WYSIWYG ou non, soit «à la main») , contenant tous les éléments prévus sur ce type de page (par exemple, une page d'article, contenant un titre, la date, le nom des auteurs, le texte…), les éléments de navigation dans le site (les rubriques liées, la hiérarchie…); à ce stade, vous pouvez afficher ce fichier HTML (par exemple article.html) directement dans votre navigateur;
- reprenez ce fichier, cette fois-ci dans un éditeur de texte (NotePad, BBedit…), et remplacez les différents éléments par leur équivalent dans le méta-langage de SPIP; à ce stade, vous devez passer par le fichier .php3 correspondant (par exemple article.php3?id_article=1) pour vérifier vos modifications.

Le méta-langage de SPIP fait l'objet d'un chapitre séparé.

LE MÉTA-LANGAGE DU FRONT OFFICE

Comme expliqué précédemment, le méta-langage de SPIP est utilisé dans les fichiers .html et permet de décrire l'interface entre les fichiers HTML (la présentation générale des pages) et les informations tirées de la base de données.

Deux notions différentes y sont utilisées : les boucles et les pseudo-tags.

Les **boucles** ont trois intérêts principaux:

■ indiquer quel type de données on va utiliser (va-t-on utiliser les informations concernant un article, une rubrique, un auteur ou un élément du forum ?) ;

■ permettre l'affichage de listes d'éléments (par exemple: la liste des articles dans une rubrique);

■ imbriquer des listes dans d'autres listes (par exemple: dans la liste des rubriques, afficher une sélection d'articles pour chaque rubrique).

Pour l'instant **À l'heure actuelle, seules les boucles portant sur les articles et les rubriques sont gérées par le système. Les boucles sur les auteurs, les forums et les brèves suivront sous peu.**

Les pseudo-tags indiquent l'emplacement des éléments des articles, des rubriques, des auteurs, des brèves… Par exemple, le pseudo-tag (#TITRE) indique qu'à cet endroit le système doit placer le titre de l'article (ou de la rubrique).

N.B. **Les pseudo-tages indiquant l'emplacement des éléments (titre, date, texte…) ne peuvent être utilisés qu'à l'intérieur de boucles.**

La structure d'un document est donc du type :

```
<BOUCLE1>
    (#TAG pour boucle 1) (#TAG pour boucle 1)
                <BOUCLE 2>
                    (#TAG pour boucle 2)
                </BOUCLE2>
</BOUCLE1>
```

Comme vous pourrez le constater, la seule difficulté ici concerne le principe des boucles. L'utilisation des pseudo-tags est, elle, très simple.

Syntaxe générale des boucles

Une boucle indique, avant tout, de quel type d'information on a besoin : les informations tirées d'un article, d'une rubrique… Cela est indispensable pour, ensuite, pouvoir remplacer les pseudo-tags en fonction du contexte défini par la boucle dans laquelle ils se trouvent.

Une boucle est, de plus, effectuée autant de fois qu'il existe d'éléments correspondant aux critères qu'elle définit. Par exemple: si une boucle cherche un article correspondant à un id_article (l'iden-

tifiant unique de chaque article), le nombre de réponses pour cette boucle sera un (l'article correspondant à cet identifiant), ou zéro (aucun article ne possède cet identifiant). Autre exemple: si un boucle doit afficher la liste des rubriques contenues dans une autre rubrique, on peut avoir zéro réponses (aucune rubrique à l'intérieur de cette rubrique), une réponse (une seule rubrique dans cette rubrique) ou plusieurs réponses (la rubrique contient plusieurs rubriques).

La syntaxe de base d'une boucle est la suivante:

```
<BOUCLEn(TYPE){critère1}{critère2}…{critère x}>
        Code HTML + pseudo-tags correspondant à cette boucle
</BOUCLEn>
```

n est un chiffre correspondant au numéro (que le webmestre choisit arbitrairement) qui sert à différencier les boucles les unes des autres. Nous verrons par la suite l'intérêt des critères.

Par exemple, le codage le plus simple pour un article sélectionné selon son identifiant :

```
<BOUCLE1(ARTICLES){id_article}>
        Code HTML…
</BOUCLE1>
```

Le code HTML se trouvant entre les éléments <BOUCLE*n*> et </BOUCLE*n*> ainsi que les pseudo-tags correspondants aux éléments recherchés (c'est l'intérêt de la boucle) seront interprétés et affichés autant de fois qu'il y a de réponses correspondantes dans la base de données.

En réalité, quelques balises optionnelles permettent d'ajouter plus de souplesse au système.

