# Documentation SPIP

## Guide des fonctions avancées

# Guide des fonctions avancées

**Gestion avancée des squelettes : variantes, inclusions, etc.**

# 1. Gestion des squelettes

## {ajax} pour les inclure

Décembre 2008 — maj : Février 2009

Le critère `{ajax}` permet de cliquer sur un lien et de ne rafraichir dans la page que la zone qui changerait si on ré-affichait toute la page avec les nouveaux paramètres.

### En pratique

Le principe est de rendre *AJAX* abordable car la complexité est cachée :

J'ai un morceau de page qui contient des liens vers cette même page et ne provoquent de changement que dans le morceau de page considéré :

1. je mets ce morceau de page dans un squelette séparé
2. je marque les liens concernés par la class ajax : `<a href='monlien' class='ajax'>...</a>`
3. j'inclus mon squelette dans la page principale avec `<INCLURE{fond=monskel}{ajax}{env}>` ou `#INCLURE{fond=monskel}{ajax}{env}`

Et c'est tout !

Quelques petites précisions tout de même :

- testez toujours d'abord votre squelette sans l'argument `{ajax}`
- `{ajax}` devrait toujours être accompagné de `{env}` afin d'éviter tout risque d'injection d'urls dans le cache de SPIP.
- par defaut, les liens a contenus dans une classe `pagination` sont aussi ajaxés
- il est possible de cibler d'autres liens en spécifiant le sélecteur jquery `var ajaxbloc_selecteur = 'a.uneautreclasse';`

### En détail

Syntaxe d'appel :

- `<INCLURE{fond=mon_fond}{ajax}{env}>`
- `[(#INCLURE{fond=mon_fond}{ajax}{env})]`

Cet appel inclut le squelette mon_fond en « ajaxant » automatiquement tous les liens ciblés par la variable js ajaxbloc_selecteur.

Un bloc `<div ..></div>` est inséré automatiquement autour du contenu du squelette inclus, pour le mécanisme de gestion de l'ajax.

Par défaut celle-ci cible les '.pagination a,a.ajax'. Autrement dit, il faut avoir dans le code source des squelettes :

- soit la gestion standard de la pagination par SPIP. A condition que le balise `#PAGINATION` soit inclue dans un élément de classe `pagination`. Ex : `<p class="pagination">#PAGINATION</p>`
- soit une classe `ajax` sur les liens (`<a class="ajax" href=...>`)

Le hit ajax relance automatiquement le calcul du squelette inclus seul en restaurant son `#ENV`, et en y ajoutant les paramètres passés dans l'url du lien

Le bloc rechargé passe en opacité 50 % et prend la class `loading` pendant le chargement ce qui permet de le styler à sa convenance.

Il est intéressant de noter que :

- les liens `ajax` sont mis en cache dans le navigateur lorsqu'ils sont cliqués une fois. Le bloc n'est donc chargé qu'une fois, même si l'internaute revient plusieurs fois sur le lien concerné
- certains liens peuvent être pré-chargés à l'avance en leur ajoutant la classe `preload`

## Quelques Exemples

## Utilisation de liens ajax

Soit le squelette `inc-liens.html` Il sera appelée dans le squelette incluant par

```
[(#INCLURE{fond=inc-liens}{ajax}{env})]
```

Et contiendra :

```
<BOUCLE_liste(AUTEURS) >
   <a class="ajax" href="[(#SELF|parametre_url{id_auteur,#ID_AUTEUR})]">
            #NOM #ID_AUTEUR
   </a>
</BOUCLE_liste>
[(#ENV{id_auteur}|oui) #ENV{id_auteur}]
```

## Utilisation de la pagination

Pour la pagination, prenons un exemple tiré de `squelettes-dist/sommaire.html`.

Mettons dans un squelette `inc-recents.html` la boucle qui liste les articles récents :

```
[(#REM) Derniers articles ]
<B_articles_recents>
<div class="menu articles">
 [(#ANCRE_PAGINATION)]
   <h2><:derniers_articles:></h2>
   <ul>
     <BOUCLE_articles_recents(ARTICLES) {par date}{inverse} {pagination 5}>
       <li class="hentry">
            [(#LOGO_ARTICLE_RUBRIQUE|#URL_ARTICLE|image_reduire{150,100})]
            <h3 class="entry-title"><a href="#URL_ARTICLE"
            rel="bookmark">#TITRE</a></h3>
            <small><abbr class="published" [title="(#DATE|date_iso)"]>
            [(#DATE|affdate_jourcourt)]</abbr>[,<:par_auteur:>
            (#LESAUTEURS)]</small>
             [<div class="#EDIT{intro} introduction entry-content">
            (#INTRODUCTION)</div>]
            </li>
            </BOUCLE_articles_recents>
     </ul>
 [<p class="pagination">(#PAGINATION)</p>]
</div>
</B_articles_recents>
```

Il suffit alors de mettre dans sommaire.html, à la place de cette boucle :

```
<INCLURE{fond=inc-recents}{env}{ajax}>
```

## Limites et cas particuliers

Ce mécanisme automatisé d'ajaxisation des blocs repose sur une hypothèse : les liens ajax ne permettent de rafraichir que le bloc le contenant, en insérant la version modifiée au même endroit. Il n'est pas possible de cibler une autre région de la page

Par ailleurs, il est important de noter que le calcul du squelette inclus ne doit reposer exclusivement que sur les paramètres passés en `#ENV`.

En effet, lors du calcul du bloc ajax, celui-ci sera recalculé tout seul, en dehors du squelette appelant. Seules les variables contenues dans `#ENV` seront restaurées.

Ainsi, il n'est pas possible de faire référence à des variables php globales : celles-ci ne seront pas restaurées au moment du calcul du bloc ajax.

Si vous voulez vraiment utiliser des variables globales php, il faut les réinjecter dans le `#ENV` au moment de l'inclusion :

```
<INCLURE{fond=monskel}{ajax}{env}{parametre=#EVAL{$GLOBALS['toto']}}>
```

## Un peu d'histoire

SPIP 1.9.2 avait entamé une tentative (non documentée car insatisfaisante) de considérer la boucle comme un morceau et de permettre une ajaxisation du contenu d'une boucle, sans avoir à faire de découpe du squelette.

Cela s'est avéré insatisfaisant car :

- l’entité boucle n’est pas forcément pertinente -> exemple d’une liste dont les li sont générés par plusieurs boucles
- cela obligeait à recalculer toute la page pour arriver à la boucle car on ne pouvait pas savoir son contexte d’entrée -> le serveur faisait presque un calcul de page complet pour en extraire le morceau, d’où un gain faible en rapidité

La nouvelle solution à l’avantage :

- d’obliger le concepteur du squelette à faire la découpe du morceau qui sera rafraichi en le mettant dans un squelette inclus
- de lui permettre de décider exactement ce qui forme un bloc fonctionnel indépendant du reste de la page
- d’expliciter son environnement en le passant en paramètre dans le inclure
- de permettre au serveur de ne recalculer effectivement **que** ce morceau de page

## La gestion des pages 404

Février 2007 — maj : Août 2009

Il est possible de créer facilement un squelette pour la page d'erreurs 404 qui sera affichée si l'internaute demande une page qui n'existe pas.

### Page d'erreur 404 , quesaco ?

Lorsqu'une personne navigue dans le Web, il peut lui arriver d'appeler une page web qui n'existe pas. Lorsqu'un serveur web reçoit la demande pour cette page, il répond alors par une page web spécifique (pour signaler que la page demandée n'existe pas), définie par le propriétaire du site (ou par défaut celle fournie avec le serveur), et qui contient généralement un message d'explication, et est accompagnée d'un code de réponse « 404 » du protocole http (qui, seul, ne serait pas très explicite), d'où le nom de « page 404 » parfois donné à ce genre de page.

### Et SPIP dans tout ça ?

Il peut également arriver qu'un utilisateur demande un fichier qui existe, mais qui, faute d'information, n'apporte pas de contenu. Pour prendre un exemple avec SPIP, si quelqu'un tape dans son navigateur un adresse comme comme http://adresse-d-un-site-spip/spip.php?article520, et que l'article 520 n'existe pas encore — ou n'est pas publié en ligne —, il serait logique que SPIP retourne un message d'erreur de la même façon que la procédure précédemment décrite. On pourrait qualifier cela de « pseudo erreur 404 ».

C'est effectivement ce que fait SPIP. Il retourne une page 404 construite à partir d'un squelette. Pour la personnaliser, il suffit donc de créer son propre fichier 404.html, comme n'importe quel squelette SPIP. Ce fichier 404.html sera donc enregistré avec les autres squelettes que vous avez créés (voir à ce sujet l'article « Où placer les fichiers de squelettes ? »).

### Réglage de SPIP

SPIP ne peut pas savoir tout seul quand retourner cette « pseudo erreur 404 », il faut donc lui expliciter dans chacun des squelettes qui pourrait la générer.

Par convention, SPIP va retourner la page d'erreur 404 seulement quand le contenu retourné par le squelette est complètement vide.

Le principe est assez simple, par exemple :

- le squelette de recherche, s'il ne trouve pas de résultats, affichera la page de recherche, probablement avec un message informant l'utilisateur de l'échec de la recherche. Mais il ne retournera pas une « pseudo erreur 404 ».
- par contre, un squelette d'article, si on lui demande un article qui n'existe pas ne doit pas afficher de page article, mais la « pseudo erreur 404 ».

Pour reprendre l'exemple de l'article 520 non publié ou inexistant et de l'appel d'une l'url http://adresse-d-un-site-spip/spip.php?article520, il faudrait concevoir le squelette article de la manière suivante :

```
<BOUCLE_principale(ARTICLES) {id_article}>
code html, y compris entete
</BOUCLE_principale>
```

La boucle _principale ne retourne rien si l'on demande un article inexistant ou non publié, le résultat produit sera alors une page vide, et SPIP engendrera un message d'erreur et retournera une « pseudo erreur 404 ».

## Réglage du serveur Web

Cette procédure est nécessaire pour les pages d'erreurs 404 « normales », c'est à dire dues à l'absence du fichier demandé par l'utilisateur.

La méthode la plus simple est sans doute de se servir du fichier .htaccess fournit par SPIP, même si vous ne vous servez pas des URLs propres. Pour cela, renommez le fichier htaccess.txt livré en standard en .htaccess. [1]. Et c'est tout !

Attention ! il peut arriver que votre hébergeur désactive l'usage du fichier .htaccess, auquel cas il faudra le contacter pour résoudre ce problème.

## Produire une page réellement vide

Il peut arriver de vouloir retourner une page vide sans provoquer une "pseudo erreur 404". Pour cela, il suffit de mettre :

```
#HTTP_HEADER{content-type:text/html}
```

## Notes

[1] il est possible que vous ne puissiez pas appeler ainsi votre fichier sur votre ordinateur. Auquel cas nommer le htacces.txt, et renommer le .htaccess lorsque vous l'aurez mis en ligne.

## Utiliser les modèles

Août 2006 — maj : avril 2010

**Qu'est-ce qu'un modèle ?** Un *modèle* est un petit squelette SPIP qui décrit un fragment de HTML facile à insérer dans un autre squelette ou — et c'est là la principale nouveauté — dans le texte d'un article.

Inspiré des modèles de Wikipédia, le système des *modèles* offre de nouvelles capacités aux webmestres et aux rédacteurs.

Les modèles sont une extension des classiques raccourcis `<img1>` et `<doc1>`. Ceux-ci correspondent désormais aux modèles `dist/modeles/img.html` et `dist/modeles/doc.html` (depuis SPIP 2.0, les modèles *de base* sont rangés dans `prive/modeles/`).

Leur syntaxe a été étendue, et il est possible de spécifier, outre l'alignement (`<img1|left>`, `<img1|right>` ou `<img1|center>`), une classe plus générique, qui soit correspond à un squelette précis, s'il existe, soit à une classe CSS (`<img1|classe>` ira chercher le modele `modeles/img_classe.html`, si ce fichier existe, sinon utilisera le modèle `modeles/img.html`, mais avec un paramètre `class="classe"`).

Mais les modèles ne se limitent pas aux images et documents ; il est ainsi possible de créer de nouveaux raccourcis de la forme `<modele1>`, simplement en ajoutant un squelette dans le répertoire `modeles/` de son dossier de squelettes !

On prendra soin de nommer son fichier en **lettres minuscules** (« bas de casse »). Pour éviter les différenciations sur certains systèmes serveur entre `le_fichier01.html` et `Le_Fichier01.html`, SPIP ne recherchera **que** des fichiers nommés en minuscules.

En l'absence de tout modèle correspondant au raccourci indiqué (par exemple, `<breve1>`), le gestionnaire de modèle de SPIP regarde s'il connaît l'objet demandé (ici, `breve`), et si ce dernier a une URL.

Dans ce cas (vérifié ici, car les brèves sont connues du système et disposent d'une fonction d'URL), SPIP remplace le raccourci `<breve1>` par un petit encadré flottant breve 1, avec un lien vers la brève, et l'affichage de son titre, comme si l'on avait indiqué `[->breve1]`.

Si l'objet n'est pas connu, SPIP laisse le raccourci intact, pour qu'il soit éventuellement traité par la suite (par un plugin ou un filtre supplémentaire), ou tout simplement ignoré.

Il est par ailleurs possible de passer des paramètres supplémentaires aux modèles (comme on le faisait pour les documents flash « embed », avec le raccourci `<emb1|autostart=true>`). La syntaxe en est généralisée et accepte même du HTML, comme dans l'exemple :

```
<son19|couleur=#ff0000|legende=Le grand <i>Count Basie</i>|photo=12>
```

qui pourrait appeler un modèle `modeles/son.html`, lequel exploiterait les paramètres pour afficher la photo numéro 12, dans un cadre de couleur #ff0000, avec une légende comportant des mots en italique.

## Des usages multiples

Nous sommes loin d'avoir exploré toutes les applications possibles des modèles. En voici quelques unes auxquelles nous avons pensé. Si vous en trouvez d'autres, n'hésitez pas à la partager en proposant vos modèles sur SPIP Zone/SPIP Contrib', qui disposent désormais d'une rubrique dédiée.

- **Changer l'aspect des raccourcis de documents.** Souvent demandée, cette fonctionnalité était jusqu'ici difficile à mettre en place, puisqu'il fallait éditer du code php dans les fichiers du noyau de SPIP. Désormais, il suffit de recopier `dist/modeles/img.html` dans un sous-répertoire `modeles/` de son répertoire de squelettes, et de modifier ce fichier. Idem pour les raccourcis `<doc1>` et `<emb1>`, bien que ce dernier soit d'une complexité qui peut rebuter... *Attention :* ne vous lancez pas dans la modification des modèles pour des modifications mineures du rendu de ces raccourcis — il est souvent plus facile de modifier les styles spip_documents_xx à partir des fichiers CSS de votre site.

- **Jouer un son avec un player flash.** Un modèle `modeles/son_player.html` pourrait donner un raccourci `<son12|player>`.

- **Associer un site à un article.** En ajoutant dans ses squelettes un modèle `modeles/site_box.html`, on crée immédiatement un nouveau raccourci `<site1|box>`. On écrit alors le modèle (avec des boucles classiques) de manière à lui faire afficher le nom du site, suivi de liens vers les 3 derniers articles syndiqués, dans une boîte flottant sur la droite, et voici une infobox facile à placer dans un article. Un paramètre pourrait indiquer le nombre d'articles à afficher, la présence ou non de résumés, etc.

- **Afficher une image timbrée.** Une fois qu'on sait faire un squelette qui affiche une photo sous forme de timbre-poste (voir http://www.paris-beyrouth.org/Un-site-dument-timbre), il suffit de le nommer `modeles/timbre.html` pour créer le raccourci `<timbre12>`. Avec, pourquoi pas, des paramètres de taille, de couleur, de choix du tampon, etc[1] [1].

Bien entendu on peut imaginer de la même façon des modèles affichant les images en sépia, en version réduite, etc. Gageons que tout cela sera rapidement disponible sous forme de plugins.

- **Créer un article composite.** Supposons qu'on ait besoin d'un article composé du « chapo » de l'article 1 et du texte de l'article 2. Rien de plus simple : on crée deux modèles, qui renvoient tout simplement, pour l'un, le chapo de l'article demandé, pour l'autre son texte, et dans notre article composite on indique, dans le champ chapo : `<article1|chapo>`, et dans le champ texte : `<article2|texte>`. On peut y ajouter filtres, balises et bidouilles à volonté...

- **Installer un article dans plusieurs rubriques.** En programmant des modèles `<article1|chapo>`, `<article1|texte>` etc, il devient envisageable de mettre une copie d'un article dans un autre article, sans dupliquer les données. On obtient ainsi un article « fantôme » qu'on peut installer dans une nouvelle rubrique (avec, en plus, la possibilité de le titrer différement ou de lui ajouter des éléments).

- **Faire un sondage.** Le plugin *Forms*, qui permet de créer des formulaires et de les exploiter dans des articles avec le raccourci `<form1>`, a été réécrit à partir des modèles.

---

[1] Si l'on préfère que le raccourci s'appelle `<img12|timbre>`, on nommera le modèle `modeles/img_timbre.html`.

- **Afficher une citation aléatoire.** Si on a mis des citations dans ses brèves, un raccourci `<citation|aleatoire>` peut en extraire une au hasard (critère `{par hasard}{0,1}` sur une boucle de brèves), et l'afficher dans un encadré flottant à côté du paragraphe courant.

- **Insérer un document dans une autre langue** que la langue de l'article. Les modèles fonctionnant comme des inclusions, le paramètre `lang=xx` y est toujours disponible. Si l'un de vos documents est bilingue (par exemple avec un « bloc multi » dans le descriptif), vous pouvez afficher le descriptif en espéranto, dans un article en japonais, en appelant `<doc1|left|lang=eo>`. Si le squelette du modèle contient des chaînes de langue, elles seront interprêtées, le cas échéant, dans la langue passée en paramètre.

- **Afficher un graphique.** On passe une table de données au modèle, qui se débrouille ensuite pour créer le graphique et l'insérer dans le flux de texte.

- **Un intertitre sous forme d'image.** Pourquoi ne pas envisager un raccourci `<imagetypo|texte=Mon intertitre>` ?

## Des paramètres à foison

La syntaxe des raccourcis de modèles est de la forme `<modele12>`, `<modele|parametre1=truc|parametre2=chose>` ou `<modele12|alignement|parametre1=etc>`. Les paramètres peuvent être composés de HTML et de raccourcis SPIP (à condition, bien sûr, que les modèles appelés aient prévu de les traiter).

On notera que, pour éviter toute collision avec les balises HTML, un modèle ne peut pas être appelé par un raccourci comme `<modele>`, qui ne contient ni chiffre ni le symbole `|`.

Les paramètres peuvent s'étendre sur plusieurs lignes, ce qui permet l'écriture relativement aérée :

```
<modele 10
 |pays=Allemagne
 |population=82000000
 |superficie=357027
 |classement=63
 |hymne=<i>Das Lied der Deutschen</i>
 |url=http://fr.wikipedia.org/wiki/Allemagne
>
```

Le squelette récupère tous ces paramètres dans la balise `#ENV`, ainsi `#ENV{population}` vaut 82000000. Cette balise étant sécurisée contre toute injection de javascript, si on veut permettre du HTML dans un paramètre, il convient d'utiliser la notation `#ENV*{hymne}` ; si l'on veut en plus appliquer la typographie de SPIP, on peut employer `[(#ENV*{hymne}|typo)]` ou `[(#ENV*{hymne}|propre)]`.

Le paramètre principal (ici, 10), est passé sous deux formes : `#ENV{id}=10`, et `#ENV{id_modele}=10`. Ce qui permet d'accéder, pour un modèle appelé par `<article3|chapo>`, au chapo de l'article 3 en faisant :

```
<BOUCLE_a(ARTICLES){id_article}>#CHAPO</BOUCLE_a>
```

ou bien aux brèves liées au mot-clé numéro 3, par :

```
<BOUCLE_b(BREVES){id_mot=#ENV{id}}>#TITRE</BOUCLE_b>
```

Comme on le voit, chaque modèle devra avoir sa propre documentation, car le raccourci n'indique rien en lui-même sur l'exploitation faite des éléments passés par le raccourci.

## Un emploi possible dans les squelettes

Les modèles ne sont pas limités à des raccourcis dans les articles. Il est possible de les appeler depuis un squelette, en utilisant la balise `#MODELE{modele}` ou `[(#MODELE{modele}{p1=truc,p2=chose}{p3=etc}|filtre...)]` ; cela est toutefois moins nouveau, car c'est équivalent à une inclusion (statique) de squelette (également permise par la nouvelle balise `#INCLURE`).

## Les modèles par défaut

SPIP est livré avec les modèles suivants :

`img`, `doc`, `emb`,

`article_mots` et `article_traductions`, qui donnent respectivement la liste des mots-clés associés à un article, et de ses traductions (raccourcis : `<article1|mots>` et `<article1|traductions>`, mais ces modèles sont aussi appelés par le squelette par `dist/article.html`) ;

`lesauteurs`, qui définit le produit de la balise `#LESAUTEURS`, mais ne peut pas être appelé comme un raccourci ;

et une série de modèles de pagination (voir Le système de pagination).

## Quelques conseils pour écrire un modèle

Il est conseillé de commencer par réfléchir à la syntaxe que l'on veut adopter : ce modèle est-il lié à un article précis ? Si oui, on partira sur un `<article1|xxx>`.

Une fois cette syntaxe établie, on crée le fichier `modeles/article_xxx.html`, et on y entre simplement la balise suivante : `#ENV`. Puis, dans un article de test, on tape notre raccourci `<article1|xxx|param=x...>` ; on voit alors (sous une forme un peu cryptique, il s'agit d'un tableau sérialisé) l'environnement tel qu'il est passé au modèle. On peut par exemple y distinguer notre identifiant d'article (sous le nom `#ENV{id}` et `#ENV{id_article}`).

Ensuite on peut commencer à entrer le squelette de notre fragment de page :

`<BOUCLE_x(ARTICLES){id_article}>` ou

`<BOUCLE_x(ARTICLES){id_article=#ENV{id}}>`. Et c'est parti...

Pour que notre modèle soit complet, il faut essayer les syntaxes `<article1|xxx|left>` et `[<article1|xxx>->www.spip.net]`.

Dans le premier cas, c'est le paramètre `align=left` qui est passé au modèle (et il est souhaitable que le modèle s'aligne alors du côté demandé).

Dans le second cas, le lien est passé dans un paramètre `lien=http://www.spip.net`, et la classe du lien dans `lien_class=spip_out`. Il est recommandé de prendre en compte l'url demandée, en la transformant en lien, quelque part dans le modèle (par exemple sur le titre ou l'icone) ; dans ce cas, il *faut* ajouter dans la première balise HTML du modèle une `class="spip_lien_ok"`, qui signalera au gestionnaire de modèle que le lien a été pris en compte (faute de quoi le gestionnaire ajoutera un `<a href=...>...</a>` autour du modèle produit.

En ce qui concerne les paramètres, la balise `#ENV{x}` a été conçue de manière à éviter toute injection de HTML ou de javascript non désirée. Dans un modèle, on peut souhaiter autoriser le HTML dans les paramètres : il faut alors utiliser `#ENV*{x}` pour récupérer les données, et éventuellement les filtrer par `|propre` ou `|typo`, selon le type de données (texte pouvant comporter plusieurs paragraphes, ou simple ligne de texte).

Sur le plan de la programmation, il est recommandé de concevoir ses modèles tout en boucles, sans aucun code php ni balise dynamique. A noter toutefois : dans un modèle comportant du php, c'est le résultat du calcul qui est mis en cache, et non le script lui-même (comme c'est le cas avec les squelettes).

On lira aussi l'article "Les modèles d'incrustation de documents et leurs filtres".

## Les variantes de squelette

par langue, par rubrique ou par branche

Août 2006 — maj : Octobre 2009

SPIP permet de gérer des variantes de squelettes par rubrique, par branche ou par langue.

Comme mentionné au début du manuel de réference et dans la documentation sur le multilinguisme, SPIP permet de gérer des *variantes des squelettes,* par langue, par rubrique ou les deux.

### Des mises en page différentes

On peut souhaiter, par exemple, que tous les articles d'une rubrique aient une mise en page différente : couleur de fond et taille de texte différentes, informations relatives aux mots clés mises en évidence, etc. Ou encore que le contenu d'une rubrique donnée soit présenté différement parce qu'il correspond à un type de données différent : par exemple en listant tous les articles par numéro, y compris leur contenu qui serait court, plutôt que les derniers en dates suivis d'une pagination de tous les articles, avec des liens vers les pages d'articles. On pourra aussi vouloir que l'interface du site soit différente selon la langue de l'article ou de la rubrique.