Le codage d'une boucle avec toutes ces balises est :

```
<B1>
        Code HTML optionnel avant
<BOUCLE1(…){…}>
        Code HTML…
</BOUCLE1>
        Code HTML optionnel après
</B1>
        Code HTML alternatif
<//B1>
```

Notez bien que vous pouvez utiliser les balises <B*n*>, </B*n*> et <//B*n*> indépendamment les unes des autres (soit toutes, soit une ou deux, en fonction des besoins).

■ Le *code optionnel avant* (précédé de <B*n*>) n'est affiché que si la boucle contient au moins une réponse. Il est bien entendu affiché avant les résultat de la boucle.

■ Le *code optionnel après* (terminé par </B*n*>) n'est affiché que si la boucle contient au moins une réponse. Il est bien entendu affiché après les résultat de la boucle.

■ Le *code alternatif* (terminé par <//B*n*>) est affiché à la place de la boucle (et donc des codes optionnels avant et après) si la boucle n'a trouvé aucune réponse.

Par exemple :

```
<B1>
        Cette rubrique contient les articles suivants :
        <UL>
<BOUCLE1(ARTICLES){id_rubrique}>
        <LI>#TITRE
</BOUCLE1>
        </UL>
```

```
</B1>
        Cette rubrique ne contient pas d'articles.
<//B1>
```

affiche la liste des articles contenus dans une rubrique (la mention #TITRE est un pseudo-tag, voir ci-après). Si la boucle comporte au moins un élément (c'est-à-dire la rubrique contient au moins un article), le système affiche le texte « Cette rubrique contient… », affiche le décalage <UL>, affiche la liste des articles précédés d'une puce <LI>, et clôt le décalage </UL>. En revanche, si la boucle ne renvoit aucun élément (la rubrique ne contient pas d'article), Seul le texte « Cette rubrique ne contient pas d'articles » est affiché.

Les types de boucles

Dans la syntaxe d'une boucle :

```
<BOUCLEn(TYPE){critère1}{critère2}…{critère x}>
```

l'élément en majuscules et entre parenthèses (TYPE) est indispensable : il indique au système de quel type de boucle il s'agit. Plusieurs catégories de types de boucles sont disponibles :

■ les boucles portant sur des éléments de la base de données : ARTICLES, RUBRIQUES [en attendant AUTEURS, FORUMS, BREVES…]. Ce sont les plus importantes, car elles permettent de structurer la base même de la navigation. En effet, c'est-à dire-l'intérieur de ces boucles que l'on installe les pseudo-tags ;

■ les boucles récursives : elles permettent d'effectuer des récursivités sur les boucles (par exemple : la liste des rubriques à l'intérieur de rubriques à l'intérieur de rubriques à l'intérieur de…). Leur maniement est relativement délicat ;

■ les boucles spécifiques : pour l'instant, le type HIERARCHIE permet de travailler sur la succession des rubriques qui mènent de la racine du site jusqu'à la rubrique actuelle.

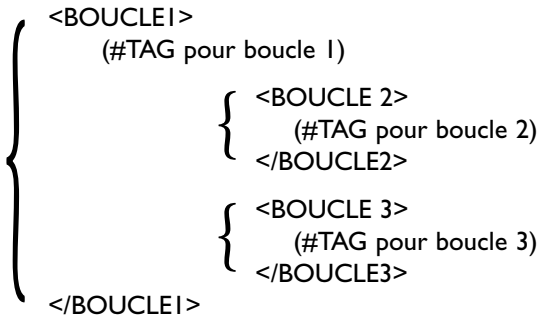
Structure des boucles entre elles

On peut structure différentes boucles dans une même page de deux façons :

■ les unes à la suite des autres ; dans ce cas, les éléments des boucles qui se succèdent sont indépendants les uns des autres (puisque chaque boucle « commence » quand une autre boucle se « termine ») ;

■ les unes dans les autres ; dans ce cas, les éléments de la boucle incluse est héritée de la boucle qui la contient.

Par exemple :



Les boucles 2 et 3 se succèdent, la boucle 3 n'hérite donc pas des données de la boucle 2. Au contraire, les boucles 2 et 3 sont à l'intérieur de la boucle 1 : elles fonctionnent donc en fonction des éléments de la boucle 1. De plus, si la boucle 1 contient plusieurs réponses (par exemple, c'est une liste de plusieurs articles), alors les boucles 2 et 3 seront effectuées autant de fois qu'il y a de réponses dans la boucle 1, et à chaque fois avec les nouveaux éléments de chaque réponse.