Les variantes de squelettes sont une solution simple — mais du coup, pas forcément très souple — pour permettre à SPIP de faire cela.

Pour lui indiquer d'utiliser des mises en pages différentes, il suffit de réaliser des squelettes différents auxquels on donnera des noms de fichier qui indiqueront quand il faut les utiliser :

- `rubrique=22.html` : squelette pour la seule rubrique numéro 22,
- `rubrique-22.html` : squelette pour la rubrique numéro 22 et toutes ses sous-rubriques,
- `article=22.html` : squelette pour tous les articles de la seule rubrique 22,
- `article-22.html` : squelette pour tous les articles de la rubrique 22 ainsi que tous ceux de toutes ses sous-rubriques,
- `article.en.html` : squelette pour les articles en langue anglaise,
- `rubrique.html` et `article.html` : squelettes par défaut, s'appliquant à toutes les rubriques et articles n'ayant pas de squelette particulier, **leur présence est obligatoire dans le même répertoire que les variantes pour que celles-ci soient prises en compte**.

### Ordre exhaustif des variantes de squelette

Prenons l'exemple des squelettes d'article, avec des valeurs données de langue et de rubriques (mais l'explication ci-dessous reste valable pour les squelettes de rubriques ou les brèves, et bien sûr, quelles que soient les langues et les numéros de rubriques).

Rappellons avant tout que SPIP recherche *d'abord* le répertoire où il prendra le squelette, comme détaillé dans « Où placer les fichiers de squelettes ? », en cherchant s'il existe un fichier **`article.html`**. Il ne faut donc pas créer de variante de squelette (par exemple un fichier `article.cs.html`) sans créer *dans le même répertoire* un fichier `article.html`, au

risque de voir notre variante ignorée au profit du squelette générique d'un répertoire de priorité moindre.

Si ce fichier existe, SPIP utilise les fichiers de squelettes selon leur nom en commençant par « les plus précis », avec précédence du rubricage par rapport à la langue. Voici donc l'ordre (par priorité décroissante) dans lequel sont utilisés les fichiers de squelettes selon leur nom :

- **`article=8.cs.html`** : si ce fichier existe, il ne s'applique qu'aux articles en langue tchèque de la rubrique numéro 8 (mais pas aux articles contenus par ses sous-rubriques).

- **`article=8.html`** : si ce fichier existe, il s'applique aux articles de la rubrique 8 (sauf aux articles concernés par un fichier spécifiant aussi la langue, tel que précedemment).

- **`article-2.es.html`** : si ce fichier existe, il s'applique aux articles en espagnol contenus dans la rubrique 2 et ses sous-rubriques. Si la rubrique 8 est une sous-rubrique de la 2 et si les fichiers ci-dessus existent, ils prévaudront.

- **`article-2.html`** : si ce fichier existe, il s'applique aux articles contenus dans la rubrique 2 et ses sous-rubriques (sauf aux articles concernés par les fichiers indiqués ci-dessus).

- **`article.vi.html`** : si ce fichier existe, il s'applique aux articles en vietnamien, dans toutes les rubriques (sauf aux articles concernés par les fichiers indiqués ci-dessus).

- Enfin, le fichier **`article.html`** s'applique à tous les articles qui ne sont pas concernés par les fichiers indiqués ci-dessus. Et, répétons-le, il est nécessaire que ce fichier existe.

## <INCLURE> d’autres squelettes

Août 2002 — maj : Janvier 2009

Lorsque l’on a des éléments de texte et des boucles communs à plusieurs fichiers, on peut vouloir extraire ces éléments des pages où ils se trouvent, les installer dans un fichier séparé, et les appeler depuis les autres squelettes. De cette façon, le code commun est regroupé dans un unique fichier, ce qui facilite notamment les modifications qui concernent plusieurs squelettes d’un seul coup.

On installe ainsi typiquement, dans de tels fichiers séparés appelés depuis de nombreux squelettes, des éléments tels que :

les déclarations d’entête des pages HTML (appel des javascript, des feuilles de style...),

les éléments de navigation communs à la plupart des pages (en haut de page, en colonne de gauche...),

la bannière supérieure (logo, liens vers les crédits du site, la page de contact...),

un pied de page...

### Syntaxe d’inclusion

Les habitués de PHP connaissent la fonction *include*, dont le principe est similaire à ce qui est présenté ici.

On peut appeler un squelette depuis un autre squelette grâce à la balise **<INCLURE>** (on peut aussi utiliser <INCLUDE>, qui est identique). Sa syntaxe générale est :

On ne précise que le nom du squelette à inclure.

Par exemple pour inclure le squelette "pied.html" la syntaxe sera :

```
<INCLURE{fond=pied}>
```

### Inclusions dépendant du contexte

Certaines inclusions peuvent dépendre du contexte. Par exemple, imaginons un squelette « hierarchie », qui affiche le chemin menant à une rubrique depuis la racine du site ; on appelerait cette page par une URL de la forme :

« spip.php?page=hierarchie&id_rubrique=xxx ».

Dans les squelettes voulant afficher la hiérarchie à partir de la rubrique courante, il faut donc indiquer que le paramètre concerné est {id_rubrique} ; si nécessaire, on aura créé une boucle permettant de récupérer le numéro de la rubrique concernée, et on placera le code suivant à l’intérieur de cette boucle :

```
<INCLURE{fond=hierarchie}{id_rubrique}>
```

Note : dans ce cas, le squelette *hierarchie.html* commencera certainement par une boucle rubriques avec le critère {id_rubrique}...

On peut imaginer que, dans certains squelettes, on désire récupérer non pas la hiérarchie en fonction d'une rubrique « variable » (au gré du contexte, par exemple le paramètre passé dans l'URL), mais en fonction d'une rubrique dont on connaît à l'avance le numéro. Pour cela, on peut fixer la valeur du paramètre ainsi :

```
<INCLURE{fond=hierarchie}{id_rubrique=5}>
```

**N.B.** Il est possible d'indiquer plusieurs paramètres dans la balise `<INCLURE>` ; cependant ce cas est très rare en pratique. Evitez d'ajouter des paramètres inutiles, qui rendront le cache moins efficace et votre site plus lent.

**N.B.** Le fichier inclus étant lui-même un squelette, il disposera donc de sa propre valeur de **`$delais`**[2]. Cela peut s'avérer pratique pour séparer des éléments lourds du site, que l'on recalculera peu souvent, et quelques éléments dynamiques nécessitant une mise à jour fréquente (par exemple, syndication).

## Dans un contexte multilingue

Si le multilinguisme de SPIP est activé, il est possible de définir la langue de l'environnement d'un squelette inclus en utilisant le paramètre `{lang}`.

S'il n'y a pas de paramètre de langue utilisé, c'est-à-dire sous la forme `<INCLURE{fond=pied}>`, le squelette inclus est appelé en utilisant la langue par défaut du site ;

`<INCLURE{fond=pied}{lang=es}>` appelle le squelette en espagnol. Bien sûr, vous pouvez remplacer « es » par le code ISO de la langue souhaitée : *en* pour l'anglais, *fr* pour le français, *vi* pour le vietnamien, etc. (voir : « Internationaliser les squelettes ») ;

`<INCLURE{fond=pied}{lang}>` appelle le squelette dans la langue courante du contexte d'inclusion.

Il convient de noter que cela rend possible l'utilisation des codes de fichiers de langue dans les squelettes inclus (voir : « Internationaliser les squelettes »).

Les squelettes inclus supportent les mêmes mécanismes de sélection par langue que les squelettes de « premier niveau ». En d'autres termes les squelettes inclus (ici `pied.html`) peuvent être déterminés par rapport à une certaine langue (`pied.es.html`, par exemple) de la même manière que tout autre squelette. Encore une fois, voir « Internationaliser les squelettes » pour plus de détails.

### `#INCLURE` en statique

La syntaxe `<INCLURE{fond=..}>` provoque l'inclusion des pages à chaque visite d'un internaute, que celle-ci concerne une page déjà en cache ou non.

Avec la nouvelle balise `[(#INCLURE{fond=...})]`, l'inclusion est réalisée lors du calcul du squelette, et son résultat est stocké dans le cache de la page appelante. Avec ce système, on ne peut plus gérer une durée de vie (`$delais` ou `#CACHE{}`) réduite pour un squelette inclus ; en

[2] Rappelons que la variable `$delais` définit la périodicité de mise à jour du cache. Voir la section « Le fichier .php3 » dans « Principe général - version archivée ».

revanche il devient possible d'appliquer des filtres sur le squelette inclus : `[(#INCLURE{fond=lettre}|version_texte)]`.

`<INCLURE{fond=..}>` et `[(#INCLURE{fond=..})]` ont donc des fonctionnements différents :

avec `<INCLURE{fond=..}>`, chaque squelette inclus a un cache indépendant ;

avec `[(#INCLURE{fond=..})]`, la page principale appelante contient, en cache, l'intégralité du code généré, et les fichiers inclus n'ont pas de cache séparé.

### `<INCLURE>` et `{doublons}`

Il existe un critère de boucle `{doublons}`. Or, si on utilise un `<INCLURE>` dans un squelette, les doublons mémorisés ne sont pas transmis au squelette inclus.

Il est possible de contourner ce problème. Il suffit pour cela de rajouter le paramètre `{doublons}>` à l'appel du squelette à inclure.

Par exemple : `<INCLURE{fond=mapage}{doublons}>`.

Notons toutefois que les doublons sélectionnés dans le squelette inclus ne "remontent" pas dans le squelette incluant.

### `<INCLURE>` et ajax

SPIP 2.0 a introduit la possibilité d'utiliser facilement la technologie ajax, qui permet de cliquer sur un lien et de ne rafraichir dans la page que la zone qui changerait si on ré-affichait toute la page avec les nouveaux paramètres.

## Les variables de personnalisation

Août 2002 — maj : Décembre 2008

Certains comportements des pages de votre site peuvent être modifiés au moyen de variables PHP. Ces variables sont normalement définies par SPIP, mais, pour obtenir une personnalisation plus fine du site, le webmestre peut les modifier.

### Où indiquer ces variables ?

Inutile d'entrer dans le code source de SPIP lui-même pour fixer ces variables (ouf !).

- **Pour l'ensemble du site**

Si vous voulez fixer ces variables pour l'intégralité du site, vous pouvez les indiquer comme globales, avec une syntaxe un peu différente, dans un fichier intitulé `mes_fonctions.php`, placé à la racine du site, ou, **de préférence**, dans votre dossier `squelettes/` (cf. Où placer les fichiers de squelettes ?).

Il faudra éventuellement créer ce fichier, et entourer les définitions de vos variables par les marqueurs `<?php` et `?>`, voir les exemples ci-dessous.

- **Pour chaque type de squelette**

Vous pouvez aussi définir ces variables squelette par squelette. Pour cela, il faut les installer *au début* du fichier PHP appelant le squelette (par exemple `article.php3`, `rubrique.php3`...). Elles s'insèrent naturellement à côté des variables obligatoires `$fond` et `$delais`. Voir les exemples.

### Les variables du texte

Ces variables sont utilisées lors du calcul de la mise en page (correction typographique) par SPIP.

- **`$debut_intertitre`** fixe le code HTML inséré en ouverture des intertitres (par le raccourci `{{{`). En standard, sa valeur est :

```
$debut_intertitre = "\n<h3 class=\"spip\">\n";
```

- **`$fin_intertitre`** est le code HTML inséré en fermeture des intertitres (raccourci `}}}`). Sa valeur normale est :

```
$fin_intertitre = "</h3>\n";
```

- **`$ouvre_ref`** est le code d'ouverture des appels des notes de bas de page ; par défaut, c'est une espace insécable et un crochet ouvrant ;

- **`$ferme_ref`** est le code de fermeture des appels des notes de bas de page ; par défaut, c'est un crochet fermant.

- **`$ouvre_note`** est le code d'ouverture de la note de bas de page (telle qu'elle apparaît dans `#NOTES`) ; par défaut, un crochet ouvrant ;

- **$ferme_note** est le code de fermeture des notes de bas de page (un crochet fermant).

Des choix alternatifs pourront être par exemple d'utiliser des parenthèses ; ou, plus joliment, d'ouvrir avec le tag HTML `<sup>`, et de fermer avec `</sup>`.

- Le fichier `puce.gif` et la variable **$puce**. Lorsque vous commencez une nouvelle ligne par un tiret, SPIP le remplace par une petite « puce » graphique. Cette puce est constituée par le fichier `puce.gif` installé dans le dossier `squelettes-dist/` à la racine du site ; vous pouvez modifier ce fichier selon vos besoins. Mais vous pouvez aussi décider de fixer vous-même le choix de la puce, au travers de la variable `$puce`. Par exemple pour indiquer un autre fichier graphique :

```
$puce = "<img src='mapuce.gif' alt='-' align='top' border='0'>";
```

ou par un élément HTML non graphique :

```
$puce = "<li>";
```

A partir de la version 1.9.2, et en cas de mutualisation, on peut aussi mettre dans config/mes_options.php, et non pas dans squelettes/mes_fonctions la variable suivante :

```
$GLOBALS['puce'];
```

- **$nombre_surligne** représente le nombre maximum de fois où un mot recherché sera surligné dans le texte. Par défaut, cette valeur est définie à **4**.

- **$ligne_horizontale** est le code de remplacement du raccourci typographique `----` (quatre tirets) ou `____` (quatre caractères de soulignement) qui permet d'insérer une ligne horizontale dans le texte. Par défaut, c'est le code `<hr class="spip" />`

- **$url_glossaire_externe** est l'adresse utilisée pour le raccourcis automatiques `[?SPIP]` vers un glossaire. Par défaut, le glossaire externe renvoie vers l'encyclopédie libre wikipedia.org.

## Les variables pour les forums publics

Il existe des variables permettant de fixer le comportement des forums publics *avec des mots-clés*.

N.B. : Ces variables ne sont utilisées que lorsque vous créez des forums publics dans lesquels les visiteurs peuvent sélectionner des mots-clés ; leur utilisation est donc extrêmement spécifique (et pas évidente...).

- `$afficher_texte` (« oui »/« non »). Par défaut, les forums publics sont conçus pour permettre aux visiteurs d'entrer le texte de leur message ; mais lorsque l'on propose le choix de mots-clés dans ces forums, on peut décider qu'aucun message n'est utile, seul la sélection des mots-clés importe. Dans ce cas, on pourra indiquer :

```
$afficher_texte = "non";
```

- `$afficher_groupe` permet d'indiquer les différents groupes de mots-clés que l'on souhaite proposer dans tel forum. En effet, tous les forums sur un site ne sont pas forcément

identiques, et si, à certains endroits, on peut vouloir afficher une sélection de tous les groupes de mots-clés (ceux que l'ont a rendu accessibles aux visiteurs depuis l'espace privé), à d'autres endroits, on peut vouloir n'utiliser que certains groupes, voire aucune groupe (pas de sélection de mots-clés du tout).

La variable `$afficher_groupe` est un tableau (*array*), et se construit donc de la façon suivante :

```
$afficher_groupe[] = 3;

$afficher_groupe[] = 5;
```

impose l'affichage *uniquement* des groupes 3 et 5.

```
$afficher_groupe[] = 0;
```

interdit l'utiliser des mots-clés dans ces forums (puisqu'il n'existe pas de groupe de mots-clés numéroté 0).

Si l'on n'indique rien (on ne précise pas `$afficher_groupe`), tous les groupes de mots-clés indiqués, dans l'espace privé, comme « proposés aux visiteurs du site public » sont utilisés.

## Interdire l'affichage des boutons d'admin

Toutes les pages de squelette se voient ajouter des « boutons d'amin » (notamment : « recalculer cette page ») lorsqu'on est administrateur et qu'on a activé le cookie de correspondance. Cette fonctionnalité, très pratique pour gérer le site, peut s'avérer malpratique dans certains cas ; par exemple pour des fichiers XML, que l'on ne veut en aucun cas voir perturbés par de tels ajouts.

La variable `flag_preserver` permet d'interdire ces affichages.

```
$flag_preserver = true;
```

On verra par exemple l'utilisation de cette variable dans `backend.php3`.

## Le dossier des squelettes

Pour mettre les squelettes d'un site dans un dossier particulier (par exemple pour mutualiser les sources de SPIP, ou pour faire des essais de différents jeux de squelettes trouvés sur Internet, etc.) il suffit de l'indiquer par la variable **`$dossier_squelettes`**. Cette variable doit être affectée dans le fichier `mes_options.php` du répertoire `config` :

```
<?php
    $GLOBALS['dossier_squelettes'] = 'design';
?>
```

À partir de ce moment-là, SPIP ira chercher en priorité les squelettes présents dans le dossier `design/` (que vous aurez créé à la racine du site). Si, de plus, vous utilisez `<INCLURE{fond=xxx}>`, SPIP ira chercher le fichier `xxx.html` d'abord dans `design/`, puis, s'il n'y figure pas, à la racine du site.

Les avantages de ce rangement sont évidents : meilleure séparation du code de spip et de la structure du site, possibilité de changer tout un ensemble de squelettes d'un seul coup, etc. L'inconvénient principal est que les liens vers les images ou les CSS, notamment, doivent tenir compte de cet emplacement. Il existe pour gérer le problème automatiquement la balise `#CHEMIN`, décrite dans "les balises propres au site". A noter également que la visualisation des squelettes via un simple navigateur sera également gênée, car le visiteur ne les trouvera pas à la racine du site : ça ne dérange en rien le fonctionnement, mais c'est peu compatible avec l'esprit du logiciel libre.

Cette variable permet en fait d'indiquer un ou plusieurs dossiers où SPIP cherchera les squelettes en priorité :

```
<?php
    $GLOBALS['dossier_squelettes'] = 'mes_skel1:mes_skel2';
?>
```

Ceci ouvre différentes possibilités, comme :

essayer un nouveau jeu de squelettes sans écraser l'ancien,

organiser vos squelettes en une arborescence de sous-répertoires : `mes_skel:mes_skel/rss:mes_skel/formulaires:mes_skel/mailing`,

gérer dynamiquement plusieurs jeux de squelettes grâce à des *plug-ins* comme le switcher,

etc.

## Exemples

Pour modifier des variables uniquement pour un certain type de squelettes (par exemples pour les pages de rubriques), il suffit de les définir dans le fichier d'appel de ces squelettes. Par exemple, pour les rubriques, on peut fixer des valeurs directement dans `rubrique.php3` :

```
<?php
    $fond = "rubrique";
    $delais = 2 * 3600;
    $espace_logos = 20;
    include ("inc-public.php3");
?>
```

Ici, on a modifié la valeur de l'espace autour des logos.

Pour modifier des valeurs de variables pour l'ensemble du site, on peut les définir dans le fichier `mes_fonctions.php`.

Attention, lorsqu'on définit des valeurs dans ce fichier, il faut impérativement utiliser la syntaxe `$GLOBALS['xxx']` pour chacune des variables à personnaliser. Par exemple, pour définir la valeur de `$debut_intertitre`, on utilise la syntaxe :

```
$GLOBALS['debut_intertitre'].
```

L'utilisation de cette syntaxe est imposée par des impératifs de sécurité des sites.

```
<?php
    $GLOBALS['debut_intertitre'] = "<h3 class='mon_style_h3'>";
    $GLOBALS['fin_intertitre'] = "</h3>";

    $GLOBALS['ouvre_ref'] = ' (';
    $GLOBALS['ferme_ref'] = ')';
    $GLOBALS['ouvre_note'] = '(';
    $GLOBALS['ferme_note'] = ') ';

    $GLOBALS['espace_logos'] = 0;
?>
```

Il existe d’autres variables permettant de modifier le comportement de SPIP au niveau technique. Ces variables sont décrites sur le site de documentation technique.

# 2. Multimedia et traitements graphiques

Le traitement automatisé des éléments graphiques est particulièrement innovant dans SPIP.

## Les modèles d'incrustation de documents et leurs filtres

Décembre 2008 — maj : Septembre 2009

Les documents joints à un article sont généralement présentés en dehors de son texte, dans une espèce de portfolio séparé. SPIP permet de mentionner chacun de ces documents en un endroit arbitraire du texte. Cette mention peut se faire soit par un lien soit par une incrustation. Jusqu'à SPIP 1.9.2 inclus, le lien était fourni par le modèle `doc` et l'incrustation par le modèle `emb`. Depuis, l'incrustation est également disponible par autant de modèles que de groupes de type simple dans la *Multipurpose Internet Mail Extension* (soit 5 en MIME 1.0, seule version qu'a jamais connu cette nomenclature). Chacun de ces modèles gère différemment l'incrustation, souvent en appliquant au contenu d'un document un filtre spécifique redéfinissable déduit de son type.

Rappelons (Utiliser les modèles) qu'un modèle est un squelette appelable directement à partir du texte présent dans la base de données, afin d'obtenir un placement précis du fragment HTML décrit par le squelette. Pour le placement des pièces jointes, il en existe essentiellement deux, `doc` (pour référencer le document) et `emb` (pour l'incruster). L'incrustation offre le plus de possibilités.

Le modèle `emb` délègue à présent le travail à cinq modèles nommés par le nom du groupe MIME du document concerné : `text, image, audio, video, application`. Ces modèles sont utilisables directement (par exemple `<audio44|center>` est équivalent à `<emb44|center>` si le document relève du groupe audio) et il est d'ailleurs plus efficace de le faire, le modèle `emb` n'étant plus là qu'en cas de doute sur le groupe du document. L'utilisation directe a aussi l'avantage de forcer l'incrustation selon le modèle choisi, même si le document est officiellement d'un autre groupe. Cela permet de contourner certaines incongruités de la classification MIME : beaucoup de documents purement textuels y sont officiellement une `application`, par exemple le XHTML qui est pourtant un sous-ensemble du HTML, qui appartient au groupe `text`.

### Le modèle `text`

Le modèle `text` incruste le contenu d'un document textuel dans un article, en appliquant au préalable si elle existe une fonction `filtre_`*T* où *T* est un nom déduit du type MIME du document. Cette déduction consiste à remplacer dans le nom du type tous les caractères non alphanumériques par un souligné. Ainsi, pour le type `text/txt` le modèle cherchera s'il existe la fonction `filtre_text_txt`.

La fonction `filtre_text_txt` est fournie d'avance par SPIP, et elle est utilisée par défaut pour les documents du groupe `text`. Elle remplace les chevrons dans le corps du document par les entités HTML correspondantes, et l'entoure par une balise `pre`, ce qui permet de présenter le contenu du document directement.

Est fournie également la fonction `filtre_text_csv`, destinée aux documents `text/csv`. Le RFC4180 étant très diversement suivi par les tableurs, cette fonction compte le nombre de tabulation, de virgule et de points-virgule dans le document et considère que le plus fréquent

des trois est le séparateur utilisé. Elle le remplace par les raccourcis SPIP appropriés, ce qui fournit une table HTML correspondant à celle fournie par le tableur. L'usage des guillemets pour introduire le séparateur lui-même ou un saut de ligne est traité correctement (par une entité HTML et une balise `br` respectivement). Le saut de ligne final optionnel est également bien géré. Si la première ligne n'a que sa première colonne de remplie, elle fournira la balise `caption` de la table. La seconde ligne (ou la première sinon) sera vue comme le nom des colonnes, et sera donc typographiée avec des balises `th`. Le format CSV n'ayant pas prévu d'indication d'encodage, il faut s'assurer que celui du document est le même que celui du site avant d'utiliser ce filtre, SPIP ne pouvant savoir si un ré-encodage est nécessaire.

Enfin, il existe `filtre_text_html`, destinée aux documents `text/html`. Elle incruste le corps du document (donc ce qui est délimité par la balise `body`). Ce corps sera toutefois expurgé de ses scripts, par mesure de sécurité. Les balises `style` éventuellement présentes dans l'en-tête, seront regroupées en une seule au début de l'incrustation. Les feuilles de styles référencées dans l'en-tête par des balises `link` de type `text/css` seront recherchées sur le WEB et leur contenu s'ajoutera à la balise `style` mentionnée ci-dessus. Toutefois cela ne sera possible que pour les feuilles dont l'URL est absolue (autrement dit complète). A cette condition seulement SPIP pourra présenter le document de manière équivalente à l'originale, et sous réserve que les balises `img` présentes dans le document possèdent elles aussi des URLs absolues.

### Le modèle `audio`

Le modèle `audio` permet d'incruster le contenu d'un document audio dans un article, sous la forme d'une balise `object`. Comme le précédent, ce modèle cherche s'il existe une fonction `filtre_`*T* où *T* est le nom du type après substitution des caractères non alpha-numériques par des soulignés. Mais il l'applique alors sur l'ID du document. Le résultat de cette fonction est inséré dans la balise `object`, ce qui permet en particulier de préciser les balises `param` souvent spécifiques au type audio concerné. De plus, les couples passés comme argument du modèle (syntaxe `nom=val`) sont eux aussi convertis en balise `param`.

### Le modèle `image`

Le modèle `image` est une petite extension du raccourci `img`, présent de longue date dans SPIP. Il en diffère seulement en produisant une balise `object` plutôt qu'une balise `img` lorsque cette balise ne gère pas le type MIME du document concerné. Pour de tels types, il se comporte donc comme le modèle `audio`, à cette différence que les couples passés en arguments seront traités comme attributs de la balise `object`. Il est donc à choisir en particulier pour l'incrustation de documents de type SVG, lequel relève officiellement du groupe `application`.

### Le modèle `video`

Le modèle `video` incruste le document en l'entourant d'un panneau de contrôle permettant de la faire défiler.

### Le modèle `application`

Ce modèle correspondant au groupe fourre-tout de la spécification MIME, il n'a pas de description synthétique bien claire. Il évoluera certainement beaucoup avec le temps.

## Utilisation

Comme pour les autres modèles, on écrit par exemple `<text67>` pour appliquer le modèle `text` au document numéro 67, et donc voir son contenu directement dans la page.

Ces modèles sont également implicitement utilisés par le squelette `article` standard de SPIP lorsque l'article a un corps vide et possède un seul document joint. Ce squelette se comporte alors comme si le texte de l'article était réduit à `<embN>`, où `N` est le numéro du document, ce qui a pour effet de présenter le corps du document comme contenu de l'article.

Pour afficher sur un site la mise en page opérée par un tableur, il suffit donc de créer un article et d'y joindre la version `csv` de ce type de document, sans aucune manipulation supplémentaire. De même si l'on veut mettre en ligne une vidéo ou un fichier audio.

On peut aussi facilement passer sous SPIP un site HTML statique en associant chacune de ses pages à un article vide : si les URLs de leurs feuilles de style et de leurs images sont complètes, SPIP les laissera conduire la mise en page.

## Images typographiques

### Titres graphiques avec la police de son choix

Mars 2006 — maj : Mai 2009

*Si GD2 et Freetype sont installés sur votre site* [1], SPIP peut fabriquer, de lui-même, des images à partir de titres (ou tout élément de la base de donnée) en utilisant une police de caractères de votre choix.

La fonction qui réalise cet effet est `image_typo`. La présente page va vous présenter les différentes variables que l'on peut utiliser avec cette fonction.

```
[(#TITRE|image_typo)]
```


Si vous ne précisez aucune variable, `image_typo` va utiliser la police par défaut fournie avec SPIP : **Dustismo,** une police GPL de Dustin Norlander. Elle figure dans le sous-dossier `polices` du dossier `dist` ou `squelettes-dist` pour SPIP 2 (dans les versions précédentes de SPIP, il s'agissait du dossier `ecrire`).

## police

Vous pouvez préciser le nom d'une autre police, que vous aurez pris soin d'installer sur votre site.

```
[(#TITRE|image_typo{police=dustismo_bold.ttf})]
```


(**Dustismo-bold** est également livré avec SPIP.)

En théorie, vous pouvez utiliser de nombreux formats de police : TrueType, PostScript Type 1, OpenType... Selon la configuration de votre site, il est possible que certains formats ne soient pas acceptés.

Les polices doivent être installées dans un sous-dossier `/polices` du dossier contenant vos squelettes.

Si nous installons, par exemple, un fichier TrueType ainsi :

```
polices/stencil.ttf
```

il est possible d'utiliser cette nouvelle police [2].

```
[(#TITRE|image_typo{police=stencil.ttf})]
```

### taille

On peut préciser la taille d'affichage de la police. Cela s'utilise avec la variable `taille`.

```
[(#TITRE|image_typo{police=stencil.ttf,taille=36})]
```


Note : on ne précise pas « 36pt », on indique seulement « 36 », sans indication de l'unité.

### couleur

Cette variable permet d'indiquer la couleur. Par défaut, le rendu est noir. Cette variable est une couleur RVB hexadécimale, toujours de la forme « 3399BB ». Notez : on omet le « # » qui précède habituellement ce type de code couleur.

[(#TITRE|image_typo{police=stencil.ttf,taille=36,couleur=4433bb})]

### largeur

La variable `largeur` permet de fixer la largeur *maximale* de l'image. Notez bien : c'est une valeur maximale ; l'image réelle est « recadrée » automatiquement, ensuite, pour adopter les dimensions du texte réellement composé.

Le premier pavé ci-dessous est composé avec une largeur maximale de 300 pixels, le second avec une largeur de 400 pixels.

```
[(#TITRE|image_typo{police=stencil.ttf,largeur=300})]
```

```
[(#TITRE|image_typo{police=stencil.ttf,largeur=400})]
```

## align

La variable `align` permet de forcer l'alignement de plusieurs lignes de texte (lorsque c'est le cas) à gauche, droite, ou au centre. Exceptionnellement, on utilise ici une syntaxe anglaise, proche de ce qui se fait pour les feuilles de style.

```
[(#TITRE|image_typo{police=stencil.ttf,align=left})]

[(#TITRE|image_typo{police=stencil.ttf,align=center})]

[(#TITRE|image_typo{police=stencil.ttf,align=right})]
```


## hauteur_ligne

`hauteur_ligne` permet de fixer la hauteur entre chaque ligne de texte (dans le cas où l'image comporte plusieurs lignes).

```
[(#TITRE|image_typo{police=stencil.ttf,taille=36,hauteur_ligne=80})]
```

## padding

Certaines polices « dépassent » de leur boîte de rendu, et on obtient un effet désastreux (polices « coupées »). La variable `padding` permet, exceptionnellement, de forcer un espace supplémentaire *autour* du rendu typographique.

```
[(#TITRE|image_typo{police=stencil.ttf,padding=5})]
```

## Filtrer l'image

Le résultat de `image_typo` étant une image, il est tout à fait possible de lui appliquer des filtres d'images. Par exemple, ci-après, on rend l'image semi-transparente, ou on lui applique une texture.

```
[(#TITRE|image_typo{police=stencil.ttf,couleur=aa2244}|image_alpha{60})]
```

```
[(#TITRE|image_typo{police=stencil.ttf,couleur=aa2244}|image_masque{carre-
mur.png})]
```


## P.-S.

**N.B.1**. L'image créée par `image_typo` est au format PNG 24 avec une couche alpha pour réaliser la transparence. Pour forcer Microsoft Explorer à afficher correctement cette transparence, SPIP utilise une classe de feuille de style spécifique, `format_png`, définie dans `spip_style.css` ; celle-ci appelle un « comportement » (*behavior*) rendant l'affichage possible sous MSIE. On a donc, encore une fois, tout intérêt à intégrer le `spip_style.css` standard dans ses propres squelettes, quitte à le surcharger avec ses propres styles.

**N.B.2**. L'affichage de certaines polices (notamment les anglaises et certaines italiques) est problématique. Les techniques de rendu typographique dans GD2 sont, visiblement, encore en développement (nous rencontrons bugs sur bugs de ce côté). Espérons que les fonctions GD2 progresseront rapidement.

**N.B.3**. De l'arabe, du farsi, de l'hébreu ? Depuis SPIP 2.0 c'est possible !

## Notes

[1] GD2 est une extension *graphique* de PHP, qui permet de nombreuses manipulations d'images. Freetype, habituellement installé avec GD2, est l'extension qui insère du texte dans une image à partir d'un fichier de police, Truetype ou Postscript.

[2] Attention : si vous ne protégez pas ce dossier, votre fichier de police sera accessible par le Web. Si vous utilisez des polices commerciales, faites attention à ne pas vous retrouver, ainsi, à diffuser des polices pour lesquelles cela n'est pas autorisé.

## Couleurs automatiques

Mars 2006 — maj : août 2010

SPIP permet d’extraire automatiquement une couleur d’une image, afin de l’appliquer à d’autres éléments d’interface.

Par exemple, à côté du logo d’un article, nous allons afficher le titre du même article *dans une couleur* tirée de ce logo. De cette façon, tout en rendant l’affichage plus varié (d’un article à l’autre, la couleur utilisée change en fonction du logo), le fait que la couleur soit extraite de l’image assure une certaine cohérence graphique.

Cette fonction consistant à récupérer une couleur dans une image est complétée par toute une série de fonctions permettant de manipuler cette couleur, principalement éclaircir et foncer la couleur. La liste de fonctions est longue, de façon à permettre un nombre très important d’effets.

### couleur_extraire

À partir d’une image (logo d’article, logo de rubrique..., mais aussi images de portfolio), on demande à SPIP de tirer une couleur.

```
[(#LOGO_RUBRIQUE||couleur_extraire)]
```

Attention : il ne s’agit pas pour SPIP de déterminer la couleur *dominante* de l’image, mais d’extraire une couleur de l’image. Pour que cette couleur soit réellement « représentative », l’image est réduite à une taille de 20 pixels maximum ; ainsi les différentes couleurs de l’image sont relativement « moyennées ». Cette valeur de 20 pixels est expérimentale : elle est suffisamment basse pour éviter d’extraire une couleur très peu présente dans l’image ; elle est suffisamment élevée pour éviter que la couleur soit systématiquement grisâtre.

Utilisée sans paramètres, la fonction `couleur_extraire` retourne une couleur située légèrement au-dessus du centre de l’image. Il est possible d’indiquer un point préféré pour le sondage, en passant deux valeurs (*x* et *y*) comprises entre 0 et 20 (conseil : entre 1 et 19 pour éviter les effets de marges).

Par exemple :

```
[(#LOGO_RUBRIQUE||couleur_extraire{15,5})]
```

retourne une couleur située en haut à droite du centre de l’image.

Pour bien comprendre le principe, appliquons ce filtre sur un logo de couleur uniforme :


Le résultat est : `ff9200`.

*Notez bien :* les valeurs retournées sont systématiquement en codage RVB hexadécimal, en omettant le « # » qui précède habituellement ces codes. On pensera donc à insérer ce dièse quand on utilise ces valeurs.

On peut, par exemple, appliquer cette couleur au fond d'un pavé :

```
<div style="background-color: #[(#LOGO_RUBRIQUE||couleur_extraire)]; width:
100px; height: 100px;"></div>
```

Appliquons ce filtre à une photographie :

En bas à gauche, la couleur extraire sans paramètres, c'est-à-dire présente un peu au dessus du centre de l'image (marron clair de la pierre du bâtiment). En bas à droite, on force une couleur située en haut à droite de l'image (bleu clair du ciel).

```
[(#LOGO_RUBRIQUE||couleur_extraire)]

[(#LOGO_RUBRIQUE||couleur_extraire{15,5})]
```

L’utilisation de ce filtre est, techniquement, très simple. En revanche, créativité et inventivité seront nécessaires pour l’exploiter...

Voici quelques utilisations :

couleur de texte ;

couleur de fond d’un pavé ;

couleur d’une image typographique ;

modifier la couleur d’une image (`image_sepia`)...

## Modifier la couleur

Une fois la couleur extraite, il est utile de la manipuler, afin de jouer avec différentes variantes de la couleur, tout en respectant la cohérence graphique.

- **couleur_foncer**, **couleur_eclaircir**

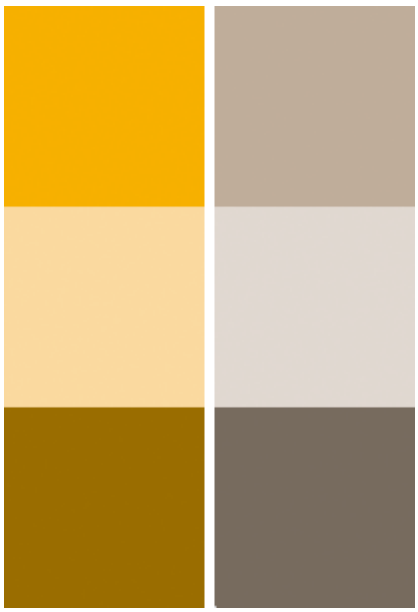
À partir de la couleur extraite d’une image, nous souhaitons afficher des couleurs plus foncées et plus claires.

```
[(#LOGO_RUBRIQUE||couleur_extraire)]

[(#LOGO_RUBRIQUE||couleur_extraire|couleur_foncer)]

[(#LOGO_RUBRIQUE||couleur_extraire|couleur_eclaircir)]
```

Appliqué aux couleurs extraites des exemples précédents, cela donne :



On constate qu’on a ainsi des camaïeux de couleurs faciles à obtenir, l’ensemble étant très cohérent.

- **couleur_foncer_si_claire**, **couleur_eclaircir_si_foncee**

Si nous appliquons la couleur extraite au fond d’un pavé de texte, il faut déterminer dans quelle couleur nous voulons écrire ce texte (par exemple : noir sur orange ou blanc sur orange ?) ; c’est ce que nous verrons avec les fonctions suivantes.

Pour l’instant, nous décidons que le texte sera d’une certaine couleur. Nous voulons par exemple que le texte soit noir. Il faut donc choisir la couleur du fond en fonction de ce texte noir : il faut que la couleur du fond soit claire (donc : texte noir sur fond clair).

Si nous appliquons le filtre `couleur_eclaircir` à notre couleur extraite, nous avons deux cas :

si la couleur est foncée, alors elle est éclaircie et nous obtenons l’effet voulu ;

si la couleur est déjà claire, alors nous l’éclaircissons encore, et nous obtenons un fond qui peut devenir quasiment blanc. Or, la couleur étant déjà claire , nous aurions voulu l’utiliser telle quelle.

C’est ici que nous appliquons le filtre `couleur_eclaircir_si_foncee` :

si la couleur est foncée, nous l’éclaircissons ;

si la couleur est claire, nous l’utilisons telle quelle.

Le filtre `couleur_foncer_si_claire` a la logique exactement inverse. Il est très utile, par exemple, pour écrire en blanc sur un fond systématiquement foncé, mais en évitant de rendre ce fond quasiment noir quand la couleur d’origine est déjà foncée.

- **couleur_extreme**, **couleur_inverser**

Le filtre `couleur_extreme` passe une couleur foncée en noir, et une couleur claire en blanc. Cela est utile pour écrire en noir ou blanc sur un fond coloré.

En réalité, récupérer la couleur « extrême » est habituellement utilisé avec `couleur_inverser`, il inverse la couleur RVB. Elle transforme notamment du noir en blanc, et du blanc en noir.

En pratique, cela permet d’assurer un bon contraste, quelle que soit la couleur du fond du bloc (alors que, dans l’exemple précédent, nous choisissions la couleur du fond du bloc en fonction d’une couleur de texte).

Appliquons, en couleur de fond, la couleur extraite de l’image :

```
<div style="background-color: #[(#LOGO_ARTICLE||couleur_extraire)];"> ...
</div>
```

On obtient donc, selon le logo de l’article, soit un fond foncé, soit un fond clair.

Appliquons, pour la couleur du texte, la couleur extraite, rendue « extrême » :

```
[(#LOGO_ARTICLE||couleur_extraire|couleur_extreme)]
```

Si la couleur est foncée, la couleur extrême est noire ; nous écrivons en noir sur fond foncé.

Si la couleur est claire, la couleur extrême est blanche ; nous écrivons en blanc sur fond clair.

Dans les deux cas, c'est peu lisible. On pourra utiliser cette couleur pour un autre effet (par exemple : une bordure autour du `div`).

Il nous reste à inverser cette couleur pour l'appliquer au texte ;

```
<div style="color:
#[(#LOGO_ARTICLE||couleur_extraire|couleur_extreme|couleur_inverser)];
background-color: #[(#LOGO_ARTICLE||couleur_extraire)];">
                  ...
</div>
```

Si la couleur extraite est foncée, la couleur extrême est noire, et l'inverse est alors blanche. On écrit en blanc sur fond foncé.

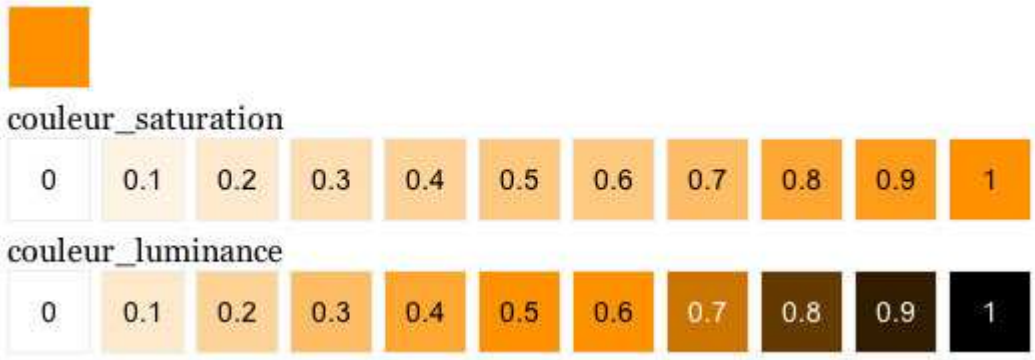
Si la couleur extraite est claire, la couleur extrême est blanche, et l'inverse est alors noire. On écrit en noir sur fond clair.

Dans les deux cas, le contraste assure une bonne lisibilité.

Le filtre `couleur_extreme` admet une variable optionnelle, entre 0 et 1 (par défaut : `0.5`), pour régler le seuil de luminosité à partir duquel on bascule en noir ou en blanc.

- **couleur_saturation** (SPIP 2), **couleur_luminance** (SPIP 2.1)

Ces deux filtres sont destinés à créer des camaïeux de couleurs dont on contrôle totalement la subtilité, et en forçant le résultat sur une échelle absolue (alors que les filtres précédents donnent un résultat relatif à la valeur d'origine).




Le filtre `couleur_saturation` parcourt la saturation de la couleur :

`|couleur_saturation{0}` retourne le blanc,

`|couleur_saturation{1}` retourne la couleur avec sa saturation au maximum (qui n'est pas le noir, sauf si l'on part d'un gris pur),

entre les valeurs 0 et 1, toute la déclinaison de la saturation de cette couleur.

Jouer sur la saturation permet d'obtenir des déclinaisons extrêmement fines à partir d'une couleur, sans introduire dans cette couleur d'effets de « grisé » (un orange de ne vient jamais marron, un vert de devient jamais grisâtre...). En revanche, la valeur maximale est une couleur « pure », dont la luminance est très variable (le bleu porté à la luminance maximale est très sombre ; le jaune porté à sa luminance maximale reste une couleur claire).

Le filtre `couleur_luminance` parcourt la luminance de la couleur :

`|couleur_luminance{0}` retourne le blanc,

`|couleur_luminance{1}` retourne le noir,

`|couleur_luminance{0.5}` retourne une variante de la couleur d'origine avec une luminance moyenne.

Avec cette fonction, on joue sur gamme de 0 à 1 facile à utiliser : on va toujours du 0 au 1 en passant par la couleur d'origine. Le résultat final est donc beaucoup plus prévisible : avec une variable de 0 à 0.5, on obtiendra toujours une couleur claire, et de 0.5 à 1 une couleur foncée.

## Traitement automatisé des images

Mars 2006 — maj : septembre 2010

SPIP permet de faire subir aux images des effets automatisés. Ces effets ont deux vocations :

- tout simplement assurer la cohérence graphique du site, en fabriquant automatiquement des éléments de navigation qui seront toujours réalisés selon les désirs du graphiste ;
- créer des effets relativement spectaculaires, sans pour autant demander aux auteurs des articles de traiter les images eux-mêmes, et sans non plus interdire les évolutions graphiques du site par la suite.

Par exemple : on veut, dans l'interface graphique du site public, que les logos de navigation des articles aient deux aspects :

dans tous les cas, ils sont « posés sur le sol », avec un reflet sous eux ;

au repos, ils sont en noir et blanc, assez foncés ; survolés, ils sont en couleur.

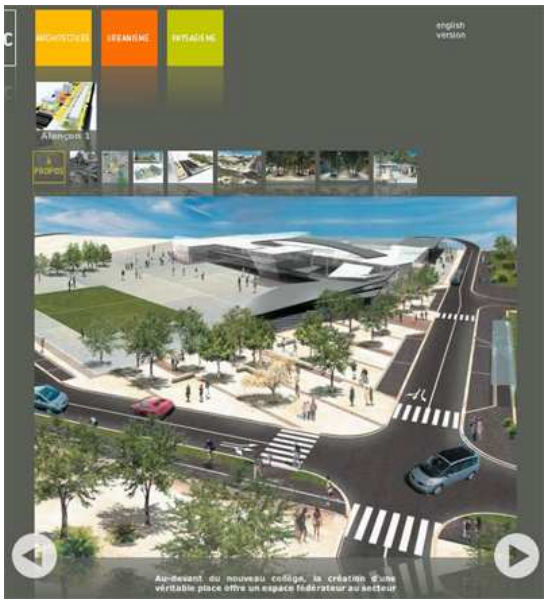
*Sans les automatismes* qui suivent, les webmestres ont pris l'habitude de créer, à la main, deux versions de ces images, et d'installer deux logos sur le site. Deux inconvénients (particulièrement gênants) :

la manipulation est longue ; par ailleurs elle ne peut pas être confiée à un tiers, qui serait de toute façon incapable de la réaliser correctement ;

lorsqu'on voudra faire évoluer l'interface graphique du site, on sera bloqué avec ces logos très typés.

*Avec les automatismes* qui suivent, on peut travailler autrement : les auteurs installent un simple logo d'article (par exemple, une photographie), sans aucun traitement spécifique ; et, automatiquement, les squelettes de SPIP fabriquent :

- une vignette à la bonne taille ;
- la même vignette en noir et blanc, légèrement foncée ;
- les reflets de l'image sur le sol.



Certains des filtres présentés utilisent les fonctions de redimensionnement des images. Il est impératif, après l'installation, de se rendre dans l'espace privé, « Configuration », puis dans l'onglet « Fonctions avancées » : là, sélectionnez « GD2 » pour la « Méthode de fabrication des vignettes ».

## Avertissement lenteur

Avant de commencer, signalons que ces fonctions sont *lourdes*. Voire très lourdes si vous utilisez de grosses images. Le calcul d'une image est relativement long, et s'il faut calculer, dans vos squelettes, plusieurs images, voire plusieurs effets pour chaque image, vous risquez d'obtenir des erreurs de timeout (temps maximum d'exécution des scripts dépassé).

Cela dit, il est important de noter que les filtres de traitement des images ont leur propre système de cache : *une fois calculée, une image « filtrée » est sauvegardée, et ne sera plus recalculée*. La charge sur le serveur est donc limitée au premier calcul.

Cet avertissement concerne, en priorité, les hébergeurs mutualisés.

La page « Vider le cache » de l'espace privé affiche la taille utilisée par SPIP pour stocker ces images calculées. On peut ainsi vider ce cache indépendamment du cache des squelettes et des pages HTML.

## Transparences

Si l'on utilise GD2, outre les fonctions exposées dans cet article, vous constaterez que les réductions d'image (`image_reduire`) respectent la transparence des fichiers GIF et PNG 24 (transparence par couche alpha).

Dans la configuration du site, pensez à sélectionner GD2 comme méthode de réduction d'image.

## L'image d'origine

Toute image gérée par SPIP peut se voir appliquer les filtres suivants. Sont donc concernés les logos (d'articles, de rubriques...), mais aussi les images de portfolio (images en tant que fichiers joints aux articles), sans oublier les nouvelles images typographiques.

Voici, pour nos exemples, une image à traiter.

## Réduire les dimensions d’une image

La fonction `reduire_image` devient `image_reduire`, pour adopter la même logique de noms que les nouvelles fonctions graphiques. L’ancienne dénomination est toujours fonctionnelle.

Elle est désormais complétée par une `image_reduire_par`, qui permet de réduire une image selon une certaine échelle. Là où `image_reduire` réduit une image à des dimensions fixées à l’avance, `image_reduire_par` réduit l’image proportionnellement.

Par exemple :

```
[(#TITRE|image_typo{taille=24}|image_reduire_par{2})]
```

réduit l’image typographique d’un facteur 2 (l’image devient deux fois plus petite).