Notez encore : si la boucle 2 est vide, la boucle 3 s'exécute tout de même (les deux boucles se succèdent, elles sont donc indépendantes). En revanche, les boucles 2 et 3 ne s'effectuent qu'en fonction des résultats de la boucle 1 (s'il y a aucune réponse pour la boucle 1, alors ces deux boucles incluses ne seront pas effectuées).

Les pseudo-tags

Avant d'aller plus avant dans la description des boucles, voyons dès à présent le principe de fonctionnement des pseudo-tags de SPIP. En effet, ils sont directement liés aux boucles qui définissent le contexte de leur utilisation. Nous verrons ensuite, pour chaque type de boucle, les pseudo-tags associés.

Un pseudo-tag est, tout simplement, l'indication d'un élément tiré de la base de données. Prenons par exemple un article dont le titre (stocké dans la base de données) est « Comment utiliser SPIP? ». Dans le fichier format, on va indiquer un pseudo-tag indiquant qu'à tel endroit doit se trouver le titre de l'article :

```
<HTML>
<BODY>
  <BOUCLE1(…){…}>
        <H1>#TITRE</H1>
  </BOUCLE1>
</BODY></HTML>
```

(Notez qu'on se place forcément dans une boucle.) Au moment du calcul de la page correspondant à notre article, l'élément #TITRE est remplacé par le véritable titre de l'article, et la page HTML envoyée à l'utilisateur devient :

```
<HTML>
<BODY>
        <H1>Comment utiliser SPIP?</H1>
</BODY></HTML>
```

(bref, du bon vieux HTML tout simple...).

Un pseudo-tag décrit donc un élément tiré de la base de données : #TITRE est remplacé par le titre (d'un article, d'une rubrique), #DATE est remplacé par la date tirée de la base… Nous donnons ci-après la liste des pseudo-tags utilisables dans chaque type de boucle.

Il existe une syntaxe plus élaborée pour gérer les pseudo-tags, qui permet l'utilisation de texte optionnel et l'utilisation de fonctions PHP :

Blah blah **[***Texte optionnel avant* **(#TAG)** *Texte optionnel aprè*s**]** blah blah…

Le #TAG est affiché entre parenthèses, et du texte avant et après est encadré de crochets.

Si le #TAG est vide (par exemple, pour #SURTITRE, un article qui ne contient pas de surtitre), alors le texte optionnel avant et après n'est pas affiché. Si le #TAG existe dans la base de données, alors le texte optionnel est affiché (en plus de l'élément tiré de la base de données). Cette fonction de texte optionnel est très utile pour obtenir des présentations très propres : par exemple, voici un codage très simple d'un entête d'article (on se trouve, bien entendu, à l'intérieur d'une boucle de type ARTICLES):

```
<CENTER>
#SURTITRE<BR>
#TITRE<BR>
#SOUSTITRE<BR>
#DATE</CENTER>
```

Imaginons qu'il n'existe, pour cet article particulier, ni surtitre ni soustitre. Une fois calculée, cette page donnerait donc :

```
<CENTER>
<BR>
Comment utiliser SPIP ?<BR>
<BR>
12/11/2000</CENTER>
```

On obtient donc deux <BR> parasites, parce que #SURTITRE et #SOUSTITRE ont simplement

été remplacés par des « blancs » en laissant les <BR> qui terminent leurs lignes. On aura donc intérêt à coder avec du texte optionnel :

```
<CENTER>
[(#SURTITRE)<BR>]
[(#TITRE)<BR>]
[(#SOUSTITRE)<BR>]
[(#DATE)]</CENTER>
```

Dans le même cas de figure (pas de #SURTITRE et #SOUSTITRE), le texte calculé serait nettement plus propre :

```
<CENTER>
Comment utiliser SPIP ?<BR>
12/11/2000</CENTER>
```

Enfin, il existe une dernière subtilité très intéressante : la possibilité de faire passer le #TAG par des fonctions PHP. Cela se code par une succession de fonctions séparées par des barres verticales

(pipe):

Blah blah **[***Avant* **(**#TAG|fonction1|fonction2|…|fonctionx**)** *Après***]** blah blah…

Technique **Traduit en PHP, le code ci-dessus correspond à : fonctionx(…(fonction2(fonction1(#TAG)));**

On peut utiliser n'importe quelle fonction de PHP n'utilisant qu'une seule variable (et traitant du texte), ou des fonctions fournies par SPIP :

■ propre ; traduit les raccourcis SPIP et corrige la typographie;

N.B. **À priori, vous n'aurez pas besoin de cette fonction, car elle est appliquée automatiquement aux éléments textuels.**

■ justifier ; transformer les tags HTML <P> en <P align=justify>. De cette façon, un texte passé préalablement par propre est justifié à droite et à gauche ;

■ majuscules ; passer tous les caractères en majuscules (plus puissant que la fonction PHP correspondante).Très pratique pour les titres.