## Recadrer une image

La fonction `image_recadre{largeur,hauteur,position}` permet de recadrer une image (équivalent du crop des logiciels de traitement d’image) avec les combinaisons de left/center/right et top/center/bottom pour la position (ex ’left center’).

Par exemple :

```
[(#FICHIER|image_recadre{90,90,center})]
```

recadre l’image originale en un carré de 90 px de largeur et hauteur ne gardant en se basant sur le centre de l’image

`image_recadre` permet également d’agrandir un fichier image, le fond est alors spécifié par une couleur ou ’transparent’ en 4eme argument :

```
[(#FICHIER|image_recadre{300,500,'top left','transparent'})]
```

## Supprimer la transparence et forcer le format de l’image

La fonction `image_aplatir` réalise deux opérations :

elle sauvegarde une image dans un format prédéfini (par exemple, transformer une image PNG en une image GIF) ;

elle supprime les informations de transparence, et remplace les zones transparentes par une couleur.

Par exemple :

```
[(#TITRE|image_typo{police=stencil.ttf,couleur=000000,taille=40}
|image_aplatir{gif,ff0000})]
```

Le titre transformé en image typographique est un fichier PNG avec des zones transparentes. En passant cette image par le filtre `image_aplatir`, on la transforme en GIF, en remplaçant les zones transparentes par du rouge (`ff0000`).

La fonction accepte en paramètres, dans l'ordre : le format, la couleur de fond, mais aussi la qualité de compression (32, 64, 128) et l'opacité (0 ou 1). Par exemple : `|image_aplatir{gif,ffffff,128,1}` convertira l'image en gif de bonne qualité en forçant la transparence du fond.

## image_nb

Le filtre `image_nb` passe une image en niveaux de gris (ce qu'on appelle « noir et blanc » lorsqu'on évoque des photographies).


Sans paramètres (image de gauche), le filtre calcule les niveaux de gris en pondérant les composantes de l'image d'origine ainsi :

luminosité = 0,299 × rouge + 0,587 × vert + 0,114 × bleu.

On peut forcer la pondération des trois composantes RVB en passant les valeurs en pour-mille. Par exemple (image de droite) :

```
[(#FICHIER|image_nb{330,330,330})]
```

On a pris chaque composante R, V et B à niveau égal.

## image_sepia

Le filtre `image_sepia` applique un filtre « Sépia ». Appliqué à une photographie, ce genre d'effet donne une tonalité de vieille photographie.

Sans paramètres (image de gauche), la valeur sépia est, par défaut, « 896f5e » (en RVB hexadécimal). On peut passer la valeur de la couleur de sépia en paramètre. Par exemple (image de droite) :

```
[(#FICHIER|image_sepia{ff0033})]
```

## image_gamma

Le filtre `image_gamma` change la luminosité d'une image. Il rend une image plus claire ou plus foncée. Son paramètre est compris entre -254 et 254. Les valeurs supérieures à zéro rendent l'image plus claire (254 rend toute l'image entièrement blanche) ; les valeurs négatives rendent l'image plus foncée (-254 rend l'image complètement noire).

```
[(#FICHIER|image_gamma{70})]
[(#FICHIER|image_gamma{150})]
[(#FICHIER|image_gamma{254})]
[(#FICHIER|image_gamma{-70})]
[(#FICHIER|image_gamma{-150})]
[(#FICHIER|image_gamma{-254})]
```

## image_alpha

Le filtre `image_alpha` rend l'image semi-transparente, en PNG 24 avec couche alpha. Si l'image était déjà semi-transparente, les deux informations sont mélangées.


```
[(#FICHIER|image_alpha{50})]
```

```
[(#FICHIER|image_alpha{90})]
```

Le paramètre est une valeur entre 0 et 127 : 0 laisse l'image inchangée (aucune transparence), 127 rend l'image complètement transparente.

## image_flou

Le filtre `image_flou` rend l'image... floue. On peut lui passer en paramètre un nombre compris entre 1 et 11, définissant l'intensité du floutage (de 1 pixel de floutage à 11 pixels de floutage).

```
[(#FICHIER|image_flou)]
```

```
[(#FICHIER|image_flou{6})]
```

Sans paramètre, la valeur de floutage est 3.


**Attention** : ce filtre est particulièrement lourd (c'est-à-dire nécessite beaucoup de puissance). Plutôt que de tenter un floutage important, on peut préférer flouter plusieurs fois avec des valeurs faibles.

Par exemple, remplacer :

```
[(#FICHIER|image_flou{6})] par
```

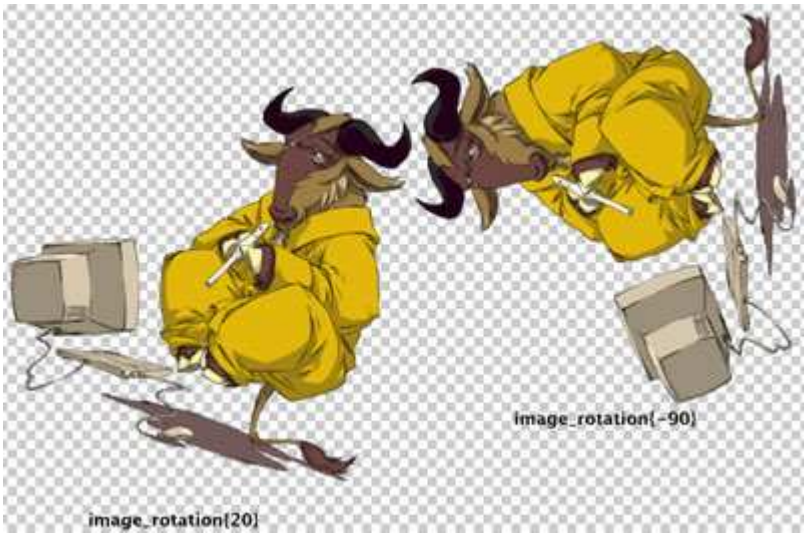
```
[(#FICHIER|image_flou|image_flou)]
```

Au pire, le calcul se fera en deux « recalcul » de squelette, le premier floutage étant sauvegardé en cache.

**Attention (2)** : ce filtre agrandit l'image, en ajoutant tout autour de l'image une « marge » équivalente à la valeur de floutage. Ainsi, avec le paramètre « 3 » (par défaut), on ajoute 3 pixels de chaque côté de l'image, et le résultat aura donc 6 pixels de large et de haut de plus que l'image d'origine.

## image_rotation

Le filtre `image_rotation` fait tourner l'image d'un angle égal au paramètre passé. Les valeurs positives sont dans le sens des aiguilles d'une montre.




```
[(#FICHIER|image_rotation{20})]

[(#FICHIER|image_rotation{-90})]
```

Sauf pour les rotations à angle droit, la rotation provoque un effet d'escalier. Nous avons tenté de le limiter, mais il reste toujours présent. Une solution pour réduire cet effet consiste à réduire l'image après avoir appliqué la rotation.

**Attention** : ce filtre est relativement lourd !

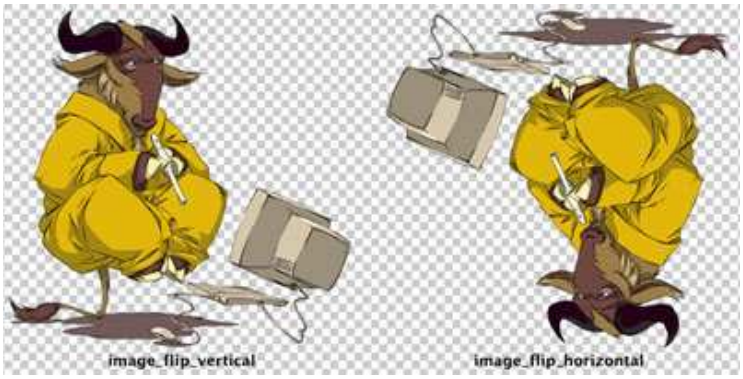
**Attention** (2) : ce filtre modifie les dimensions de l'image.

## image_flip_vertical et image_flip_horizontal

Le filtre `image_flip_vertical` applique un effet de « miroir » selon un axe vertical ; `image_flip_horizontal` selon un axe horizontal.

Très simple d'utilisation, il n'y a pas de paramètre.

```
[(#FICHIER|image_flip_vertical)]
[(#FICHIER|image_flip_horizontal)]
```



## image_masque

`image_masque` est le filtre le plus puissant de cette série. De fait, sa logique est nettement plus complexe que les autres filtres. Il permet, à partir d'un fichier PNG 24 en niveaux de gris et avec couche de transparence alpha, de modifier :

- le cadrage de l'image ;
- la transparence d'une image ;
- la luminosité d'une image.

- **Dimensions de l'image**

Si l'image d'origine est plus grande que le fichier masque, alors l'image d'origine est réduite et découpée au format du masque, puis on applique les informations de transparence et de luminosité du masque. Utile pour créer les vignettes de navigation.

Si l'image d'origine est plus petite que le masque, alors on ne recadre pas, on applique simplement les informations de luminosité et de transparence du masque (lui-même non redimensionné).

Voici notre image d'origine :

- **Masque de transparence**

Les informations de transparence du masque sont directement appliquées à l'image d'origine. Un pixel transparent du masque rend le pixel correspondant de l'image d'origine transparent, selon la même valeur de transparence. (Si l'image d'origine est déjà transparente, les informations sont mélangées de façon à conserver les deux infos de transparence.)

Si on a le fichier masque suivant, nommé « decoupe1.png » :

qu'on applique ainsi :

```
[(#FICHIER|image_masque{decoupe1.png})]
```

on obtient l'image suivante :

L'image d'origine a été redimensionnée aux dimensions de « decoupe1.png », et les zones transparentes du masque sont devenues les zones transparentes du résultat.

**Attention !** Le filtre utilise la réduction d'image. Pour qu'il fonctionne correctement, il est impératif de choisir la méthode de réduction GD2 dans la configuration du site, onglet « Fonctions avancées ».

- **Masque de luminosité**

Dans l'exemple ci-dessus, l'image masque est entièrement en gris à 50% (#808080). Les couleurs de l'image d'origine sont alors laissées inchangées (on s'est contenté de « découper » l'image.

En faisant varier les couleurs du masque, on va appliquer ces différences de luminosité à l'image traitée. Lorsqu'un pixel du masque est plus clair, alors le fichier résultant est éclairci ; si le pixel du masque est foncé, alors on fonce le fichier résultant.

Par exemple, si notre masque est « decoupe2.png » :

```
[(#FICHIER|image_masque{decoupe2.png})]
```

on obtient l'image suivante :

Dans ces deux exemples, le masque est plus petit que l'image d'origine, on obtient donc une sorte de vignette de l'image. Si le masque est plus grand que l'image d'origine, on l'applique à l'image non redimensionnée. Cela est pratique pour « texturer » une image. On peut ainsi réaliser l'effet suivant :

en appliquant un masque en niveau de gris, masque que nous avons créé plus grand que l'image d'origine.

**Attention** : l'impact sur la luminosité est plus important sur l'image finale que dans le fichier masque.

**Attention** (2) : en réalité, le masque de luminosité est un masque de coloration. Si l'image masque est colorée, alors on modifiera non seulement la luminosité, mais aussi les couleurs de l'image. Mais cet effet est particulièrement difficile à maîtriser, notamment en partant d'images en couleur.

**Attention** (3) : à partir de SPIP 2.1 la syntaxe des #LOGO_xx change. Merci de vous référer à l'article qui l'explique

## P.-S.

Sur son site http://www.paris-beyrouth.org/tutoriaux-spip, ARNO donne de nombreux exemples d'utilisation de ces filtres graphiques, et explique notamment la philosophie qui les sous-tend : http://www.paris-beyrouth.org/Pourquoi-utiliser-les-filtres

## Le traitement des images

via GD, GD2 ou Imagemagick

Avril 2005 — maj : Décembre 2007

SPIP permet d'utiliser les systèmes de traitement d'images installés sur votre site d'hébergement. Ces fonctions de SPIP sont complétées et plus puissantes.

SPIP utilise le traitement d'images de trois manières différentes :

- la création de vignettes de prévisualition pour les images installées comme « documents joints » ; SPIP permet de plus, dans ces « portfolios », de faire subir à chaque image une rotation à 90° (cela est particulièrement intéressant lorsqu'on installe une série de photographies depuis un appareil photo numérique) ;


- à de nombreux endroits dans l'espace privé, l'affichage de « vignettes » destinées à illustrer la navigation, à partir des logos des articles, des rubriques et même des auteurs (par exemple, si l'on dote les participants à un site de logos d'auteurs, des vignettes de ces logos accompagneront tous les messages de ces auteurs dans les forums de l'espace privé) ;


- dans les squelettes, les webmestres disposent d'une fonction `image_reduire` particulièrement utile pour contrôler sa mise en page, et créer différentes versions (de différentes tailles) d'une même image. Nous ne pouvons qu'encourager les

webmestres à « jouer » avec cette fonction pour enrichir et contrôler leur interface graphique ; on en tirera avantage pour obtenir :

- des alignements d'images parfaits (par exemple : toutes les images de la même largeur), sans s'obliger à installer des images de dimensions prédéfinies ;
- la garantie de ne pas faire « exploser » sa mise en page lorsqu'une image trop grande est installée par un rédacteur,
- des effets de survol et d'animation réalisés simplement en utilisant la même image à des tailles différentes (sans devoir jouer avec les « logos de survol »),
- des interfaces de portfolios (galeries de photos) épatantes...

## Choix du système de traitement d'images

Mais, pour réaliser ces opérations de traitement d'images, SPIP fait appel à des systèmes qui ne peuvent pas être installés automatiquement avec SPIP, mais doivent être présents sur le serveur qui héberge votre site. Il faut donc que ces systèmes soient présents *en plus de SPIP* (dit autrement : il ne suffit pas que SPIP soit installé pour que les fonctions de traitement d'images soient disponibles ; il faut en fait que ces fonctions soient présentes par ailleurs).


Le choix d'un système de traitement d'images se fait dans la partie « Configuration » (configuration avancée) de l'espace privé. SPIP permet de choisir parmi 5 méthodes différentes de traitement des images.

## Imagemagick

Imagemagick en tant qu'extension de PHP (php-imagick) est le choix privilégié par SPIP. SPIP est capable de déterminer seul sa présence. Si Imagemagick est présent sur votre serveur, alors SPIP l'utilisera automatiquement.

Si Imagemagick n'est pas présent sur votre serveur, alors SPIP vous proposera de choisir parmi d'autres méthodes. Ces méthodes n'étant pas détectables par SPIP (et, en tout cas, pas parfaitement), une vignette vous est proposée pour chaque méthode ou, éventuellement, pas de vignette si la méthode ne fonctionne pas sur votre site. Vous êtes alors invité à sélectionner votre méthode préférée (parfois : sélectionner la seule réellement disponible !).

### GD, GD2

GD (et sa version 2, nettement plus puissante) est une extension de PHP désormais fréquemment présente sur les serveurs, y compris les hébergeurs mutualisés.

Si GD2 est présente, vous pouvez l'utiliser, elle donne des résultats de bonne qualité.

En revanche, GD (comprendre : « version 1 de GD ») est proposée comme pis-aller : le traitement des images se fait en 256 couleurs, et introduit de fortes dégradation des images ; elle n'est donc à sélectionner que *si aucune autre méthode ne fonctionne sur votre site*.

### Imagemagick par convert

Convert est le logiciel en ligne de commande de Imagemagick. La qualité est absolument épatante, cependant son installation est relativement complexe.

Une fois *convert* installé sur votre site, vous devez configurer le chemin d'accès dans `mes_options.php3` (il s'agit d'un appel en ligne de commande) par la variable suivante :

```
$convert_command = '/bin/convert';
```

Il convient ici d'indiquer le chemin complet d'accès au programme. Sous Linux, ce chemin est souvent :

```
$convert_command = '/bin/convert';
```

sous MacOS X, s'il est installé avec Fink :

```
$convert_command = '/sw/bin/convert';
```

(ces valeurs sont fournies à titre indicatif ; comme tout programme, il peut être installé quasiment n'importe où...).

### NetPBM

Cette méthode consiste en trois programmes, déjà anciens, qui permettent de réaliser le redimensionnement de l'image. L'avantage de cette méthode est que ces programmes peuvent être installés sans accès *root* sur la plupart des hébergements.

On trouvera, sur le site du logiciel *gallery*, une explication claire et des versions précompilées de NetPBM.

Dans SPIP, on configure le chemin d'accès à pnmscale (l'un seulement des trois programmes installés - les deux autres chemins s'en déduiront, puisque les programmes sont installés dans le même répertoire) par la variable suivante :

```
$pnmscale_command = '/bin/pnmscale';
```

(encore une fois, c'est-à-vous de déterminer le chemin d'accès réel de votre installation).

Pour rappel, vous pouvez obtenir nombre d'informations utiles sur votre système via la page /ecrire/ ?exec=info, notamment : — le système utilisé (utile pour installer NetPBM précompilé) ; — la version de PHP ; — la présence éventuelle des extensions GD, GD2 et Imagemagick.

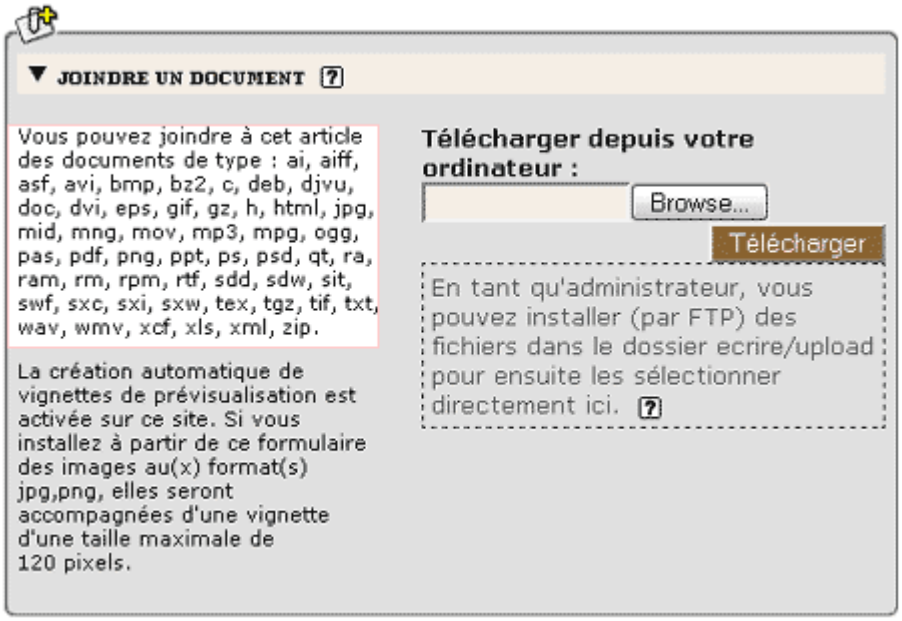
Enfin, en cas de difficulté, la meilleure solution consiste à contacter votre hébergeur pour qu'il installe les extensions nécessaires si aucune n'est présente. La présence d'au moins une extension graphique de PHP est désormais une norme chez les hébergeurs, n'hésitez pas à demander leur installation pour en bénéficier sur votre site.

## Ajouter un type de document

Août 2003 — maj : Décembre 2007

Il est possible d'installer des documents joints aux articles (et, en option, aux rubriques).

*Pour des raisons de sécurité*, SPIP n'autorise pas l'installation de n'importe quels types de documents. En effet, permettre l'installation de documents sur un serveur distant à partir d'une interface Web peut poser de sérieux problèmes de sécurité. C'est pourquoi cette liste d'autorisations existe, et pour cette même raison SPIP ne propose pas d'interface pour modifier cette liste.




Comme vous pouvez le constater, la liste des types de fichiers autorisés est déjà relativement fournie, et nous l'enrichissons régulièrement lorsque le besoin est exprimé sur les listes.

Avant de poursuivre, prenez le temps de bien lire ce qui suit :

- Cette manipulation est *potentiellement dangereuse* et peut introduire un énorme trou de sécurité dans votre site. Certains formats de fichiers (exécutables sur le serveur) ne doivent surtout pas être acceptés. En particulier, **n'acceptez jamais l'installation de fichiers de type PHP** (.php, .php3...) sur votre site, la sécurité de votre site serait *totalement compromise*. N'hésitez pas à vous renseigner avant d'ajouter des types de fichiers.
- Volontairement, nous n'avons pas installé dans SPIP d'interface pour modifier la liste des types de fichiers autorisés. Cela pour réserver cette modification aux utilisateurs confirmés. Pour effectuer la manipulation, vous devez utiliser un gestionnaire de base de données (par exemple : phpMyAdmin). Or s'attaquer directement à la base de données de SPIP « à la main » (sans utiliser l'interface et les automatismes de SPIP) est potentiellement destructeur pour votre site. Si vous ne savez pas exactement ce que vous faites, ne le faites pas. Si vous n'avez pas l'habitude de phpMyAdmin, ne l'utilisez pas sur la base de SPIP. Dans tous les cas, effectuez une sauvegarde de votre site SPIP avant de procéder à des interventions manuelles dans la base.
- Accessoirement, l'ajout d'un type de fichier ne se justifie que si les visiteurs de votre site peuvent utiliser (« lire ») ce type de fichier. Utiliser un format de fichier que l'on ne peut lire qu'avec un logiciel ultra-spécialisé sur un site grand public n'a pas

vraiment de sens. Avant d'ajouter un type de fichier, vérifiez qu'il peut bien être utilisé par vos utilisateurs (notamment : lecteur ou plug-in gratuit et facile à installer ; ce format est-il lisible sur tous les ordinateurs, Mac, PC, Linux...?).

- Pour toutes ces raisons, nous insistons sur le fait que **cette manipulation ne doit être effectuée qu'en parfaite connaissance de cause**. Si vous avez des doutes, renseignez-vous auprès de personnes compétentes ou, plus simplement encore, convertissez vos fichiers dans un format déjà autorisé et largement répandu (par exemple, un document FrameMaker ne pourra pas être lu directement par le grand public ; s'il s'agit simplement de diffuser son contenu, vous pouvez aussi bien en faire un fichier PDF directement téléchargeable par vos visiteurs plutôt que de vouloir ajouter le type « FrameMaker » dans la liste des fichiers autorisés par SPIP).

Pour ajouter un type de fichier autorisé sur votre site sous SPIP, utilisez phpMyAdmin (ou tout logiciel équivalent) pour accéder à la gestion de la base de données.

Il vous faut ajouter un nouvel élément dans la table `spip_types_documents`.

| ←T→ | | id_type | titre | descriptif | extension | mime_type | inclus | upload | maj |
|---|---|---|---|---|---|---|---|---|---|
| Edit | Delete | 59 | RealAudio | | rm | | embed | oui | 20020829183952 |
| Edit | Delete | 58 | RealAudio | | ram | | embed | oui | 20020829183952 |
| Edit | Delete | 53 | PDF | | pdf | | non | oui | 20020829183952 |
| Edit | Delete | 3 | GIF | | gif | | image | oui | 20020829183952 |

- `id_type`. Laissez ce champ vide. La numérotation des `id_type` est effectuée automatiquement par mySQL.
- `titre`. Indiquez ici le nom du type de fichier (souvent il s'agit du nom du programme qui permet de créer et lire ce type de fichier). Choisissez un nom court et aussi générique que possible. Sur un site multilingue, prenez soin à ne pas utiliser un titre propre à une langue (par exemple, un visiteur anglophone ne sera pas tellement intéressé par un fichier indiqué comme étant une « image vectorielle pour Illustrator » ; on se contentera donc d'indiquer « Adobe Illustrator »).
- `descriptif`. Laissez vide ; ce champ n'est pas utilisé.
- `extension`. Ce champ est le plus important : c'est là qu'on indique le type de fichier identifié par son extension. Par exemple, « rm » et « ram » pour du Real, « pdf » pour un fichier Acrobat PDF, « gif » pour une image au format GIF...
- `mime_type`. Laissez vide ; ce champ n'est pas utilisé.
- `inclus`. Le choix est laissé entre : « embed », « non », « image ». Prenez bien soin à effectuer ici le bon choix (c'est très important pour le bon fonctionnement de votre site) ; il détermine de quelle manière ce type de document sera « appelé » dans votre site pour être présenté aux visiteurs :
  - « non » : ce type de fichier ne peut pas être inclus directement *à l'intérieur* d'une page HTML ; on ne peut que le présenter au travers d'un lien hypertexte. Par exemple, un fichier PDF ne peut pas être affiché à l'intérieur d'une page HTML : le seul moyen de le joindre est de créer un lien hypertexte permettant d'ouvrir le fichier dans une nouvelle fenêtre (ou de le télécharger sur le disque dur du visiteur) ; un fichier de type « pdf » se voit donc attribuer le champ `inclus` fixé à « non » ;
  - « embed » : ce type de fichier peut être directement affiché à l'intérieur d'une page HTML où il sera lu grâce à une extension du butineur (plug-in...). C'est le cas de la majorité des formats multimédia utilisés sur le Web : Flash, Shockwave, films vidéo...