N.B. **D'autres fonctions seront ajoutées au fur et à mesure.**

La boucle ARTICLES

Une boucle article se code en plaçant ARTICLES (avec un « s ») entre parenthèses :

<BOUCLEn(ARTICLES){critères…}>

Les éléments contenus dans une telle boucle sont ceux des articles.

N.B. **Les boucles ARTICLES ne retournent que des articles publiés. Il y a donc forcément moins d'articles présentés dans le front office que dans le back office (qui, lui, affiche les articles en cour de rédaction).**

Les critères

{tout} les articles sont sélectionnés dans l'intégralité du site (dans toutes les rubriques). Très utile notamment pour afficher les articles les plus récents (dans l'intégralité du site) sur la page d'accueil.

{id_article} retourne l'article dont l'identifiant est id_article. Utilisation : dans la principale boucle d'une page présentant un article (article.html par exemple). Comme l'identifiant de chaque article est unique, ce critère ne renvoit qu'une ou zéro réponse.

{id_secteur} retourne les articles dont le secteur a pour identifiant id_secteur (un secteur est une rubrique qui ne dépend d'aucune autre rubrique).

{id_rubrique} retourne les articles de la rubrique dont l'identifiant est id_rubrique.

{id_auteur} retourne les articles correspondant à cet identifiant d'auteur (utile pour indiquer la liste des articles écrits par un auteur).

Les critères précédents ne peuvent pas s'utiliser simultanément. Les critères suivants, en revanche, complètent les critères précédents.

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc après chaque élément de la liste, sauf le dernier).

{par XXX} permet de forcer le classement de la boucle selon l'élément XXX (cet élément correspond à l'un des champs de la base de données). Les éléments de classement réellement intéressants sont cependant peu nombreux : date (la date de publication de l'article) et titre (le titre de l'article). Ce qui donne : {par date} et {par titre}.

{inverse} présente la boucle dans l'ordre inverse (par rapport à l'ordre alphabétique). Utile notamment pour présenter les articles par date, mais en affichant d'abord les plus récents : {par date}{inverse}.

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément. Par exemple, {par date}{inverse}{0,10} affiche les 10 plus récents articles (notez que pour afficher le premier élément, il faut que a = 0). Pour afficher le troisième élément d'une liste : {2,1} (1 élément après l'élément numéro 2).

{exclus} permet de ne pas afficher l'article utilisé pour le calcul de la boucle. Ainsi, lorsqu'on veut afficher la liste des articles contenus dans la même rubrique qu'un autre article, on ne veut certainement pas afficher ce dernier.

#FORMULAIRE_FORUM insère le formulaire complet permettant aux utilisateurs de répondre à cet article (les liens, les champs, le titre sont gérés automatiquement).

Les pseudo-tags acceptés

Puisque cette boucle gère des listes d'articles, il est normal que les pseudo-tags soient ceux correspondant aux champs des articles de la base de données.

#ID_ARTICLE est l'identifiant de l'article. Utile en particulier pour fabriquer les liens hypertexte vers des articles.

#SURTITRE retourne le surtitre.

#TITRE retourne le titre de l'article (tous les articles ont obligatoirement un titre).

#SOUSTITRE retourne le soustitre.

#DESCRIPTIF retourne le descriptif de l'article.

#CHAPO retourne le texte introductif de l'article.

#TEXTE retourne le texte principal de l'article.

#PS retourne le post-scriptum de l'article.

#NOTES retourne les notes de bas de page de l'article. Cet élément ne correspond à aucun élément de la base de données, mais il est indispensable : les notes sont en effet fabriquées par la fonction propre lors de l'analyse des différents éléments.

#DATE retourne la date de publication de l'article.

#ID_RUBRIQUE est l'identifiant de la rubrique dans laquelle se trouve l'article.

#ID_SECTEUR est l'identifiant du secteur de l'article (la rubrique la plus « haute » quand on remonte la hiérarchie depuis l'article).

#LESAUTEURS contient la liste des auteurs de l'article, chaque nom étant souligné d'un lien vers l'adresse email. Ce tag n'est pas indispensable, mais il permet de gagner du temps en évitant de recourir aux boucles AUTEURS.

Exemple

Voici une boucle qui, placée dans la boucle d'une rubrique (c'est-à-dire que l'on connait un id_rubrique), retourne la liste des articles contenus dans cette rubrique, classés par date inverse (les plus récents au début), munis de leurs liens hypertextes :

```
<B2><UL>
<BOUCLE2(ARTICLES){id_rubrique}{par date}{inverse}>
        <LI>[<A HREF="article.php3?id_article=#ID_ARTICLE">(#TITRE)</A>]
        [<BR><SMALL>(#DESCRIPTIF)</SMALL>]
        [<BR>par (#LESAUTEURS)]
</BOUCLE2>
</UL></B2>Il n'y a aucun article dans cette rubrique.<//B2>
```

La boucle RUBRIQUES

Le boucle RUBRIQUES, comme son nom l'indique, retourne une liste de rubriques.

```
<BOUCLEn(RUBRIQUES){critères…}>
```

Les critères

N.B. **Les critères présentés ici vont sans doute changer. Je les trouve généralement assez peu clairs (ils correspondent aux requêtes SQL, mais il est sans doute inutile de conserver cet aspect trop technique). Si les noms de critères changent, ceux indiqués ci-après seront conservés, mais complétés par des critères plus explicites.**

N.B. **Attention : le front office n'affiche que les rubriques actives, c'est-à-dire les rubriques contenant des articles publiés (ainsi que toutes les rubriques permettant d'arriver à cet article). La hiérarchie des rubriques affichée dans le front office est donc forcément moins complète que celle apparaissant dans le back-office.**

{id_rubrique} retourne la rubrique dont l'identifiant est id_rubrique. Utilisation : dans la principale boucle d'une page présentant une rubrique (rubrique.html par exemple). Comme l'identifiant de chaque rubrique est unique, ce critère ne renvoit qu'une ou zéro réponse.

{id_parent} retourne les rubriques qui dépendent la présente rubrique. Permet notamment d'afficher la liste des rubriques contenues dans une rubrique.

{id_secteur} retourne les rubriques dont le secteur a pour identifiant id_secteur.

{id_enfant} retourne la rubrique qui contient la présente rubrique.

{meme_parent} retourne la liste des rubriques dépendant de la même rubrique que la rubrique en cours. Permet d'afficher les rubriques qui se trouvent au même niveau dans la hiérarchie.

Les critères précédents ne peuvent pas s'utiliser simultanément. Les critères suivants, en revanche, complètent les critères précédents.

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc après chaque élément de la liste, sauf le dernier).

{exclus} permet de ne pas sélectionner la rubrique actuelle. Par exemple, lorsque l'on affiche la liste

des rubriques se trouvant au même niveau dans la hiérarchie ({meme_parent}), on exclus la présente rubrique pour éviter une navigation « sur place ».

{par XXX} permet de forcer le classement de la boucle selon l'élément XXX (cet élément correspond à l'un des champs de la base de données). Concernant les rubriques, le seul intérêt est sans doute le classement par titre {par titre}.

{inverse} présente la boucle dans l'ordre inverse (par rapport à l'ordre alphabétique).

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément (même principe que pour les articles).

Les pseudo-tags acceptés

#ID_RUBRIQUE est l'identifiant de la rubrique.

#TITRE retourne le titre de la rubrique (toutes les rubriques ont obligatoirement un titre).

#DESCRIPTIF retourne le descriptif de la rubrique.

#TEXTE retourne le texte principal de l'article.

#NOTES retourne les notes de bas de page de la rubrique. Cet élément ne correspond à aucun élément de la base de données, mais il est indispensable : les notes sont en effet fabriquées par la fonction propre lors de l'analyse des différents éléments.

#ID_SECTEUR est l'identifiant du secteur de la rubrique (la rubrique la plus « haute » quand on remonte la hiérarchie depuis l'article).

#FORMULAIRE_FORUM insère le formulaire complet permettant aux utilisateurs de répondre à cette rubrique (les liens, les champs, le titre sont gérés automatiquement).

La boucle AUTEURS

Le boucle AUTEURS, comme son nom l'indique, retourne une liste d'auteurs.

```
<BOUCLEn(AUTEURS){critères…}>
```

Les critères

{id_auteur} retourne l'auteur dont l'identifiant est id_auteur : donc zéro ou une réponse.

{id_article} retourne les auteurs correspondant à ce numéro d'article.

{tout} retourne la liste de *tous* les auteurs ayant au moins un article publié sur le site.

{inverse} présente la boucle dans l'ordre inverse (par rapport à l'ordre alphabétique).

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément (même principe que pour les articles).

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc après chaque élément de la liste, sauf le dernier).

{par XXX} permet de forcer le classement de la boucle selon l'élément XXX (cet élément correspond à l'un des champs de la base de données). Concernant les rubriques, le seul

intérêt est sans doute le classement par titre {par titre}.