    - « image » : il s'agit de formats d'image affichés directement dans la page HTML sans extension particulière (avec le code HTML `<img src=...>`). A priori, vous n'aurez pas besoin d'ajouter de tels types de fichiers, la liste fournie par SPIP étant déjà exhaustive. (Notez bien : certains formats d'images réalisés avec des logiciels de « dessin » ne peuvent pas être affichés directement en tant que « image » et nécessitent une extension pour être lus ; de tels formats seront alors « embed », voire « non ».)
- `upload`. Indique que vous autorisez l'installation de ce type de fichier via l'interface Web de SPIP. On choisit donc « oui ».
- `maj`. Ce champ est géré automatiquement par SPIP. Laissez vide.

Cette opération effectuée, vous pouvez créer une nouvelle icone (vignette de prévisualisation) correspondant à ce type de fichier. Pour assurer la cohérence graphique avec les icones livrées par défaut avec SPIP, cette vignette aura une taille d'environ 48 pixels de large et 52 pixels de haut.

Vous pouvez utiliser tout type de format (GIF, PNG, JPG) ; préférez un format autorisant un fond transparent.

Le nom de votre fichier sera formé ainsi :

- l'extension du type de fichier autorisé ;
- le format graphique de cette vignette (« .gif », « .png »...) ;
- par exemple, une vignette sauvegardée en PNG, créée pour le format PDF, sera nommée : « pdf.png ».
- N.B. Les noms se terminant par « -dist » sont réservés aux fichiers distribués avec SPIP. Ainsi, n'utilisez pas le nom « pdf-dist.png », ce nom est réservé au fichier créé par les développeurs de SPIP ; si vous utilisez tout de même ce nom, votre fichier risque d'être écrasé lors de votre prochaine mise-à-jour de SPIP.

Cette vignette s'installe par FTP dans le dossier `/IMG/icones`.

# 3. Interactivité

**Les formulaires dynamiques.**

## La fonction verifier() des formulaires CVT

Avril 2010

### Comment définir la fonction

La fonction *verifier()* d'un formulaire XXX (qui sera affiché dans les squelettes par `#FORMULAIRE_XXX`) est définie dans le fichier `formulaires/xxx.php` ou dans le fichier `formulaires/xxx/verifier.php`. Le dossier `formulaires/` pouvant être rangé dans le dossier d'un plugin, ou dans le dossier squelettes.

Cette fonction devra être nommée `function formulaires_xxx_verifier_dist()`. Le suffixe *_dist* permettant à un développeur de surcharger la fonction pour changer son comportement, en créant une fonction `function formulaires_xxx_verifier()`

### Les arguments de la fonction

La fonction *verifier()* reçoit automatiquement la valeur de chaque argument qui sont passés à la balise #FORMULAIRE_XX, dans le même ordre. Par exemple en écrivant :

```
#FORMULAIRE_XX{#ID_ARTICLE,#ID_RUBRIQUE}
```

et la fonction :

```
formulaires_xxx_verifier_dist($arg1,$arg2){
...
}
```

$arg1 vaudra #ID_ARTICLE, et $arg2 vaudra #ID_RUBRIQUE.

### Que doit faire la fonction

La fonction *verifier()* doit renvoyer un tableau d'erreurs de saisies résultant de la vérification des données saisies.

Si toute la saisie est correcte, le tableau sera vide, et SPIP appellera alors la fonction *traiter()* du formulaire, chargée de finir le travail.

Si le tableau renvoyé par la fonction *verifier()* n'est pas vide, alors la fonction *traiter()* ne sera pas appelée, le formulaire sera de nouveau affiché avec les messages d'erreur éventuels pour correction par l'utilisateur.

Il convient donc de faire toutes les vérifications nécessaires au traitement dans la fonction *verifier()* pour assurer la meilleure interactivité possible.

Chaque erreur est renvoyée sous forme d'une association `cle=>valeur`, et il est d'usage d'utiliser comme clé le nom de la variable saisie dans le formulaire.

**Exemple de fonction verifier()**
Voyons un exemple :

```
function formulaires_xxx_verifier_dist(){
      $erreurs = array();
      if (!_request('nom'))
            $erreurs['nom'] = _T('info_obligatoire');
      include_spip('inc/filtres');
      if ($email = _request('email')  AND !email_valide($email))
            $erreurs['email'] = _T('form_email_non_valide');

      return $erreurs;
}
```

Ici, la fonction

- vérifie que le nom a bien été renseigné, et produit un message d'erreur dans le cas contraire ;
- regarde si une adresse mail a été saisie, et dans ce cas vérifie sa validité, avec un message d'erreur si l'adresse mail est incorrecte.

## Champs particuliers

La fonction *verifier()* peut renvoyer certaines valeurs particulières dans le tableau :

**message_erreur**
Cette valeur est utilisée dans le squelette du formulaire pour afficher un message d'erreur général, qui concerne tout le formulaire. Il peut être judicieux de renseigner ce message dès qu'une erreur sur un champ a été identifiée

**message_ok**
Cette valeur permet de renvoyer un message de succès. Ce message peut-être utile quand par exemple l'internaute a entré une première valeur qui permet de vérifier certaines informations, sans que la saisie du formulaire ne soit finie pour autant.
La présence de ce message empêche l'appel à la fonction *traiter()* du formulaire, qui n'est appelée que si, et seulement si, le tableau retourné par *verifier()* est vide.

## Personnalisation

Comme la fonction *charger()*, la fonction *verifier()* d'un formulaire existant peut être personnalisée par deux mécanismes

**Surcharge**
Comme indiqué ci-dessus, il est possible de redéfinir la fonction *verifier()* par défaut en définissant sa propre fonction `function formulaires_xxx_verifier()` qui sera appelée à la place de la fonction par défaut qui comporte le suffixe _dist. Cette fonction surchargée pourra être définie dans le fichier `formulaires/xxx/verifier.php`, ou dans un fichier `options.php` appelé à chaque hit de façon à ce que la fonction soit définie au moment où SPIP va la chercher.

**Pipeline**
Le pipeline *formulaire_verifier* permet de modifier le résultat de la fonction *verifier()* par défaut de n'importe quel formulaire CVT.

C’est la méthode qu’il faut privilégier dans un plugin.

Le pipeline reçoit en argument un tableau de cette forme :

```
array(
      'args'=>array('form'=>$form,'args'=>$args),
      'data'=>$erreurs)
)
```

En écrivant la fonction pipeline sous cette forme :

```
function monplugin_formulaire_verifier($flux){
...
}
```

Alors vous trouverez dans $flux les éléments suivants :

| | |
|---|---|
| `$flux['args']['form']` | nom du formulaire (xxx dans notre exemple) |
| `$flux['args']['args']` | arguments de la fonction *charger()* dans l’ordre où ils ont été passés à la balise #FORMULAIRE_XXX |
| `$flux['args']['data']` | tableau `$erreurs` renvoyé par la fonction *verifier()* par défaut |

Tous les formulaires passent par le même pipeline. Il faut donc tester la valeur de `$flux['args']['form']` pour ne modifier que le comportement du formulaire xxx.

## La fonction traiter() des formulaires CVT

Avril 2010

### Comment définir la fonction

La fonction *traiter()* d'un formulaire XXX (qui sera affiché dans les squelettes par `#FORMULAIRE_XXX`) est définie dans le fichier `formulaires/xxx.php` ou dans le fichier `formulaires/xxx/traiter.php`. Le dossier `formulaires/` pouvant être placé dans le dossier d'un plugin, ou dans le dossier squelettes.

La fonction devra être nommée `function formulaires_xxx_traiter_dist()`. Le suffixe *_dist* permettant à un développeur de redéfinir la fonction pour changer son comportement, en créant une fonction `function formulaires_xxx_traiter()`

### Les arguments de la fonction

Comme les fonctions *charger()* et *verifier()*, la fonction *traiter()* reçoit automatiquement, dans le même ordre, la valeur de chacun des arguments passés à la balise #FORMULAIRE_XX. Par exemple en écrivant :

```
#FORMULAIRE_XX{#ID_ARTICLE,#ID_RUBRIQUE}
```

et la fonction :

```
formulaires_xxx_traiter_dist($arg1,$arg2){
...
}
```

$arg1 vaudra #ID_ARTICLE, et $arg2 vaudra #ID_RUBRIQUE.

### Que doit faire la fonction

La fonction *traiter()* fait tous les traitements nécessaires suite à la saisie de l'internaute, et après les vérifications faites par la fonction *verifier()*.

Le point important ici est que la fonction *traiter()* sera toujours appelée juste après la fonction *verifier()* et uniquement si celle-ci a bien renvoyé un tableau vide signifiant l'absence d'erreurs.

Une fois les actions réalisées (par exemple : enregistrement des données, écriture dans un fichier, envoi de mail ...), la fonction *traiter()* doit renvoyer un tableau de valeurs décrivant le résultat de l'action.

Ce tableau renvoyé par *traiter()* doit renseigner au moins une des deux valeurs `message_ok` ou `message_erreur` pour indiquer le succès ou l'échec éventuel du traitement.

**message_ok**
Si tout c'est bien déroulé, l'indiquer dans un message permet à l'utilisateur d'être rassuré sur l'action réalisée. C'est une bonne pratique de toujours renvoyer un message, même si tout se passe bien !

**message_erreur**
Malgré les vérifications réalisées par la fonction *verifier()*, le traitement peut échouer pour d'autres raisons non détectables à l'avance. Dans ce cas la fonction *traiter()* indique la cause de l'erreur dans message_erreur, et l'utilisateur sera informé du problème.

**editable**
Par défaut, après la saisie du formulaire et son traitement avec succès, la saisie du formulaire ne sera pas re-proposée, seul le message de succès ou d'erreur étant affiché.

En renvoyant une valeur `true` pour ce champ, cela permet de demander à l'utilisateur de remplir à nouveau le formulaire.

**id_xx**
Si le traitement a inséré ou modifié une (ou plusieurs) donnée en base, il est judicieux de renvoyer les clé primaires, généralement commençant par `id_`, des enregistrements concernés.

Ainsi, un plugin qui veut intervenir à la suite du traitement par défaut sait sur quel objet a porté l'opération, et peut agir en complément.

**redirect**
Il suffit d'indiquer une url dans ce champ pour rediriger l'internaute sur une autre page après la saisie.

Attention, il est important que la fonction *traiter()* ne fasse pas elle-même la redirection, car cela empêcherait tout intervention d'un plugin à la suite de la fonction *traiter()*.

**Exemple de fonction traiter()**
Voyons un exemple :

```
function formulaires_xxx_traiter_dist() {
	$res = array();
	$nom = _request('nom');
	$prenom = _request('prenom');
	$email = _request('email');

	include_spip('action/editer_auteur');
	if ($id_auteur = insert_auteur()) {
		auteurs_set($id_auteur,array('nom'=>"$nom
$prenom",'email'=>$email));
		$res['message_ok'] = "Enregistrement réussi !";
		$res['id_auteur'] = $id_auteur;
	}
	else
		$res['message_erreur'] = "Un probleme a été rencontré,
impossible
d'enregistrer votre saisie";

	return $res;
}
```

Ici, la fonction

- récupère les informations saisies ;
- ajoute un auteur dans la table des auteurs de SPIP ;
- enregistre les nom, prénom et email saisis pour cet auteur ;
- indique dans le résultat l'id_auteur ajouté, ainsi que le message de succès.

Si l'auteur ne peut être ajouté, un message d'erreur est renvoyé.

## Cas particuliers

Les formulaires peuvent être facilement ajaxés en les incluant dans une classe ajax. Les fonctions *charger()*, *verifier()* et *traiter()* n'en savent rien, et ne voient pas de différence.

Cependant, il est parfois nécessaire que la fonction *traiter()* ne soit pas appelée en ajax. Par exemple parcequ'elle va modifier fortement la base de données et qu'il est préférable que toute la page soit ré-affichée après la saisie, pas seulement le formulaire.

Dans ce cas, il faut commencer la fonction *traiter()* par un appel à la fonction `refuser_traiter_formulaire_ajax()`.

Si le formulaire n'a pas été soumis en ajax, cette fonction ne fera rien, et la suite de la fonction *traiter()* sera exécutée normalement.

Si le formulaire a été soumis en ajax, alors cette fonction va tout arrêter en renvoyant un code spécifique au javascript de la page. Lorsque le javascript reçoit ce code, il provoque à nouveau un envoi des données du formulaire, mais sans ajax cette fois-ci. SPIP va alors de nouveau appeler la fonction *verifier()*, puis la fonction *traiter()*, qui cette fois s'exécutera complètement et pourra réaliser ses opérations.

Il est donc primordial que la fonction *traiter()* commence par cet appel à `refuser_traiter_formulaire_ajax()`, et ne fasse aucune manipulation avant d'appeler cette fonction, car sinon ces éventuelles opérations seraient exécutées deux fois.

## Personnalisation

Comme les fonctions *charger()* et *verifier()*, la fonction *traiter()* d'un formulaire existant peut être personnalisée par deux mécanismes

**Surcharge**
Comme indiqué ci-dessus, il est possible de redéfinir la fonction *traiter()* par défaut en définissant sa propre fonction `function formulaires_xxx_traiter()` qui sera appelée à la place de la fonction par défaut qui comporte le suffixe _dist.
Cette fonction surchargée pourra être définie dans le fichier `formulaires/xxx/traiter.php`, ou dans un fichier `options.php` appelé à chaque hit de façon à ce que la fonction soit définie au moment où SPIP va la chercher.

**Pipeline**
Le pipeline *formulaire_traiter* permet de modifier le résultat de la fonction *traiter()* par défaut de n'importe quel formulaire CVT.

C'est la méthode qu'il faut privilégier dans un plugin.

Le pipeline reçoit en argument un tableau de cette forme :

```
array(
      'args'=>array('form'=>$form,'args'=>$args),
      'data'=>$res)
)
```

En écrivant la fonction pipeline sous cette forme :

```
function monplugin_formulaire_traiter($flux){
...
}
```

Alors vous trouverez dans $flux les éléments suivants :

| | |
|---|---|
| `$flux['args']['form']` | nom du formulaire (xxx dans notre exemple) |
| `$flux['args']['args']` | arguments de la fonction *charger()* dans l'ordre où ils ont été passés à la balise #FORMULAIRE_XXX |
| `$flux['args']['data']` | tableau `$res` renvoyé par la fonction *traiter()* par défaut |

Tous les formulaires passent par le même pipeline. Il faut donc tester la valeur de `$flux['args']['form']` pour ne modifier que le comportement du formulaire xxx.

## La fonction charger() des formulaires CVT

Août 2009 — maj : août 2010

### Comment définir la fonction

La fonction *charger()* d'un formulaire XXX (qui sera affiché dans les squelettes par `#FORMULAIRE_XXX`) est définie dans le fichier `formulaires/xxx.php` ou dans le fichier `formulaires/xxx/charger.php`. Le dossier `formulaires/` pouvant être placé dans le dossier d'un plugin, ou dans le dossier squelettes.

Cette fonction devra être nommée `function formulaires_xxx_charger_dist()`. Le suffixe *_dist* permettant à un développeur de redéfinir la fonction pour changer son comportement, en créant une fonction `function formulaires_xxx_charger()`

### Les arguments de la fonction

La fonction *charger()* reçoit automatiquement, *dans le même ordre*, la valeur de chaque argument passé à la balise #FORMULAIRE_XX. Par exemple en écrivant :

```
#FORMULAIRE_XX{#ID_ARTICLE, #ID_RUBRIQUE}
```

et la fonction :

```
formulaires_xxx_charger_dist($arg1, $arg2) {
...
}
```

$arg1 vaudra #ID_ARTICLE, et $arg2 vaudra #ID_RUBRIQUE.

### Que doit faire la fonction

La fonction *charger()* doit renvoyer un tableau de valeurs par défaut pour chaque champ de saisie du formulaire. Ce tableau a une double fonction :

- déclarer la liste des champs saisis (la liste des « name » des input du formulaire)
- fournir une valeur par défaut pour chaque saisie

Les valeurs par défaut fournies seront utilisées au premier affichage du formulaire, mais seront ensuite écrasées par les saisies de l'internaute en cas de re-présentation du formulaire suite à une erreur de saisie.

**Exemple de fonction charger()**
Voyons un exemple :

```
function formulaires_xxx_charger_dist() {
  $valeurs = array();
  $valeurs['nom'] = 'Indiquez votre nom';
  $valeurs['prenom'] = '';
  $valeurs['email'] = '';
  return $valeurs;
}
```

Ici, la fonction *charger()* va renvoyer un tableau associatif avec 3 champs 'nom', 'prenom', 'email' et une valeur par défaut pour chacun d'entre-eux.

Même si la valeur par défaut est vide, il est important de faire figurer le champ dans le tableau retourné par la fonction *charger()*. En effet, SPIP utilise ensuite ce tableau pour savoir quels champs seront saisis.

Il n'est pas nécessaire de protéger la valeur des champs si ils contiennent des guillemets par exemple : SPIP le fera automatiquement.

Chaque valeur du champ sera retrouvée dans le squelette du formulaire html avec, par exemple ici, `#ENV{nom}`, `#ENV{prenom}`, `#ENV{email}`

## Champs particuliers

La fonction *charger()* peut renvoyer certaines valeurs particulières

**editable**
Cette valeur est utilisée dans le squelette du formulaire pour permettre ou non la saisie. Par défaut, après le traitement du formulaire, « editable » est faux et la saisie du formulaire n'est pas re-proposée. Seul le message de succès est affiché.

Il est possible de fournir ici une valeur pour « editable » pour gérer des conditions particulières sur l'activation ou non de la saisie du formulaire.

**message_ok**
Le message de succès est fourni, en principe, par la fonction *traiter()*. Il est néanmoins possible de le fournir par la fonction *charger()* de manière dérogatoire.

**message_erreur**
Le message d'erreur est fourni, en principe, par la fonction *traiter()*. Il est néanmoins possible de le fournir par la fonction *charger()* de manière dérogatoire.

**action**
Cette valeur précise l'URL sur laquelle est postée le formulaire. C'est par défaut l'URL de la page en cours. Il est important que cela reste le cas général car en cas d'erreur de saisie, il faut pouvoir re-présenter le formulaire.

Dans certains cas dérogatoires, il peut être utile de modifier cette URL. À réserver à des usages très particuliers.

**_forcer_request**
Par défaut, SPIP vérifie que le formulaire posté est bien le bon pour permettre d'avoir plusieurs formulaires du même type dans une page, et ne traiter que celui qui a été soumis. La vérification est basée sur la liste des arguments passés à la balise `#FORMULAIRE_XXX`.

Dans certains cas, quand ces arguments changent suite à la saisie, SPIP peut se tromper et croire que la saisie vient d'un autre formulaire.
En lui passant `true` pour `_forcer_request`, vous lui indiquez qu'il ne doit pas faire cette vérification et qu'il doit traiter la saisie dans tous les cas.

**_action**
Si le traitement du formulaire xxx doit faire appel à une fonction du répertoire actions/ qui est protégée par *securiser_action()*, il est possible, en indiquant le nom de l'action, que SPIP fournisse automatiquement le hash de protection correspondant.

**_hidden**
La valeur de ce champ sera ajoutée directement dans le code HTML du formulaire généré. Elle est utilisée pour y ajouter des input hidden qui devront être écrits explicitement :

```
$valeurs['_hidden'] = "<input type='hidden' name='secret'  value='hop' />";
```

**_pipeline**
Ce champ permet de demander à ce que le résultat du calcul du formulaire passe dans le pipeline mentionné, permettant son extension par des plugins.
Ce peut être une simple chaîne, pour définir le nom du pipeline :

```
$valeurs['_pipeline'] = "monpipeline";
```

ou un tableau de deux valeurs, pour préciser des arguments à passer au pipeline :

```
$valeurs['_pipeline'] = array("monpipeline", array('arg1' => 'unevaleur',
...));
```

**Pré-fixage et protection des champs pour saisie**
Par défaut, toutes les valeurs du tableau sont protégées pour pouvoir être insérées en toute sécurité dans l'attribut `value` d'un `input` dans le squelette du formulaire. Lorsque la valeur est elle-même un tableau, chacune de ses valeurs est protégée et ainsi de suite.

Toutefois, les champs préfixés par un _ (*underscore*) ne seront pas protégés automatiquement par SPIP. Lorsqu'il est nécessaire de passer des valeurs au squelette, mais qui ne seront pas utilisées pour la saisie, il est conseillé d'utiliser systématiquement ce préfixe.

Par ailleurs, il ne faut pas utiliser un préfixe _ (*underscore*) pour un champ de saisie, car la valeur entrée par l'utilisateur est ignorée lorsque le formulaire est re-présenté en cas d'erreur.

## Personnalisation

Une fonction *charger()* d'un formulaire existant peut être personnalisée par deux mécanismes :

**Surcharge**
Comme indiqué ci-dessus, il est possible de redéfinir la fonction *charger()* par défaut en définissant sa propre fonction `function formulaires_xxx_charger()` qui sera appelée à la place de la fonction par défaut qui comporte le suffixe _dist. Cette surcharge pourra être placée dans le fichier `formulaires/xxx/charger.php`, ou dans un fichier `options.php` toujours chargé.

**Pipeline**
Le pipeline *formulaire_charger* permet de modifier le résultat de la fonction *charger()* par défaut de n'importe quel formulaire CVT.

C'est la méthode qu'il faut privilégier dans un plugin.

Le pipeline reçoit en argument un tableau de cette forme :

```
array(
      'args' => array('form' => $form, 'args' => $args),
      'data' => $valeurs)
)
```

En écrivant la fonction pipeline sous cette forme :

```
function monplugin_formulaire_charger($flux){
...
}
```

son argument comportera les éléments suivants :

| | |
|---|---|
| `$flux['args']['form']` | nom du formulaire (xxx dans notre exemple) |
| `$flux['args']['args']` | arguments de la fonction *charger()* dans l'ordre où ils ont été passés à la balise #FORMULAIRE_XXX |
| `$flux['args']['data']` | tableau `$valeurs` renvoyé par la fonction *charger()* par défaut |

Tous les formulaires passent par le même pipeline. Il faut donc tester la valeur de `$flux['args']['form']` pour ne modifier que le comportement du formulaire concerné.

## Structure HTML des formulaires de SPIP 2

Formulaires d'édition utilisés dans l'espace privé

Juin 2009 — maj : Juillet 2009

Document de références précisant comment les formulaires dans SPIP doivent être structurés

### Structure HTML

Un formulaire de base est ainsi structuré :

```
<div class="formulaire_spip formulaire_editer
formulaire_editer_nomformulaire" id="formulaire_editer_nomformulaire-id">
  <a id="nomformulaire" name="nomformulaire"></a>
  <form action="#" method="post">
      <fieldset>
            <legend>Une légende</legend>
            <p class="explication">Un texte d'explication</p>
            <ul>
                  <li class="editer_nomlabel obligatoire erreur">
                        <label for="nomlabel">Courriel</label>
                        <em class="aide">#AIDER{arttitre}</em>
                        <p class="explication"> Explication du label</p>
                        <span class="erreur_message">Message
                        d'erreur</span>
                        <input type="type" class="type" name="nomlabel"
                        id="nomlabel" value="" />
                  </li>
            </ul>
      </fieldset>
</form>
</div>
```

Le div englobant porte la classe générique **`formulaire_spip`**. Les formulaires d'édition de l'espace privé portent de plus la classe **`formulaire_editer`** indiquant qu'on a affaire à un formulaire d'édition de données d'une base.