Les pseudo-tags acceptés

#ID_AUTEUR est l'identifiant de l'auteur.

#NOM est le nom de l'auteur.

#EMAIL est l'adresse électronique de l'auteur.

#BIO est la biographie de l'auteur.

#NOM_SITE est le nom du site Web de cet auteur.

#URL_SITE est l'URL de ce site Web.

Exemple

Nous avons vu qu'il existe unpseudo-tag #LESAUTEURS pour les boucles ARTICLES. Ce raccourcis est l'équivalent de la boucle suivante (à l'intérieur d'une boucle ARTICLES) :

```
<B2>par
<BOUCLE2(AUTEURS){id_article}{", "}>
        [<A HREF="mailto:#EMAIL">(#NOM)</A>]</BOUCLE2>
```

La boucle BREVES

Le boucle BREVES, comme son nom l'indique, retourne une liste de brèves.

```
<BOUCLEn(BREVES){critères…}>
```

Les critères

{id_breve} retourne la brève dont l'identifiant est id_breve.

{id_rubrique} retourne les brèves correspondant à cette rubrique. Attention : les brèves ne peuvent être attachés qu'à une tête de secteur (rubrique rattachée directement à l'entrée du site) ; ainsi on préférera utiliser le critère id_secteur.

{id_secteur} retourne les brèves correspondant à la rubrique dont l'identifiant est id_secteur (entrée dans le site).

{tout} retourne la liste de toutes les brèves.

{inverse} présente la boucle dans l'ordre inverse (par rapport à l'ordre alphabétique).

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément (même principe que pour les articles).

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc après chaque élément de la liste, sauf le dernier).

{par XXX} permet de forcer le classement de la boucle selon l'élément XXX (cet élément corres-

pond à l'un des champs de la base de données). Concernant les rubriques, le seul intérêt est sans doute le classement par titre {par titre}.

Les pseudo-tags acceptés

#ID_BREVE est l'identifiant de la brève.

#ID_RUBRIQUE est l'identifiant de la rubrique à laquelle la brève est attachée.

#ID_SECTEUR est l'identifiant du secteur correspond à cette brève. Inutile : c'est un reste d'une ancienne version de SPIP.

#DATE est la date de publication.

#TITRE est le titre de la brève.

#TEXTE est le texte de la brève.

#INTRODUCTION correspond aux 200 premiers caractères du #TEXTE.

#NOM_SITE est le nom du site Web signalé par la brève.

#URL_SITE est l'URL de du site Web signalé par la brève.

#FORMULAIRE_FORUM insère le formulaire complet permettant aux utilisateurs de répondre à cette brève (les liens, les champs, le titre sont gérés automatiquement).

La boucle FORUMS

Le boucle FORUMS retourne une liste de message de forums.

```
<BOUCLEn(FORUMS){critères…}>
```

Les critères

{if_forum} retourne le message correspondant à cet identifiant.

{id_rubrique} retourne les messages correspondant à cette rubrique.

{id_article} retourne les messages correspondant à cet article.

{id_breve} retourne les messages correspondant à cette brève.

{id_parent} retourne les messages dépendant d'un autre message, dont l'identifiant est id_rubrique. Indispensable pour gérer des *threads* dans les forums.

{id_secteur} retourne les messages correspondant au secteur. À priori, peu utile ; mais cela permet par exemple de faire un forum thématique regroupant tous les messages d'un secteur, quelque soit l'endroit où l'on se trouve.

{tout} retourne la liste de toutes les brèves.

{inverse} présente la boucle dans l'ordre inverse (par rapport à l'ordre alphabétique).

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément (même principe que pour les articles).

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc après chaque élément de la liste, sauf le dernier).

{par XXX} permet de forcer le classement de la boucle selon l'élément XXX (cet élément correspond à l'un des champs de la base de données).

Les pseudo-tags acceptés

#ID_FORUM est l'identifiant du forum.

#ID_BREVE est l'identifiant de la brève à laquelle ce message est attaché (attention, pour l'instant, non récursif : un message répondant à un message lui-même attaché à une brève n'est pas lui-même considéré comme ayant un id_breve.

#ID_RUBRIQUE est l'identifiant de la rubrique à laquelle le message est attaché (même remarque).

#ID_ARTICLE est l'identifiant de la rubrique à laquelle le message est attaché (même remarque).

#DATE est la date de publication.

#TITRE est le titre de la brève.

#TEXTE est le texte de la brève.

#INTRODUCTION correspond aux 500 premiers caractères du #TEXTE.

#NOM_SITE est le nom du site Web signalé par le message.

#URL_SITE est l'URL de du site Web signalé par le message.

#NOM est le nom de l'auteur du message.

#EMAIL est l'adresse mail de l'auteur du message.

#IP contient l'adresse IP de l'auteur du message (à priori, ne sera utilisé que pour des raisons de sécurité, et non d'affichage).

#FORMULAIRE_FORUM insère le formulaire complet permettant aux utilisateurs de répondre à ce message (les liens, les champs, le titre sont gérés automatiquement).

La boucle HIERARCHIE

Cette boucle est très pratique, car elle permet de gérer l'affichage de la hiérarchie qui permet de passer de la racine du site à la rubrique (ou l'article) où l'on se trouve.

```
<BOUCLEn(HIERARCHIE){critères…}>
```

La boucle HIERARCHIE retourne un succession de rubriques.

Les critères

Le nombre de critères est très limité, puisque cette boucle est déterminée par son contexte. Naturellement, puisqu'elle va déterminer la hiérarchie depuis la racine jusqu'à la rubrique dans laquelle on se trouve.

{"*inter*"} indique une chaîne de caractères (*inter*) placée entre chacun des éléments retournés (donc

après chaque élément de la liste, sauf le dernier).

{a,b} (où a et b sont des chiffres) limite l'affichage de la boucle aux b éléments qui suivent la a-ème élément. Cela peut sembler paradoxal puisque, tant qu'à afficher la liste des rubriques qui mènent à tel endroit, autant l'afficher en entier (sinon cela n'a pas d'intérêt). Ce critère est surtout utile pour des raisons de présentation graphique : on peut ainsi affecter un traitement graphique aux premiers éléments de la hiérarchie, et un autre aux suivants (pour cela, on effectue donc deux boucles HIERARCHIE successives).

Les pseudo-tags acceptés

Puisque le résultat de cette boucle est une liste de rubriques, les pseudo-tags acceptés sont exactement les mêmes que ceux de la boucles RUBRIQUES.

Les boucles récursives

Les boucles récursives sont sans doutes les plus puissantes mais les plus sensibles à maîtriser.

Si vous n'arrivez pas à la maîtriser, ça n'est pas très grave : on peut déjà créer un très beau site sans les utiliser. Ce serait plutôt la cerise sur la gâteau (ou le meilleur moyen de planter le serveur de son hébergeur…).

L'appel d'une boucle récursive est très simple : il suffit d'indiquer dans le TYPE de la boucle le numéro d'une autre boucle (qui devra précéder la présente boucle) :

```
<BOUCLEn(boucle x)>
</BOUCLEn>
```

La boucle *n* ne contient rien (on indique tout de même la fermeture de cette boucle, même s'il ne faut rien installer entre l'ouverture et la fermeture). Elle renvoit tout simplement au niveau de la boucle *x* (un peu comme si vous aviez simplement « recopié» la déclaration de la boucle *x* à cet endroit. Si la boucle *n* n'est pas incluse dans la boucle *x*, on a le même comportement que si l'on avait copié-collé la boucle *x* au niveau de la boucle *n*. En revanche, si la boucle *n* est à l'intérieur de la boucle *x*,ça devient plus intéressant : la boucle *x* est à nouveau effectuée et, comme elle contient la boucle *n* qui renvoit à son niveau, elle s'effectue de manière récursive jusqu'à épuisement des résultats des boucles (ou du processeur PHP si on a mal calculé son coup).

En cas d'erreur **Le risque, en cas de mauvaise programmation, c'est bien entendu de lancer le programme dans une boucle infernale dont il ne peut pas sortir. Ne vous inquiétez pas : si tel est le cas, le programme s'interrompera de lui-même, soit parce qu'il a dépassé la taille mémoire allouée à votre script, soit au bout d'un certain temps (généralement 30 ou 60 secondes).**

Notez bien : comme on ne place rien à l'intérieur de telles boucles (puisqu'il s'agit simplement de revenir à une boucle précédente qui comporte elle-même ses propres pseudo-tags), on ne définit ici aucune liste de pseudo-tags spécifiques. Pour la même raison, il n'y a aucun critère lié à ce type de boucle.