Le premier fieldset étant optionnel, on peut aussi écrire, sans le fieldset et sans les paragraphes optionnels :

```
<div class="formulaire_spip formulaire_editer
formulaire_editer_nomformulaire formulaire_editer_nomformulaire-id">
<a id="nomformulaire" name="nomformulaire"></a>
<form action="#" method="post">
      <ul>
            <li class="editer_nomlabel obligatoire">
                  <label for="nomlabel">Courriel</label>
                  <input type="type" class="type" name="nomlabel"
                  id="nomlabel" value="" />
            </li>
      </ul>
</form>
</div>
```

## Les classes spéciales

- « **`explication`** » : pour indiquer un message d'explication (qui porte soit pour l'ensemble des champs, soit sur une étape). Exemple : `<p class="explication">`.
- « **`attention`** » : pour afficher un message concernant un champ d'édition critique. Exemple : `<em class="attention"><:texte_login_precaution:></em>`.
- « **`obligatoire`** » : pour signaler un champ obligatoire, à appliquer à l'élément de liste parent. Exemple : `<li class="obligatoire">`.
- « **`erreur`** » : pour signaler une étape en erreur, à appliquer à l'élément de liste parent. Exemple : `<li class="erreur">`. chaque erreur bénéficie d'un message explicatif, portant la class « `erreur_message` » : `<span class="erreur_message">`.

## Cadre englobant

Ce formulaire peut optionnelement être inclu dans un cadre-formulaire-editer, et peut alors contenir un entete-formulaire :

```
<div class="cadre-formulaire-editer">
      <div class="entete-formulaire"></div>
      <div class="formulaire_editer formulaire_editer_site
formulaire_editer_site-#ENV{id_site,nouveau}"></div>
</div>
```

## Gestion des messages réussite/erreur

### Messages globaux

Un formulaire comporte obligatoirement deux paragraphes permettant d'afficher les réussites et erreurs globales qui ont pu apparaître à la soumission. Les variables d'environnements `message_ok` et `message_erreur` sont des retours envoyés par SPIP (formulaires CVT).

```
<div class="formulaire_editer formulaire_editer_site
formulaire_editer_site-#ENV{id_site,nouveau}">
      [<p class="reponse_formulaire
reponse_formulaire_ok">(#ENV*{message_ok})</p>]
      [<p class="reponse_formulaire
reponse_formulaire_erreur">(#ENV*{message_erreur})</p>]
</div>
```

### Messages spécifiques

Chaque champ de formulaire, encapsulé dans un li peut recevoir un message d'erreur spécifique. Celui-ci est contenu dans la le tableau d'environnement 'erreurs' et peut être obtenu de la sorte :

```
[(#ENV**{erreurs}|table_valeur{nom_du_champ})]
```

On peut attribuer la classe ’erreur’ au li et afficher une erreur spécifique si elle existe de la sorte :

```
<li class="editer_descriptif[
      (#ENV**{erreurs}|table_valeur{descriptif}|oui)erreur]">
      <label for="descriptif"><:texte_descriptif_rapide:></label>
      [<span class='erreur_message'>(#ENV**{erreurs}
            |table_valeur{desctiptif})
      </span>]
      <textarea name='descriptif' id='descriptif' rows='2'
            cols='40'>[(#ENV**{descriptif})]
      </textarea>
</li>
```

## Particularités pour les styles css

### Champs input

Chaque `<input />` différent de `hidden` doit posséder une classe identique à son type (afin de palier à un déficience du navigateur Internet Explorer) :

```
<input type="text" class="text" name="titre" id="titre"
value="[(#ENV**{titre})]" />
```

### Boutons de soumission

Les boutons de soumissions sont inclus dans une boîte `.boutons` (qui peut recevoir plusieurs boutons) :

```
<p class="boutons"><input type="submit" class="submit"
value="<:bouton_enregistrer:>" /></p>
```

### radio/checkbox

Dans le cas de bouton radio ou checkbox, on peut ne pas reprendre tout a fait la même structure, par exemple pour avoir le bouton avant le label, ou pour avoir la liste radio en horizonal

Chaque entrée (radio + label) peut alors être encadrée par un bloc .choix :

```
<li class="editer_syndication">
      <div class="choix"><input type='radio' class="radio"
      name='syndication' value='non' id='syndication_non'
      [(#ENV{syndication}|=={non}|?{'checked="checked"'})] />
            <label for='syndication_non'> <:bouton_radio_non_syndication:>
            </label>
      </div>
      <div class="choix">
            <input type='radio' class="radio" name='syndication' alue='oui'
            id='syndication_oui'[(#ENV{syndication}|=={oui}|?
            {'checked="checked"'})] />
            <label for='syndication_oui'><:bouton_radio_syndication:>
            <em>#AIDER{rubsyn}</em>
            </label>
      </div>
</li>
```

Par défaut, la liste est verticale. Pour rendre la liste horizontale, il suffit de spécifier que .champ en question est de type inline :

```
.formulaire_editer .editer_syndication .choix {display:inline;}
```

## P.-S.

Principes généraux de ces formulaires pour avoir les labels alignés à gauche

On se retrouve dans un cas où l'on a :

```
<ul>
      <li>
            <label />
            <input />
      </li>
      <li>
            <label />
            <em />
            <span />
            <input />
      </li>
</ul>
```

1) on applique un padding gauche au LI supérieur à la taille du label (ex:120px)

```
.formulaire_spip li {
      margin: 0;
      padding: 10px 10px 10px 130px;
      clear:both;
      border-top: 1px solid
#[(#GET{foncee}|couleur_eclaircir|couleur_eclaircir)];
}
```

2) on demande au label d'être flottant et décalé sur la gauche (via un margin négatif)

```
.formulaire_spip label {
      width: 120px;
      float:left;
      margin-left:-125px;
      text-align: left;
      vertical-align: top;
}
```

3) lorsqu'un fieldset suit un li, on le décale aussi vers la gauche (seul IE n'en tient pas compte) pour que le fieldset prenne toute la largeur aussi

```
.formulaire_spip li fieldset {
      border:1px solid #888;
      background:white;
      margin-left:-125px; /* redecalage vers la gauche... IE < 8 ne le
                              prend pas en compte */
}
```

## Formulaires CVT par l'exemple

Un formulaire de contact en 4 étapes

Décembre 2008 — maj : avril 2010

### Étape 1 : Créer le squelette de son formulaire

Dans le sous-répertoire formulaires/ de votre dossier « squelettes », créez le fichier contact.html qui va contenir le code HTML de votre formulaire de contact. Par exemple :

```
<form action='#ENV{action}' method='post'>
      #ACTION_FORMULAIRE{#ENV{action}}
      <label>Votre email</label>
      <input type='text' name='email' value='#ENV{email}' />
      <br />
      <label>Votre message</label>
      <textarea name='message'>#ENV{message}</textarea>
      <input type='submit' name='ok' value='ok' />
</form>
```

Comme vous pouvez le constater, ce formulaire ne contient que quelques particularités :

- l'attribut action de la balise `<form>` contient `#ENV{action}` qui est l'url de la page en cours sur laquelle le formulaire sera envoyé
- `#ACTION_FORMULAIRE{#ENV{action}}` indique à SPIP de prendre en charge les données envoyées par ce formulaire et les appels vers les fonctions « charger, verifier, traiter »
- chaque input, textarea ou select contient la valeur `#ENV{xxx}` où xxx correspond à l'attribut name correspondant.

À ce stade, vous pouvez déjà afficher ce formulaire dans un squelette, avec la balise `#FORMULAIRE_CONTACT` ou directement dans le texte d'un article avec le code `<formulaire|contact>`. Vous apprécierez de pouvoir tester et affiner le rendu et le contenu de votre formulaire sans écrire une seule ligne de PHP.

Toutefois, si vous saisissez du texte et cliquez sur le bouton OK, votre formulaire ne fait rien, et vous ne retrouvez pas votre saisie.

### Étape 2 : la fonction charger

Nous allons donc maintenant indiquer à SPIP quels sont les champs que l'internaute peut remplir.

Créez pour cela un fichier contact.php dans le sous-répertoire formulaires/ (juste à côté de votre squelette contact.html, donc), et mettez-y le code suivant :

```
<?php
function formulaires_contact_charger_dist(){
      $valeurs = array('email'=>'','message'=>'');
      return $valeurs;
}
?>
```

Que fait ce code ?

Il déclare la fonction **charger** du formulaire **contact** du répertoire **formulaires** consécutivement nommée `formulaires_contact_charger`.
Le suffixe _dist signale qu'il s'agit de la fonction charger par défaut de ce formulaire, mais qu'elle est éventuellement personnalisable (voir au sujet des surcharges http://programmer.spip.org/Surcharger-une-fonction-_dist).

Cette fonction liste, dans la variable `$valeurs`, les champs de saisie de votre formulaire et, pour chaque champ, la valeur initiale par défaut.
La fonction `formulaires_contact_charger_dist` renvoie donc cette liste de champs.

Testez à nouveau votre formulaire : si vous remplissez les champs et validez avec le bouton OK, vous verrez que cette fois-ci le formulaire n'a pas perdu les valeurs que vous avez saisies (même s'il ne fait toujours rien).

Dans cet exemple, tous les champs sont vides par défaut. Nous pourrions améliorer notre fonction en remplissant automatiquement l'email lorsque le visiteur est identifié.
Cela nous donnerait le code suivant pour la fonction charger :

```
function formulaires_contact_charger_dist(){
      $valeurs = array('email'=>'','message'=>'');
      if ($GLOBALS['visiteur_session']['email'])
            $valeurs['email'] = $GLOBALS['visiteur_session']['email'];
      return $valeurs;
}
```

Si vous testez maintenant votre formulaire, en étant identifié, vous constaterez :

- que le champ email est déjà rempli avec l'email de votre compte SPIP
- que les valeurs saisies ou modifiées sont conservées telles quelles après validation

## Étape 3 : Vérifier que la saisie est correcte

Avant de prendre en compte les valeurs saisies par l'internaute nous devons bien évidemment vérifier que ce qu'il nous envoie est correct.

Nous allons, par exemple, définir les contraintes suivantes :

- les champs email et message sont obligatoires
- l'email doit être valide

Pour vérifier la saisie, nous allons créer la fonction `formulaires_contact_verifier_dist` (sur le même modèle que la fonction charger) toujours dans le même fichier contact.php :

```
function formulaires_contact_verifier_dist(){
	$erreurs = array();
	// verifier que les champs obligatoires sont bien la :
	foreach(array('email','message') as $obligatoire)
		if (!_request($obligatoire)) $erreurs[$obligatoire] = 'Ce champ
		est obligatoire';

	// verifier que si un email a été saisi, il est bien valide :
	include_spip('inc/filtres');
	if (_request('email') AND !email_valide(_request('email')))
		$erreurs['email'] = 'Cet email n\'est pas valide';
	if (count($erreurs))
		$erreurs['message_erreur'] = 'Votre saisie contient des erreurs
!';
	return $erreurs;
}
```

Remarque : l'utilisation de `_request()` est expliquée sur le site programmer.spip.org

La fonction verifier renvoie une liste de champs en erreurs, avec le message d'erreur correspondant à chaque champ.

À ce stade, vous ne verrez aucune différence si vous testez le formulaire : en effet, votre formulaire contact.html n'affiche pas les erreurs. Vous allez donc le compléter comme suit :

```
[<p class='formulaire_erreur'>(#ENV*{message_erreur})</p>]
<form action='#ENV{action}' method='post'>
	#ACTION_FORMULAIRE{#ENV{action}}
	<label>Votre email</label>
	[<span class='erreur'>(#ENV**{erreurs}|table_valeur{email})</span>]
	<input type='text' name='email' value='#ENV{email}' />
	<br />
	<label>Votre message</label>
	[<span class='erreur'>(#ENV**{erreurs}|table_valeur{message})</span>]
	<textarea name='message'>#ENV{message}</textarea>
	<input type='submit' name='ok' value='ok' />
</form>
```

Notez bien :

- l'affichage d'un message d'erreur général renvoyé par la fonction verifier `[<p class='reponse_formulaire reponse_formulaire_erreur'>(#ENV*{message_erreur})</p>]`
- les messages d'erreur devant chaque champ : `[<span class='erreur_message'>(#ENV**{erreurs}|table_valeur{email})</span>]`

Vous pouvez maintenant tester votre formulaire en saisissant un email erroné ou en laissant un champ vide : votre formulaire commence déjà à interagir avec l'internaute !

## Étape 4 : traiter !

Lorsque la fonction verifier ne renvoie aucune erreur, SPIP appelle automatiquement la fonction traiter correspondante. Déclarons cette fonction

`formulaires_contact_traiter_dist` (toujours dans le fichier contact.php) et faisons-lui envoyer un mail au webmestre :

```
function formulaires_contact_traiter_dist(){
      $envoyer_mail = charger_fonction('envoyer_mail','inc');
      $email_to = $GLOBALS['meta']['email_webmaster'];
      $email_from = _request('email');
      $sujet = 'Formulaire de contact';
      $message = _request('message');
      $envoyer_mail($email_to,$sujet,$message,$email_from);
      return array('message_ok'=>'Votre message a bien été pris en compte.
            Vous recevrez prochainement une réponse !');
}
```

Vous remarquerez que la fonction traiter ne fait aucune vérification : elles ont toutes été faites au préalable (dans la fonction verifier). Si la fonction traiter est appelée c'est qu'il n'y a aucune erreur.

La fonction traiter renvoie un message de succès, « message_ok », contenu dans un tableau. Comme pour les messages d'erreur, il faut compléter notre formulaire pour qu'il affiche ce message. Comme suit :

```
[<p class='formulaire_ok'>(#ENV*{message_ok})</p>]
[<p class='formulaire_erreur'>(#ENV*{message_erreur})</p>]
[(#EDITABLE|oui)  <form action='#ENV{action}' method='post'>
            #ACTION_FORMULAIRE{#ENV{action}}
            <label>Votre email</label>
            [<span class='erreur'>
            (#ENV**{erreurs}|table_valeur{email})</span>]
            <input type='text' name='email' value='#ENV{email}' />
            <br />
            <label>Votre message</label>
            [<span class='erreur'>
            (#ENV**{erreurs}|table_valeur{message})</span>]
            <textarea name='message'>#ENV{message}</textarea>
            <input type='submit' name='ok' value='ok' />
      </form>
]
```

Notez bien :

- l'affichage d'un message de succès au début `[<p class="formulaire_message">(#ENV*{message_ok})</p>]`
- l'affichage conditionnel du formulaire de saisie en fonction de #EDITABLE : après la saisie, il est en général plus clair pour l'internaute de ne pas réafficher le formulaire complet mais juste le message de succès. #EDITABLE sert à cela.

**C'est fini** ! Vous pouvez utiliser votre formulaire de contact.

## Le bonus

Vous souhaitez que l'interaction entre votre formulaire et l'internaute soit plus rapide et ne passe pas par le rechargement complet de la page à chaque erreur ou lors de la validation ?

Il faut que votre formulaire soit implémenté en AJAX. D'habitude c'est là que se situe la difficulté, ce type d'implémentation étant souvent long et laborieux. Mais c'est simplifié dans

SPIP, par le formalisme CVT, qui vous fait bénéficier de mécanismes automatisés, dont l'ajaxisation de vos formulaires.

Pour cela, il suffit de placer votre formulaire dans une div dotée de la class ajax :

```
<div class='ajax'>
#FORMULAIRE_CONTACT
</div>
```

**Et le tour est joué !**

## Pour aller plus loin

- Cet exemple simple ne produit pas un joli formulaire. Pour un résultat plus esthétique, sémantique et accessible, n'hésitez pas à utiliser les recommandations de structures de Structure HTML des formulaires de SPIP 2 pour tous vos formulaires SPIP
- Tout savoir sur la fonction *charger()* : La fonction charger() des formulaires CVT
- Tout savoir sur la fonction *verifier()* : La fonction verifier() des formulaires CVT
- Tout savoir sur la fonction *traiter()* : La fonction traiter() des formulaires CVT

## Les formulaires CVT de SPIP 2.0

Décembre 2008 — maj : avril 2010

La réalisation de formulaires dynamiques a été simplifiée avec SPIP 2.0 grâce à un formalisme clair découpé en une vue (ou squelette) pour l'affichage, et trois étapes Charger, Vérifier, Traiter (CVT).

### Une vue et 3 étapes

Prenons l'exemple d'un formulaire de contact que l'on veut gérer avec la balise `#FORMULAIRE_CONTACT`.

**Affichage**
Lorsque SPIP rencontre la balise `#FORMULAIRE_CONTACT`, il reconnait qu'il s'agit d'une balise du type `#FORMULAIRE_xxx`. Il cherche alors le squelette `formulaires/contact.html` pour afficher le formulaire.
Il n'y a aucun autre pré-requis pour l'affichage du formulaire[3], ce qui permet son intégration visuelle indépendamment des 3 fonctions ci-dessous.

**Chargement**
Avant l'affichage du squelette, SPIP appelle, si elle existe, la fonction `formulaires_contact_charger_dist()` pour fournir la liste des champs saisis dans le formulaire, avec des valeurs par défaut éventuelles. Ces champs et valeurs seront fournis au squelette `formulaires/contact.html` qui en fait donc usage.

**Vérification**
Lorsque l'internaute remplit le formulaire et clique sur le bouton de validation, SPIP appelle la fonction `formulaires_contact_verifier_dist()` pour vérifier la validité de la saisie. La fonction renvoie une liste de messages d'erreur correspondants à chaque champs erroné, ou une liste vide en cas d'absence d'erreur.

**Traitement**
Si la fonction de vérification n'a pas renvoyé d'erreur, alors SPIP appelle automatiquement la fonction de traitement du formulaire `formulaires_contact_traiter_dist()` qui pourra réaliser toutes les opérations de traitement du formulaire : envoi d'un mail, enregistrement en base de donnée...

La fonction de traitement renvoie une liste d'information, dont un message confirmant que la saisie a bien été prise en compte.

### Arguments

Les arguments passés à la balise formulaire, sous la forme `#FORMULAIRE_CONTACT{#ID_AUTEUR}` sont automatiquement transmis aux fonctions charger, verifier, traiter, dans le même ordre.

### Personnalisations

Le découpage modulaire du formulaire en un squelette d'affichage et trois fonctions Charger, Verifier et Traiter permet de personnaliser indépendamment chacune des étapes :

[3] ... mais on consultera avec intérêt la page Structure HTML des formulaires de SPIP 2

- le squelette d’un formulaire de SPIP peut être personnalisé dans le dossier `squelettes/`
- la fonction `formulaires_contact_charger_dist()` peut être personnalisée en définissant la fonction `formulaires_contact_charger()`
- la fonction `formulaires_contact_verifier_dist()` peut être personnalisée en définissant la fonction `formulaires_contact_verifier()`
- la fonction `formulaires_contact_traiter_dist()` peut être personnalisée en définissant la fonction `formulaires_contact_traiter()`

Ces fonctions sont a définir dans un fichier `contact.php`

## AJAX

La standardisation dans SPIP du formalisme CVT permet de bénéficier de fonctionnalités supplémentaires sans développement.

En particulier, pour améliorer la fluidité de la navigation et le temps de réponse perçu par l’internaute, une technique consiste à soumettre le formulaire par une requête asynchrone envoyée par le navigateur sans recharger toute la page.
Ce type de requête, souvent désignée par l’acronyme AJAX est en général lourd à développer.

SPIP et CVT simplifient radicalement la mise en place du traitement AJAX de votre formulaire.

Pour cela, il suffit d’encapsuler votre formulaire dans une div avec la classe `ajax` :

```
<div class='ajax'>
#FORMULAIRE_CONTACT
</div>
```

Lorsque l’internaute soumettra le formulaire, celui-ci sera renvoyé à SPIP par une requete asynchrone, et SPIP ne renverra que le résultat du formulaire après vérification et traitement éventuel. L’internaute aura une réponse plus rapide, car il n’aura pas rechargé toute la page, mais seulement la zone correspondant au formulaire !

## Compatibilité avec les anciens formulaires dynamiques de SPIP

Les anciens formulaires dynamiques de Spip reposant sur un squelette et une serie de fonction balise_formulaires_xxx, balise_formulaires_xxx_stat, et balise_formulaires_xxx_dyn restent fonctionnels, mais il est conseillé de les migrer progressivement vers le formalisme CVT.

Techniquement CVT est une surcouche implémentée par des fonctions génériques balise_formulaires_xxx, balise_formulaires_xxx_stat, et balise_formulaires_xxx_dyn.

A ce titre, il est possible, par exemple pour la collecte automatique d’arguments, de définir sa propre fonction balise_formulaires_contact_stat.

Sur le même sujet, on lira aussi avec bonheur la page "Formulaires CVT par l’exemple".

# 4. Multilinguisme

## Réaliser un site multilingue

Octobre 2003 — maj : Septembre 2009

### Préalable : qu'est-ce qu'un site multilingue ?

Il n'est pas question dans cet article de rédiger un tutoriel complet sur les sites multilingues : d'une part, il y a certainement plusieurs « visions » de ce qu'on appelle « multilinguisme » ; d'autre part, nous manquons toujours de recul pour pouvoir définir « la meilleure méthode ».

Vous trouverez donc ci-dessous une revue des différents outils que SPIP propose pour gérer des sites multilingues ; à vous de les utiliser, et discutons-en dans les espaces prévus à cet effet (forums, wiki, listes de discussion, etc.)

Mais avant de lire, oubliez un peu votre projet du jour, et pensez aux situations suivantes :
- un site de poésies, classées par thèmes (rubriques) ;
- un site de documentation pour, par exemple, un logiciel comme SPIP ;
- un site institutionnel en 25 langues ;
- un site corporate bilingue ;
- le site d'une association bulgare avec quelques pages en anglais.

Le site de poésie choisira plutôt ses langues article par article ; la documentation de SPIP, pour sa part, les ventile par « secteurs » (rubriques de premier niveau), et affiche les traductions disponibles pour chaque article, quand ces traductions sont disponibles. Le site institutionnel en 25 langues ne pourra sans doute pas fournir les 25 traductions en même temps, mais cherchera tout de même à conserver des arborescences parallèles ; le site corporate bilingue aura obligatoirement une traduction en face de chacun des articles, et une arborescence de rubriques en deux parties, la partie anglaise étant « clonée » sur la partie en auvergnat ; l'association bulgare affectera l'anglais à un secteur particulier de son site, le reste des secteurs étant tous en bulgare (par défaut).

### Principe

Pour pouvoir autoriser ces situations différentes, et d'autres encore, le modèle mis en place dans SPIP consiste à déterminer une langue pour chaque article, chaque rubrique et chaque brève. Dans l'espace public comme dans l'espace privé, cette langue détermine le mode de correction typographique qui est appliqué aux textes ; dans l'espace public cela détermine également la langue des élements insérés par SPIP autour de ces « objets » : dates et formulaires principalement.

Pour créer un site multilingue avec SPIP, il faut d'abord configurer le site en conséquence : dans la configuration du site, à la section « langues » bien entendu. Là vous pourrez activer la gestion du multilingue, et choisir les langues que vous utiliserez sur votre site.

### Configurer l'espace privé

Pour gérer plus facilement le site, on peut choisir dans la configuration du site avec quelle précision s'effectuera le réglage des langues, ce qui permet de masquer l'interface là où elle n'est pas nécessaire et de limiter les risques d'erreurs [1]. SPIP propose trois niveaux

d’interface différents pour choisir les langues affectées aux articles (et brèves, etc.) ; par ordre croissant de complexité :

- **Par secteur** (rubrique de premier niveau) : à chaque secteur du site correspond une langue modifiable par les administrateurs, qui concerne toutes ses sous-rubriques ainsi que les articles et les brèves qui y sont publiés ; **ce réglage devrait satisfaire les besoins de la plupart des sites multilingues tout en conservant une structure et une interface simples.**

- **Par rubrique** : de manière plus fine, avec ce réglage, on peut changer la langue pour chacune des rubriques du site, pas seulement celles de premier niveau.

- **Par article** : la langue peut être modifiée au niveau de chaque article ; ce choix est compatible avec les précédents (on peut par exemple choisir la langue par rubrique mais appliquer des exceptions de-ci de-là à certains articles) et permet toutes les finesses imaginables, mais attention à ne pas produire un site à la structure incompréhensible...

## Blocs multilingues

Certains objets, comme les auteurs ou les mots-clés, peuvent s’orthographier différemment selon qu’ils sont affectés à un article dans une langue ou dans une autre. Cependant, il serait absurde de concevoir des « traduction d’un mot-clé » ou « traduction d’un auteur », car c’est bien le même auteur qui signe les deux articles, ou le même mot-clé (même « concept ») qu’on leur attache. Ces objets n’ont donc pas de langue au sens de SPIP, mais il est tout de même possible, à l’aide des « blocs multi », de les faire s’afficher dans la langue du contexte dans lequel ils sont invoqués (pour le dire plus simplement : faire que le mot-clé « Irak » s’affiche « Iraq » quand il est affecté à un article en anglais).

Le « bloc multi » est un raccourci SPIP, dont la structure est relativement intuitive :
`<multi>chaîne 1 [xx] chaîne 2 [yy] chaîne 3 ...</multi>`

Pour reprendre l’exemple du mot-clé, son titre serait entré sous cette forme dans l’espace privé :

```
<multi>[fr]Irak [en]Iraq</multi>
```

Si un bloc multi est appelé à s’afficher dans une langue qui n’est pas prévue, c’est toujours la première partie du bloc qui s’affiche (« chaîne 1 » dans le premier exemple, « Irak » dans le second). Cela, afin de ne jamais avoir d’affichage vide [2].

*NB :* les blocs multi peuvent aussi être utilisés, selon la même structure, dans les squelettes, *cf.* Internationaliser les squelettes.

## Boucles et balises : comment faire

Une fois l’espace privé réglé aux petits oignons, passons maintenant au site public. Hé oui, même si chaque article dispose maintenant de sa propre langue judicieusement choisie (selon le mécanisme expliqué plus haut), les squelettes doivent bien pouvoir en tenir compte dans l’affichage du site.

**1.** Une bonne nouvelle pour commencer : **le multilinguisme des squelettes est pour la plus grande part totalement naturel ; il n'est pas nécessaire de faire des squelettes différents pour afficher des articles de langues différentes. Un même squelette adapte *automatiquement* son affichage à la lange courante.**

Ainsi tous les éléments affichés autour et dans un article d'une langue donnée, seront affichés dans cette langue. Cela concerne aussi bien la date de publication de l'article que les formulaires de réponse au forum, de signature d'une pétition, etc. Plus généralement : toute balise SPIP incluse dans une boucle `ARTICLES` sera affichée dans la langue de l'article (de même pour les rubriques et les brèves).

*Exemple :* si votre page d'accueil contient un sommaire affichant les dix derniers articles publiés ainsi que leur date de publication, la date des articles en vietnamien s'affichera en vietnamien, celle des articles en créole de la Réunion s'afficheront en créole de la Réunion, etc.

*Note : ce fonctionnement suppose que la langue de l'article fait l'objet d'une traduction dans SPIP. Ainsi, si un article est écrit en volapück mais que votre version de SPIP n'est pas encore traduite en volapück (nous vous invitons bien évidemment à corriger cette lacune en participant à l'effort de traduction), la date de l'article s'affichera en toutes lettres certes, mais dans une langue par défaut - le français probablement.*

### 2. Le sens de l'écriture

Si votre site contient des langues s'écrivant de gauche à droite (la plupart des langues) mais aussi des langues s'écrivant de droite à gauche (notamment l'arabe, l'hébreu ou le farsi), il faudra de petits compléments au code HTML pour que l'affichage se fasse sans accroc [3].

SPIP offre à cet effet une balise spécifique : **`#LANG_DIR`**, qui définit le sens d'écriture de la langue courante. Cette balise est utilisable comme valeur de l'attribut `dir` dans la plupart des tags HTML (cela donne donc « `ltr` » pour les langues s'écrivant de gauche à droite, et « `rtl` » pour les autres [4]).

Une boucle d'affichage du sommaire devient donc :

```
<ul>
<BOUCLE_sommaire(ARTICLES){par date}{inverse}{0,10}>
  <li dir="#LANG_DIR">[(#DATE|affdate)]:
  <a href="#URL_ARTICLE">#TITRE</a></li>
</BOUCLE_sommaire>
</ul>
```

Si la mise en page repose sur des éléments alignés à droite ou à gauche, ceux-ci devront être inversés pour les langues écrites de la droite vers la gauche : on peut tout de suite penser à remplacer *tous* [5] les éléments du squelette marqués `left` ou `right` par les balises `#LANG_LEFT` et `#LANG_RIGHT`. Pour ce qui est de la définition de la page elle-même, il est alors judicieux de commencer par indiquer la langue de l'élément demandé, et la direction générale de la page :

```
<html dir="#LANG_DIR" lang="#LANG">
<head>
...
</head>
<body>
...
</body>
</html>
```

### 3. Les liens de traduction

SPIP propose un système de traduction entre articles : on peut spécifier quelles sont les différentes traductions d'un article (note : ces traductions sont elles-mêmes des articles à part entière). Le critère **`{traduction}`** permet alors, dans une boucle `ARTICLES`, de récupérer toutes les versions d'un même article.

Par exemple, pour afficher toutes les traductions de l'article courant :

```
<BOUCLE_traductions(ARTICLES){traduction}{exclus}>
[<a href="#URL_ARTICLE" rel="alternate"
hreflang="#LANG">(#LANG|traduire_nom_langue)</a>]
</BOUCLE_traductions>
```

Notons le critère `{exclus}`, qui permet de ne pas afficher la version courante, et le filtre `|traduire_nom_langue` qui fournit le nom véritable de la langue à partir de son code informatique (cela permet d'afficher « français » au lieu de « `fr` », « English » au lieu de « `en` », etc.).

- Un critère complémentaire `{origine_traduction}` (pour les plus acharnés) permet de sélectionner uniquement la « version originale » de l'article courant.

Une page du wiki de spip-contrib rassemble des exemples de boucles utilisant ces critères : http://www.spip-contrib.net/-Carnet-Multilinguisme-

### 4. Éléments supplémentaires

SPIP introduit d'autres éléments permettant de fabriquer des sites multilingues :

- le critère `{lang_select}` sert à forcer la sélection de la langue pour la boucle (AUTEURS), qui normalement ne le fait pas (à l'inverse, le critère `{lang_select=non}` permet de dire aux boucles (ARTICLES), (RUBRIQUES) ou (BREVES) de ne pas sélectionner la langue).
- la variable de personnalisation `$forcer_lang` indique à SPIP qu'il doit vérifier si le visiteur dispose d'un cookie de langue, et si oui le renvoyer vers la page correspondante. C'est ce que fait la page de connexion à l'espace privé livrée en standard avec SPIP.
- les balises `#MENU_LANG` (et `#MENU_LANG_ECRIRE`) affichent un menu de langue qui permet au visiteur de choisir « cette page en ... ». La première balise affiche la liste des

langues du site ; la seconde la liste des langues de l'espace privé (elle est utilisée sur la page de connexion à l'espace privé).

- enfin, les critères optionnels permettent d'utiliser une même boucle (en fait, un même squelette) pour afficher soit tous les articles du site dans toutes les langues, soit seulement les articles dans la langue passée dans l'URL. Ça peut être utile dans les backend, par exemple, ou dans les boucles de recherche :

```
<BOUCLE_recents(ARTICLES){lang?}{par date}{inverse}{0,10}>
<BOUCLE_recherche(ARTICLES){lang?}{recherche}{par points}{inverse}{0,10}>
```

## Des squelettes internationaux pour un site massivement multilingue

Ce qui précède nous a permis de rendre multilingue la partie proprement SPIP de notre squelette : tout ce qui est issu des boucles s'affiche dans le bon sens, avec la bonne typographie, et les éléments produits par SPIP (formulaires, dates...) sont dans la langue demandée.

Pour un site présentant un nombre modeste de langues (bilingue par exemple), ou pour lequel il existe une langue principale et quelques langues annexes, on pourrait en rester là. Les textes figés présents dans les squelettes, c'est-à-dire les mentions écrites directement dans le HTML comme « Plan du site », « Espace de rédaction », « Répondre à ce message »... peuvent dans certains cas rester dans une seule langue ; ou alors, un site bilingue pourra utiliser des squelettes séparés pour chacune des deux langues.

Cependant, si vous voulez réaliser et gérer efficacement un site présentant beaucoup de langues à part entière, il devient illusoire de maintenir des squelettes séparés, ou d'imposer une navigation dans une langue unique (même en anglais ou en espéranto...). Pour réaliser un jeu de squelettes unique fonctionnant dans toutes les langues, il faut internationaliser les squelettes afin de modifier les textes indépendamment du code HTML qui les contient (qui reste, lui, figé d'une langue à l'autre). Cette tâche nécessite de mettre un peu « les mains dans le cambouis » et fait l'objet d'un article séparé.

## Détails annexes

- Les raccourcis typographiques `<code>` et `<cadre>` produisent toujours un texte écrit de gauche à droite, même si la langue de l'article s'écrit normalement de droite à gauche. En effet ces deux raccourcis sont destinés à afficher du code ou des données informatiques, qui sont à peu près toujours écrits de gauche à droite (et, la plupart du temps, en caractères occidentaux).
- Toujours en ce qui concerne le sens d'écriture, notons que les attributs `left` et `right` du HTML sont aussi souvent présents dans les feuilles de style. Cela veut dire que vous devrez peut-être inclure la partie correspondante de la feuille de style dans vos squelettes (pour utiliser les balises `#LANG_LEFT` et `#LANG_RIGHT`) plutôt que de la placer dans un fichier séparé.

Voici un récapitulatif du comportement des balises SPIP liées au sens de l'écriture :

| Langue | #LANG_LEFT | #LANG_RIGHT | #LANG_DIR |
|---|---|---|---|
| langues écrites de gauche à droite | left | right | ltr |
| arabe,farsi,hébreu... | right | left | rtl |

- Un filtre `|direction_css` permet d'« inverser » un fichier CSS pour les langues s'écrivant de droite à gauche. Si la feuille de style CSS à inverser s'appelle par exemple *style.css*, ce filtre utilise (dans le cas où la langue courante s'écrit de droite à gauche) une éventuelle feuille *style_rtl.css* ; si celle-ci n'existe pas, il crée automatiquement une feuille RTL en remplaçant toutes les occurrences de *left* par *right* et vice-versa (et la stocke dans le répertoire `IMG/cache-css/`). Il s'applique le plus souvent sur une balise `#CHEMIN`, de cette façon : `[(#CHEMIN{style.css}|direction_css)]`.

## Notes

[1] On précise ici que dans la configuration livrée d'origine, SPIP reste monolingue, afin de ne pas compliquer du tout l'interface.

[2] Si l'on veut au contraire un affichage vide, il faut créer explicitement une première partie vide avec un nom de langue quelconque.

[3] Théoriquement le HTML devrait régler tous ces détails automatiquement, mais le résultat n'est pas toujours à la hauteur, surtout lors des passages à la ligne ou si l'on mélange des langues utilisant un sens d'écriture différent.

[4] Malheureusement, les instances de normalisation semblent pour l'instant ignorer le boustrophédon, ce qui empêche son utilisation en HTML.

[5] Tous, ou presque tous, nous vous laissons découvrir si votre mise en page présente des cas particuliers...

## Internationaliser les squelettes

Octobre 2003 — maj : novembre 2010

### Pourquoi créer des squelettes multilingues ?

Dès la mise en ligne de documents, SPIP adapte certaines informations « automatiques » dans la langue désirée. Notamment les dates sont affichées dans la langue du site, ou d'un article, ou d'une rubrique, les formulaires sont affichés dans la langue correspondante (par exemple l'interface pour poster des messages de forums)... Tout cela est d'ores et déjà traduit par SPIP.

Ce n'est cependant pas suffisant : les webmestres insèrent dans leurs squelettes un certain nombre d'informations, décrivant notamment les principes de navigation dans le site. Il est nécessaire, par exemple, d'afficher des textes du style « Plan du site », « Répondre à cet article », « Articles du même auteur », « Dans la même rubrique »... Pour un site dans une seule langue, ces différents éléments sont faciles à insérer : on les insère tels quels dans le code HTML des squelettes. Le problème apparaît lorsque le site est multilingue : sous un article en français, on veut afficher *« Répondre à cet article »*, mais sous un article en anglais, on a besoin d'afficher un autre texte (*« Comment on this article »*).

SPIP propose trois méthodes pour gérer ces éléments de texte différents selon les langues :

- (1)une méthode consistant à stocker les éléments de texte des squelettes dans des *fichiers de langue* (un fichier différent par langue utilisée sur le site), séparés des squelettes ; un squelette *unique* (par exemple `article.html` appelant, selon des codes définis par le webmestre, ces éléments de texte en fonction de la langue utilisée ; de cette façon, un même squelette `article.html` affichera automatiquement le texte « Répondre à cet article » ou « Comment on this document » en fonction de la langue de l'article. Cette méthode est vivement conseillée, elle offre le plus de souplesse, elle facilite les mises à jour du site (on travaille avec un squelette unique qui gère automatiquement plusieurs langues), et à terme des outils seront ajoutés à SPIP facilitant le travail collectif de traduction de l'interface de votre site (plusieurs administrateurs, parlant chacun une langue différente, pourront *traduire* l'interface d'un même site depuis l'espace privé, sans avoir besoin d'intervenir sur les fichiers des squelettes) ;

- (2)une méthode plus rapidement accessible, techniquement très simple, reposant sur la création de fichiers de squelettes différents pour chaque langue. Dans cette méthode, on fabrique un fichier `article.html` pour gérer les articles en français, et un fichier `article.en.html` pour les articles en anglais (note : `article.html` gère en réalité toutes les langues sauf l'anglais). Cette méthode est déconseillée si le site utilise de nombreuses langues et/ou si on utilise des squelettes distincts selon les rubriques. Par ailleurs, elle est dans tous les cas avantageusement remplacée par la méthode 3 décrite ci-dessous :
- (3)la méthode des « blocs multilingues », introduite par [SPIP 1.7.2], fonctionne aussi bien dans les contenus que dans les squelettes. Il suffit de mettre dans le squelette la construction suivante :

```
<multi>
[fr] Répondre à cet article
[en] Comment on this article
</multi>
```

et la phrase s'affichera dans la langue voulue. Si ce système est très souple, il atteint toutefois ses limites dès que le nombre de langues est important, et que l'on souhaite que différents traducteurs interviennent dans le site (il faut en effet qu'ils puissent modifier le squelette, ce que la méthode des fichiers de langue permet d'éviter).

## 1. Méthode des fichiers de langue

Le principe des fichiers de langue consiste à insérer dans un squelette unique un *code*, lequel correspondra dans chaque langue à un élément de texte (une « chaîne ») ; entre les différentes langues le code ne varie pas, mais le texte est traduit.

Par exemple, nous pouvons décider que le *code* **`telechargement`** correspond :

- en français, à la chaîne *« télécharger ce fichier »*,
- en anglais, à la chaîne *« download this file »*,
- en espagnol, à la chaîne *« descargar este archivo »*,
- etc.

Dans le fichier de squelette des articles (un seul fichier gérera toutes les langues), `article.html`, il suffit d'insérer le code (notez la syntaxe) :

```
<:telechargement:>
```

Lors de l'affichage d'un article, ce *code* sera remplacé par sa traduction dans la langue de l'article.

Par exemple, dans le squelette `article.html`, nous insérons dans la boucle affichant les documents associés à l'article, le code suivant :

```
<a href="#URL_DOCUMENT"><:telechargement:></a>
```

Si l'article en question est en français, cela produira :

```
<a href="/IMG/jpg/mondocument.jpg">télécharger ce fichier</a>
```

si l'article est en anglais :

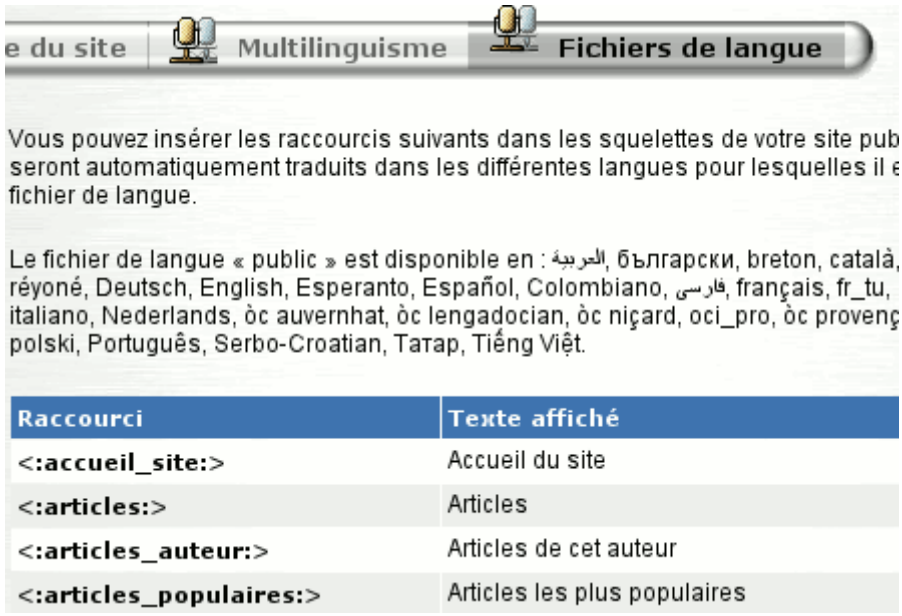
```
<a href="/IMG/jpg/mondocument.jpg">download this file</a>
```

et ainsi de suite. Un unique squelette, contenant un unique code, affiche un texte traduit dans toutes les langues utilisées sur le site.

- **Utiliser des textes déjà traduits**

Pour faciliter le travail des webmestres, SPIP fournit un ensemble de chaînes déjà traduites (par les traducteurs de SPIP). En utilisant ces chaînes, correspondant à des éléments de texte fréquemment utilisés sur des sites Web, le webmestre peut rapidement réaliser une interface fonctionnant dans différentes langues, mêmes celles qu'il ne parle pas lui-même.

Vous pouvez lister les chaînes disponibles depuis l'espace privé : allez dans la section « Gestion des langues » de la partie « Administration du site », puis cliquez sur l'onglet « Fichiers de langues ». Vous n'avez plus qu'à y piocher les codes de votre choix pour la réalisation de vos squelettes.




**Les fichiers de langue dans l'espace privé**

*Exemple :* un webmestre veut réaliser l'interface pour un site en français, en espagnol et en arabe, mais il ne parle pas lui-même l'espagnol et l'arabe. En insérant dans ses squelettes les codes livrés avec SPIP, il n'a pas à se soucier d'obtenir des traductions en espagnol et en arabe, car le travail de traduction a déjà été effectué en amont par les traducteurs de SPIP ; ainsi, en mettant au point l'interface en français avec les codes fournis par SPIP, il sait que ses pages s'afficheront immédiatement en espagnol et en arabe.

Si par la suite on ajoute des articles en polonais, ils seront immédiatement affichés avec les éléments de texte traduits en polonais, sans que le webmestre ait à intervenir à nouveau.

Autre avantage de cette méthode : elle facilite la création de squelettes « à distribuer » immédiatement multilingues. Des squelettes réalisés selon cette méthode seront immédiatement utilisables dans toutes les langues dans lesquelles sera traduit SPIP.

D'un point de vue technique, les éléments de texte fournis en standard avec SPIP sont stockés dans les fichiers de langue « public » :

- `/ecrire/lang/public_fr.php` contient les chaînes en français,
- `/ecrire/lang/public_en.php` en anglais
- etc.

- **Créer ses propres codes**

Il est de plus possible de créer ses propres codes, correspondant à des chaînes que l'on désire ajouter soi-même.

Il s’agit alors de créer des fichiers de langue personnels, sur le modèle des fichiers `public...`. Pour créer ses propres fichiers, on installera, dans un répertoire `squelettes/lang` à créer le cas échéant.

- `local_fr.php` pour définir les chaînes en français,
- `local_en.php` en anglais,
- ...

Par exemple, on pourra créer les chaînes suivantes :

- `telechargement` pour afficher « Télécharger la dernière version »,
- `quoideneuf` pour afficher « Modifications récentes ».

Selon cette méthode, on insère dans les squelettes les codes `<:telechargement:>` et `<:quoideneuf:>`, ils seront ensuite affichés avec les traductions correspondantes, telles qu’elles sont définies dans les fichiers `local_...php`.

Notons que les codes sont arbitraires : c’est vous qui les choisissez. Nous recommandons bien évidemment de choisir des codes qui vous permettent de les retenir facilement (plutôt que des numéros par exemple). Comme souvent avec les codes informatiques, il est préférable de n’utiliser que des lettres de l’alphabet latin, et sans accents...

Les *fichiers de langue* contiennent les différentes traductions des codes que vous utiliserez ; ce sont des fichiers PHP contenant chacun un tableau associant aux codes les chaînes correspondantes dans chaque langue.

Ils contiendront par exemple :

- *Version française :*

```
<?php
$GLOBALS[$GLOBALS['idx_lang']] = array(
      'telechargement' => 'Télécharger la
dernière version',
      'quoideneuf' => 'Modifications récentes'
);
?>
```

- *Version catalane :*

```
<?php
$GLOBALS[$GLOBALS['idx_lang']] = array(
      'telechargement' => 'Descarregar la darrera versió',
      'quoideneuf' => 'Modificacions recents'
);
?>
```

La construction est la suivante :

- au début du fichier :

  ```
  <?php
  $GLOBALS[$GLOBALS['idx_lang']] = array(
  ```

- à la fin du fichier :

  ```
  );
  ?>
  ```

- la partie qu'il faut enrichir soit-même consiste en plusieurs lignes de *définitions*, sur le modèle :

  ```
  'code' => 'La chaîne à afficher',
  ```

**N.B.** Chaque ligne de définition se termine par une virgule, sauf la dernière ligne.

**N.B.2.** Le texte de la chaîne à traduire doit être convertie en codes HTML (les caractères accentués, par exemple, sont convertis en leur équivalent HTML, du type `&eacute;`).

Les apostrophes à l'intérieur de la chaîne doivent être *échappées*, c'est-à-dire précédées d'un antislash. Par exemple, la chaîne « sur l'internet » doit être inscrit : `sur l\'internet`.

*Note :* à terme, il est prévu d'inclure un outil permettant de gérer et créer ses propres fichiers de langue sans avoir à modifier « à la main » des fichiers PHP. Un tel outil facilitera de plus l'utilisation de caractères « spéciaux » (caractères accentués, caractères dans des alphabets non occidentaux, échappement des apostrophes...), ainsi que la collaboration de plusieurs personnes au processus de traduction de l'interface du site public.

Pour l'instant, l'outil permettant de gérer la traduction des chaînes de texte n'est pas livré directement avec SPIP, et son utilisation très généraliste (nous l'utilisons pour traduire toute l'interface de SPIP, et pas seulement des fichiers de langue de type `local...php`) le rend un peu complexe par rapport à cette tâche. Ce programme, **trad-lang**, qui nous sert à traduire le logiciel SPIP, le site spip.net, etc., est lui-même disponible sous licence GNU/GPL, mais il n'est pas intégré en standard à SPIP.

Vous pourrez le télécharger (http://eledo.com/article17.html), pour l'utiliser pour votre site ou pour d'autres projets de logiciels.

Si vous l'améliorez, ou avez des idées pour le transformer, venez en discuter sur la liste des traducteurs de SPIP, spip-trad (http://listes.rezo.net/mailman/listinfo/spip-trad).

## 2. Des squelettes séparés pour chaque langue

La seconde méthode, plus accessible aux webmestres débutants, consiste à créer des squelettes différents pour chaque langue. Un peu sur le même principe qui consiste à créer des squelettes spécifiques pour différentes rubriques pour obtenir des interfaces graphiques différentes.

Nous voulons réaliser un site en français (langue par défaut), en anglais et en espagnol. Nous réalisons trois fichiers de squelette différents :

- `article.html` pour le français (en réalité, pour toutes les langues qui n'ont pas de fichier de langue spécifique ci-après),
- `article.en.html` pour l'anglais,
- `article.es.html` pour l'espagnol.

(Note : si on publie un article en allemand, alors qu'il n'existe pas de squelette `article.de.html` sur notre site, c'est le squelette `article.html` qui sera utilisé.)

*Important :* pour que les squelettes « par langue », définis par l'ajout d'un `.lang` à la fin du nom, soient pris en compte, il faut obligatoirement qu'il existe la version « par défaut » sur le site.

Ici, si `article.html` n'existe pas (on aurait préféré nommer directement un `article.fr.html`), les fichiers anglais et espagnol ne seront pas pris en compte.