Exemples

Voici comment afficher très simplement la liste de toutes les rubriques contenues dans une

rubrique, puis les rubriques dans ces rubriques, etc. Manière d'afficher la « plan » du site :

```
<B1><UL>
<BOUCLE1(rubriques){id_parent}>
        [<LI><A HREF="rubrique.php3?id_rubrique=#ID_RUBRIQUE">(#TITRE)</A>]
        <BOUCLE3(boucle1)></BOUCLE3>
</BOUCLE1>
</UL></B1>
```

Reprenons cet exemple et affichons en plus, cette fois-ci, la liste des articles dans chaque rubrique :

```
<B1><UL>
<BOUCLE1(rubriques){id_parent}>
[<LI><A HREF="rubrique.php3?id_rubrique=#ID_RUBRIQUE">(#TITRE)</A>]
        <B2><UL>
        <BOUCLE2(articles){id_rubrique}>
        [<LI>(#TITRE)]
        </BOUCLE2>
        </UL></B2>
        <BOUCLE3(boucle1)></BOUCLE3>
</BOUCLE1>
</UL></B1>
```

Ajoutez un graphisme qui va bien (avec des tableaux de couleur inclus les uns dans les autres par exemple), et vous obtenez en quelques instants un effet très puissant.

Un exemple pratique

Voyons ensemble comment fabriquer un format « imprimer » : ce format permet d'afficher un article avec un minimum de mise en page pour qu'il soit facilement imprimable. Commençons par fabriquer un fichier imprimer.php3 : c'est lui qui sera appelé depuis la page de chaque article. L'URL sera du type : imprimer.php3?id_article=3.

```
<?
$fond="imprimer";
$delais=10;
include ("inc-public.php3");
?>
```

Nous indiquons $fond="imprimer", ce qui signifie que notre format-type sera dans un fichier imprimer.html. Puisque nous allons travailler sur ce format (et effectuer sans doute de multiples essais), nous fixons un délais de rafraîchissement très court (10 secondes). Lorsque nous aurons terminé, nous reviendrons sur ce fichier, et nous fixerons un délais beaucoup plus long (par exemple $delais=36000 pour une mise à jour toutes les 10 heures).

Commençons par faire un fichier imprimer.html en HTML pur (sans notre meta-langage), afin de tester la mise en page. Nous allons donc travailler avec du texte bidon, que nous remplacerons dans un deuxième temps par les commandes nécessaires :

```
<HTML><BODY>
<A HREF="article.php3?id_article=3">Retour à l'article</A>
<BLOCKQUOTE>
        <H3>Le surtitre</H3>
```

```
        <H1>LE TITRE</H1>
        <H3>Le soustitre</H3>
        <H4>10 septembre 2000</H4>
        <P><B>Le chapo</B>
        <P>Le texte
        <P><FONT SIZE=2>Le postscriptum</FONT>
        <P><HR><FONT SIZE=2>Les notes</FONT>
</BLOCKQUOTE>
</BODY></HTML>
```

Rien à redire : c'est une mise en page très simple ! Commençons par placer notre boucle (en ajoutant un message s'il n'existe pas d'article correspondant à cet identifiant — ce qui, en gros, correspond à gérer une sorte d'ERREUR 404 sur ce type de page) :

```
<HTML><BODY>
<BOUCLE1(ARTICLES){id_article}>
        <A HREF="article.php3?id_article=3">Retour à l'article</A>
        <BLOCKQUOTE>
                <H3>Le surtitre</H3>
                <H1>LE TITRE</H1>
                <H3>Le soustitre</H3>
                <H4>10 septembre 2000</H4>
                <P><B>Le chapo</B>
                <P>Le texte
                <P><FONT SIZE=2>Le postscriptum</FONT>
                <P><HR><FONT SIZE=2>Les notes</FONT>
        </BLOCKQUOTE>
</BOUCLE1>
        <H2>Il n'y a pas d'article à cette adresse.</H2>
</B1>
</BODY></HTML>
```

Voilà, il suffit de remplacer les informations « en clair » par les pseudo-tags correspondants, et le tour est joué (notez qu'on passe le titre en majuscules, la date en clair et le texte et le chapo en justification totale) :

```
<HTML><BODY>
<BOUCLE1(ARTICLES){id_article}>
        <A HREF="article.php3?id_article=#ID_ARTICLE">Retour à l'article</A>
        <BLOCKQUOTE>
                [<H3>(#SURTITRE)</H3>]
                [<H1>(#TITRE|majusules)</H1>]
                [<H3>(#SOUSTITRE)</H3>]
                [<H4>(#DATE|affdate)</H4>]
                [<P><B>(#CHAPO|justifier)</B>]
                [<P>(#TEXTE|justifier)]
                [<P><FONT SIZE=2>(#PS)</FONT>]
                [<P><HR><FONT SIZE=2>(#NOTES)</FONT>]
        </BLOCKQUOTE>
</BOUCLE1>
        <H2>Il n'y a pas d'article à cette adresse.</H2>
</B1>
</BODY></HTML>
```