On peut combiner cela avec le nommage « par rubriques », et l'ordre suivant sera pris en compte :

- `article=8.es.html` (le squelette pour les articles en espagnol de la rubrique 8, mais pas ses sous-rubriques),
- `article=8.html` (le squelette pour les articles de la rubrique 8, mais pas ses sous-rubriques),
- `article-2.es.html` (le squelette pour les articles en espagnol de la rubrique 2 et ses sous-rubriques),
- `article-2.html` (le squelette pour les articles de la rubrique 2 et ses sous-rubriques),
- `article.es.html` (le squelette pour les articles en espagnol),
- `article.html` (le squelette pour les articles),
- `article-dist.html` (le squelette pour les articles livré avec SPIP).

*Note :* sauf pour quelques exceptions, il faut utiliser ici les codes de langues à deux lettres normalisés par l'ISO, comme « `es` ». On en trouvera notamment la liste sur le site de la Bibliothèque du Congrès des Etats-Unis (eh oui ! http://www.loc.gov/standards/iso639-2/langcodes.html). Pour s'assurer d'un maximum de compatibilité, il faut utiliser en priorité les codes ISO à deux lettres (« iso 639-1 »), quand ils existent, et pour une désignation plus spécialisée d'une langue dans sa famille linguistique les codes ISO à trois lettres (« iso 639-2 T »). Par exemple, l'allemand sera désigné comme « `de` ». Mais l'ISO ne prend en compte que 182 langues sur les 5 000 à 7 000 langues parlées dans le monde ; pour les langues non encore répertoriées, on peut s'inspirer du répertoire proposé par le site ethnologue.com (http://www.ethnologue.com/iso639/codes.asp); pour en savoir plus, rendez-vous sur la liste spip-trad (http://listes.rezo.net/mailman/listinfo/spip-trad).
*Se simplifier la vie.* Une des méthodes de structuration très simple qu'autorise SPIP pour gérer un site multilingue consiste à associer directement les langues aux rubriques (et non article par article). Ainsi, dans le cas où les articles d'une même langue sont regroupés dans une même rubrique (voire par secteurs), on peut se contenter de créer les squelettes spécifiques *par rubrique*, sans utiliser alors les noms de fichiers par langues.

De cette façon, si, tous les articles en espagnol sont regroupés dans la rubrique 8 (et ses sous-rubriques), on peut se contenter de nommer le fichier adapté à l'espagnol `rubrique-8.html` plutôt que `rubrique.es.html`.

On devine les problèmes qui peuvent se poser avec cette méthode :

- le fichier `article-2.html` *doit* exister si on veut que `article-2.es.html` puisse être sélectionné ;

- on ne peut pas décider, à l'inverse, que `article.es.html` doit être utilisé à la place d'`article-2.html` si `article-2.es.html` n'existe pas : la sélection par rubriques a toujours la priorité sur la sélection par langues ;
- une correction de mise en page dans un squelette implique de faire la même correction dans les autres versions,
- on doit pouvoir insérer soi-même des éléments de texte dans les fichiers des squelettes, alors qu'on ne comprend pas forcément la langue (imaginez-vous créer ainsi les squelettes en arabe alors que vous parlez un peu le français de l'ouest et assez mal l'occitan du nord...).

Cette méthode est donc destinée à travailler rapidement sur des sites peu complexes (peu ou pas de squelettes spécifiques aux rubriques) et/ou comportant peu de langues différentes. Dès que votre projet multilingue devient un peu plus ambitieux, il est vivement conseillé de travailler avec la méthode des fichiers de langue.

## 3. Les blocs multilingues

Les blocs multi définis dans l'article Réaliser un site multilingue fonctionnent aussi bien dans le texte des auteurs ou mots-clés que dans les squelettes. Attention, ces blocs ne gèrent *que* du texte, et pas des boucles SPIP !

Pour gérer rapidement un site multilingue, c'est sans doute, dans un premier temps, la meilleure méthode, à la fois accessible et efficace ; ensuite, une fois votre site stabilisé, si vous avez besoin d'affiner vos squelettes (par exemple, pour les ouvrir à plus de langues ; ou pour les distribuer sous forme de contrib ; ou encore pour les « professionnaliser »), il faudra transformer vos blocs multi en fichiers de langue.

# 5. Rechercher avec SPIP

**SPIP intègre un moteur de recherche interne, qui permet d'effectuer des recherches sur différents types d'informations présentes dans la base de données : les articles, les rubriques, les brèves, les mots-clés, etc.**

## Comment fonctionne le moteur de recherche de SPIP ?

Mai 2002 — maj : Décembre 2007

Le moteur de recherche intégré à SPIP est très simple d'utilisation, et cependant relativement puissant. Même si la plupart des utilisateurs n'ont aucune raison de se demander « comment ça marche ? », nombreux sont les courriers qui demandent des précisions sur son fonctionnement...

Voici les principes sur lesquels repose le moteur de SPIP.

Afin d'être rapide et efficace (c'est-à-dire pour qu'il donne des réponses pertinentes), le moteur de SPIP utilise un système d'**indexation** des contenus. L'indexation consiste à analyser tous les textes contenus dans SPIP, à en extraire tous les mots, et à sauvegarder pour chaque mot l'endroit où il apparaît.

Comme l'index dans un livre vous présente les mots importants du livre, suivis des numéros des pages où les retrouver, l'index du moteur de recherche fait correspondre à chaque mot utilisé sur le site le numéro de l'article, de la brève... où le retrouver.

Ensuite, lorsqu'on utilise le moteur pour effectuer une recherche, SPIP n'a plus besoin de consulter l'ensemble des textes du site, mais simplement d'aller consulter l'index pour savoir où se trouvent ces mots. On gagne ainsi énormément de temps (et plus le site est gros, plus on gagne de temps).

### L'indexation

Le principe est donc le suivant : prendre un texte (plus ou moins long), en extraire tous les mots, et noter chacun de ces mots dans une base de données, en faisant correspondre ce mot à l'endroit où il se trouve.

Par exemple, notre site a trois articles, dont les textes (très courts) sont :
- *article 1* : « Le petit chat est mort de froid et de faim. »
- *article 2* : « Le gros chat est rentré à la maison. »
- *article 3* : « La maison résiste au froid. »

Nous allons extraire les mots de chaque article, et enregistrer pour chaque mot à quel article il correspond (nous ne prendrons que les mots de plus de trois lettres, nous expliquerons plus loin pourquoi) :
- *petit* : 1
- *chat* : 1, 2
- *mort* : 1
- *froid* : 1, 3
- *faim* : 1
- *gros* : 1
- *rentré* : 2

- *maison* : 2, 3
- *résiste* : 3

Et ainsi de suite, en considérant que notre site est certainement beaucoup plus gros, et les articles beaucoup plus long.

- Si l'on recherche le mot *chat* :
  une solution sans indexation consisterait à relire tous les articles, pour y trouver le mot *chat* ; sur un gros site, même pour un ordinateur, cela prend beaucoup de temps ;
- puisque nous avons un index, il suffit de consulter l'entrée *chat* : on sait immédiatement qu'il se trouve dans les articles 1 et 2.

**La pondération**

À l'indexation s'ajoute un deuxième principe : la pondération. Il s'agit d'essayer de rendre le moteur plus pertinent. Par exemple, si un mot apparaît dans le titre d'un article, et dans le corps du texte d'un autre article, on considère que si l'on recherche ce mot, il faut en premier indiquer celui où il apparaît dans le titre. De plus, si un mot apparaît 25 fois dans un article, et seulement deux fois dans un autre, on veut afficher en premier l'article où le mot est le plus fréquent.

On voit que l'indexation simple ne suffit pas. Si on recherche *chat*, on trouvera les articles où il apparaît, mais sans pouvoir ensuite classer ces articles entre eux (selon que le mot chat apparaît une fois ou 20 fois, ou s'il se trouve dans le titre ou seulement dans le texte...).

SPIP va donc calculer une pondération pour chaque mot dans chaque article. C'est-à-dire donner des points à ce mot en fonction de l'endroit où il se trouve, et du nombre de fois où il apparaît :

| | |
|---|---|
| dans le titre | 8 points |
| dans le soustitre | 5 points |
| dans le surtitre | 5 points |
| dans le descriptif | 4 points |
| dans le chapo | 3 points |
| dans le texte | 1 point |
| dans le post-scriptum | 1 point |

Si le mot apparaît plusieurs fois, on additionne les occurences.

Par exemple, si dans un article, le mot *chat* apparaît :
- une fois dans le titre : 8 points
- deux fois dans le descriptif : 2 fois 4 = 8 points
- six fois dans le texte : 6 fois 1 = 6 points
- *total* : 8 + 8 + 6 = 22 points.

Le mot *chat*, dans l'index, est donc ainsi enregistré :
- *chat*, dans l'article numéro X, 22 points ;
- *chat*, dans l'article numéro Y, 15 points ;
- ...

Si l'on recherche le mot *chat*, grâce à l'index, on saura donc qu'il se trouve dans les articles X et Y, et de plus on peut classer ces articles entre eux : 22 points dans X, 15 points dans Y, donc on considère que l'article X répond mieux à la recherche.

*Les mots-clés :* beaucoup d'utilisateurs confondent les mots-clés et l'indexation. Les mots-clés n'ont, par nature, aucun rapport avec l'indexation : quand SPIP effectue une recherche, il ne recherche pas dans les mots-clés associés à des articles (ce serait très limité), il recherche dans l'index qu'il a fabriqué à partir du texte exact des articles. Donc, dans le principe même de l'indexation, les mots-clés n'interviennent pas du tout.

En revanche, les mots-clés sont tout de même utilisés pour fabriquer la pondération des articles. Si un mot-clé est associé à un article, il entre alors dans l'indexation de l'article lui-même, avec une forte pondération (12 points pour le nom du mot-clé, 3 points pour son descriptif). Ainsi, si notre article Y (15 points en prenant simplement compte son contenu propre) est associé au mot-clé *chat* (parce qu'on indique par ce mot que c'est le sujet de cet article), il faut ajouter à l'indexation de cet article, pour le mot *chat*, les 12 points du mot-clé : total 27 points. Dans notre recherche sur *chat*, l'article Y devance désormais l'article X.

Notons enfin que tous les éléments de contenu de SPIP font ainsi l'objet d'une indexation : les articles, les brèves, les rubriques, les auteur, les mots-clés, les sites référencés (si le site est une syndication, les titres des articles récupérés sur ce site entrent également dans l'indexation).

**Où est-ce stocké ?**

Les données de l'indexation sont directement stockées dans la base de données. Cette partie est un peu technique, aussi je ne m'étendrai pas longtemps sur le sujet.

Sachez simplement qu'il y a plusieurs index (listes des mots), correspondant chacun à un type de contenu (un index pour les articles, un index pour les rubriques, un index pour les brèves...). De plus, contrairement à l'explication ci-dessus, où les entrées de l'index sont des mots, dans SPIP les entrées des index sont des nombres calculés à partir de ces mots (des *hachages*) ; une autre table de la base de données stockant par ailleurs les correspondance entre les véritables mots et ces nombres ; cette méthode permet d'accélérer les recherches dans les index (il est plus rapide pour le logiciel de rechercher des nombres plutôt que des mots).

Notez surtout que la taille des index est très importante. Sur uZine, par exemple, la partie de la base de données consacrée au stockage des articles pèse 9,7 Mo. La partie qui stocke l'index correspond aux articles pèse 10,5 Mo. Et la partie qui stocke la correspondance entre les mots et leur traduction en nombres 4,1 Mo. Ainsi, pour un site dont les articles occupent 9,7 Mo, l'indexation de ces articles représente, à elle seule, près de 14,6 Mo. L'espace nécessaire à ces articles et à la recherche a donc plus que doublé la taille occupée par le site. C'est l'une des raisons, notamment, pour lesquelles on peut préférer désactiver le moteur de recherche, si l'on a d'importantes limitations en matière d'hébergement.

**Quels mots sont indexés ?**

Nous l'avons vu, tous les mots de tous les éléments de contenu du site sont extraits, puis analysés (pour pondération). Cependant, SPIP ne va pas retenir tous les mots.

- Les codes HTML sont exclus de l'indexation. Cela est évidemment nécessaire pour obtenir des recherches « propres »...

- Les mots de moins de 4 lettres ne sont pas retenus (en réalité, de moins de 3 lettres, mais les mots de 3 lettres ne sont pour l'instant pas exploités réellement lors des recherches). Ce point soulève beaucoup de questions de la part des utilisateurs...

Le problème est d'obtenir des résultats aussi pertinents que possible. Il faut donc privilégier, dans les recherches, les mots réellement importants dans le sens de la recherche. Par exemple, si l'on recherche les mots *le chat*, le mot important est *chat*, et non *le*...

Reprenons notre premier exemple (avec trois articles constitués chacun d'un phrase). Si l'on avait indexé tous les mots, nous aurions notamment les mots :
- *le* : 1, 2
- *est* : 1, 2
- ...

En recherchant les mot *le froid est dangereux*, nous trouverions les entrées :
- *le* : 1, 2
- *froid* : 1,3
- *est* : 1, 2
- *dangereux* : nulle part.

En ajoutant les résultats de ces mots pour chaque article (en réalité, la pondération de chaque article, mais considérons pour notre exemple que chaque mot a une pondération d'un seul point), on obtiendrait :
- article 1 : 3 mots
- article 2 : 2 mots
- article 3 : 1 mot.

Le classement mettrait donc en tête l'article 1, puis l'article 2, puis l'article 3. Or, l'article 2 ne parle pas de froid, contrairement à l'article 3. À cause de l'utilisation des mots sans importance pour le sens (*le*, *est*), le classement est faussé.

D'où le besoin, dans l'indexation, de ne pas tenir compte des mots qui n'entrent pas dans le sens de la recherche. La méthode la plus propre consiste à fournir au système une liste de mots à ne pas indexer ; cependant, cela nécessite un travail énorme, c'est-à-dire la constitution de dictionnaires d'exclusion (et cela dans toutes les langues)... Donc, plus simplement, dans SPIP nous choisissons de considérer que les mots de trois lettres et moins ne sont pas « importants » ; évidemment, il y a de la casse, puisque des mots comme *Val*, *mer*, *sud*... ne sont plus pris en compte ; c'est donc un compromis, qui se juge sur l'efficacité des recherches (qui sont globalement de bonne qualité).

N.B. Depuis la version 1.6 de SPIP, les sigles de deux lettres et plus, y compris ceux contenant des chiffres (G8, CNT, PHP, ONU...), sont indexés. Un sigle est un mot ne comprenant aucune minuscule.

**Quand a lieu l'indexation ?**

L'indexation a lieu à trois moments différents :

- lorsque vous validez un article, celui-ci est immédiatement indexé ;

- lorsque vous modifiez un article déjà publié, il est à nouveau indexé ;

- lors de la visite du site public, si un élément accessible au public n'est pas indexé (par exemple, lorsque vous venez d'effacer les données d'indexation depuis l'espace privé, ou lorsque vous venez de rétablir une sauvegarde de votre site - les index ne sont pas sauvegardés -, ou si vous avez activé le moteur de recherche après avoir déjà publié des articles), alors il est indexé (en tâche de fond).

Retenez que l'opération d'indexation est relativement lourde : elle demande de nombreux calculs (calculs peu complexes, mais effectués sur tous les mots de l'article), et provoque de très nombreux appels à la base de données. Là aussi, chez un hébergeur très lent (vraiment très lent !), il peut être préférable de désactiver le moteur de recherche.

Retenez également que, si vous activez le moteur après avoir déjà publié des articles, ceux-ci ne sont pas immédiatement indexés : ce seront les visites sur le site public qui provoqueront leur indexation. Sur un gros site, cela peut prendre un certain temps.

## La recherche

Puisque tous les documents sont indexés, on peut désormais effectuer des recherches.

*Si vous recherchez un seul mot...*

SPIP va consulter l'index, et trouver l'entrée correspond à ce mot. Pour le mot *chat*, nous avions trouvé les articles X et Y. SPIP va de plus récupérer le nombre de points attribués à ce mot pour chaque article (22 points dans X, et 27 points pour Y). On peut donc classer nos résultats : l'article Y, puis l'article X.

*Si vous recherchez plusieurs mots...*

SPIP n'autorise pas les constructions du type « ET », « OU », il ne fonctionne pas de cette manière.

Lorsque vous recherches plusieurs mots, SPIP va effectuer l'opération de recherche pour chaque mot, récupérer les points de chaque article (ou brève, ou rubrique, etc.), et ajouter ces points.

Recherchons par exemple les mots *chat*, *gros*, *maison*. On obtient les résultats suivants pour chaque mot :
- *chat* : article X (22 points), article Y (27 points),
- *gros* : article X (12 points), article Y (2 points), article Z (5 points),
- *maison* : article Y (3 points), article Z (17 points).

SPIP fait l'addition de tous ces points pour chaque article :
- article X : 22 + 12 = 34 points,
- article Y : 27 + 2 + 3 = 32 points,
- article Z : 5 + 17 = 22 points.

Le classement des articles, pour la recherche *chat, gros, maison*, est : article X, puis article Y, puis article Z. (Dans les squelettes, on peut d'ailleurs demander l'affichage des points de chaque résultat grâce à la balise `#POINTS`, voir l'article « Les boucles et balises de recherche ».)

Ca n’est donc pas une recherche de type « ET » ni « OU », c’est une addition de points. Et à l’usage, cela se montre plutôt efficace (on trouve ce que l’on cherche, ce qui est bien le but d’un moteur...).

## Le moteur de recherche

Juin 2001 — maj : Décembre 2007

SPIP intègre un moteur de recherche, désactivé par défaut. Ce moteur, lorsqu'il est activé par un administrateur dans la page de configuration, permet d'effectuer des recherches sur différents types d'informations présentes dans la base de données : les articles, les rubriques, les brèves, les mots-clés et les auteurs. Les fils de discussion des forums (threads) et les signatures de pétitions sont également indexés.

### Principe

Il y a deux grandes façons de faire un moteur de recherche. La première est de chercher tout bêtement dans le type de stockage existant (fichiers HTML, base de données... selon le type de site). Cette méthode est très lente car le type de stockage n'est pas prévu à cet effet.

La seconde méthode, qui a été choisie pour SPIP (et qui est aussi celle de tous les moteurs professionnels), est d'établir un mode de stockage spécifique aux besoins de la recherche. Par exemple, le score de chaque mots d'un article peut être stocké directement afin d'être facilement retrouvé, et d'avoir le score total d'une recherche par une simple addition. L'avantage est que la recherche est très rapide : presque aussi rapide que n'importe quel calcul de page. L'inconvénient est qu'il faut une phase de construction du dit stockage des informations : cela s'appelle l'indexation. L'indexation a un coût en termes de ressources (temps de calcul et espace disque), et elle introduit également un léger décalage temporel entre l'ajout ou la modification d'un contenu, et la répercussion de cet ajout ou de cette modification sur les résultats de recherche.

D'autre part, dans le cas de SPIP, nous sommes obligés d'utiliser PHP et MySQL comme pour le reste du logiciel, ce qui ne permet pas de réaliser un moteur très performant, en termes de rapidité, mais aussi de pertinence ou d'enrichissements divers (indexation de documents extérieurs au site, création de champs sémantiques permettant de proposer des recherches plus fines, etc.).

L'avantage du moteur interne, cependant, c'est qu'il permet de gérer l'affichage des résultats à travers les mêmes méthodes (squelettes) que le reste des pages de SPIP, et à l'intérieur du même environnement visuel.

### L'indexation

L'indexation est réalisée lors des visites du site public. Afin d'éviter que le cumul d'une indexation et d'un calcul de page ne mène à un *timeout* sur les serveurs particulièrement lents, SPIP attend qu'une page soit affichée en utilisant le cache[4].

L'indexation traite une à une les différentes données textuelles d'un contenu donné : par exemple, pour un article, le chapo, le descriptif, le titre, le texte... Pour chaque donnée textuelle, le score de chaque mot est calculé en comptant simplement le nombre d'occurrences. A cet effet, les mots de trois caractères ou moins sont ignorés (ils sont, pour la plupart, non significatifs, et alourdiraient la base de données) ; d'autre part, les caractères accentués sont translittérés (convertis en leurs équivalents non-accentués), pour éviter les

[4] Si, donc, vous avez mis tous les `$delais` à zéro, ou si votre site n'est pas visité (site de test), l'indexation n'aura pas lieu.

problèmes de jeux de caractères et aussi pour permettre d'effectuer des recherches en version non accentuée.

Ensuite, les scores de chaque mot sont cumulés, de façon pondérée, entre les différentes données textuelles du contenu indexé. La pondération permet, par exemple, de donner plus de poids aux mots présents dans le titre d'un article que dans le corps du texte ou le post-scriptum...

Les fonctions d'indexation peuvent être étudiées au sein du plugin recherche_etendue avec d'autres fonctions de gestion de l'indexation).

## La recherche

La recherche s'effectue simplement en séparant le texte de recherche en ses différents mots ; le même filtre est appliqué que lors de l'indexation : suppression des mots de trois lettres ou moins (sauf sigles), et translittération.

Pour chaque contenu recherché, le score des différents mots est ensuite récupéré puis additionné afin d'obtenir le score total. Enfin, les résultats sont en général affichés par ordre décroissant de score (`{par points}{inverse}`), c'est-à-dire de pertinence (mais cela est laissé à la volonté de la personne qui écrit les squelettes de mise en page).

## Performances

### Rapidité

Sur un serveur récent et pas trop chargé, l'indexation d'un texte long (plusieurs dizaines de milliers de caractères) prendra entre une et deux secondes : l'attente est presque imperceptible, comparée aux délais de chargement via le réseau. Les contenus courts sont indexés de façon quasi-instantanée. Bien sûr, ces affirmations doivent être modulées selon la taille du site.

Un site vraiment très gros risque de voir les temps d'indexation s'allonger légèrement ; pour relativiser, signalons qu'un site comme Le Courrier des Balkans (http://www.balkans.eu.org/) comporte, à la date d'écriture de ce texte, environ 3 800 articles publiés, et plus de 7500 messages de forum, et que le moteur de recherche de SPIP ne donne aucun signe de faiblesse.

Par ailleurs, statistiquement, on peut considérer de façon approximative que chaque contenu ne sera indexé qu'une seule fois : compte tenu qu'il y a en général beaucoup plus de visites sur un site que de mises à jour de contenus, le surcroît de charge du serveur apparaît négligeable.

### Qualité

La qualité de l'indexation est plus faible que sous des moteurs de recherche professionnels. PHP étant un langage plutôt lent, la phase d'extraction des mots a dû être simplifiée au maximum pour que les temps d'indexation restent minimes. Par conséquent, les données d'indexation comportent quelques « déchets », c'est-à-dire des morceaux de texte qui ne correspondent pas à de « vrais » mots, mais ont été indexés comme tels (il s'agit souvent de contenus techniques comme des noms de fichiers, ou de passages à la ponctuation malmenée). L'exemple d'uZine (http://www.uzine.net/), où l'on constate environ 2 % de tels « déchets »,

nous laisse cependant penser que ces données sont quantité négligeable, d'autant qu'il y a peu de chance qu'elles déclenchent un résultat positif lors d'une recherche.

La recherche n'offre pas d'opérateurs booléens, l'opérateur implicite étant grosso modo un « OU » logique. Cependant les articles trouvés s'affichent dans un ordre qui privilégie les résultats contenant le plus de mots orthographiés précisément selon la requête. Ainsi, une requête sur « la main rouge » mettra en évidence les articles contenant « main » et « rouge », loin devant les articles ne contenant que « maintenance » ou « rouget » - ceux-ci apparaîtront, mais plus loin dans le classement.

**Espace disque**

MySQL n'étant pas spécialement conçu pour le stockage de données d'indexation, l'utilisation du moteur de recherche a tendance à faire beaucoup grossir l'espace disque utilisé par la base de données. Pour donner quelque précision, disons qu'un contenu génère des données d'indexation de taille comprise entre la taille du contenu et le double de celle-ci. Donc, si l'on fait abstraction des données ne donnant pas lieu à indexation (les forums par exemple), l'indexation fait entre doubler et tripler la place prise par la base de données. Cela peut être gênant si la place vous est très comptée.

Si jamais vous désactivez le moteur de recherche afin d'économiser de l'espace disque, n'oubliez pas ensuite d'effacer les données d'indexation (dans la page de sauvegarde/restauration de la base de données) afin de réellement libérer l'espace disque occupé par ces données.

## Les boucles et balises de recherche

Mai 2001 — maj : août 2010

SPIP dispose d'un moteur de recherche intégré. Il faut donc prévoir une page permettant d'afficher les résultats des recherches.

### Le formulaire de recherche

Pour afficher le formulaire de recherche, il suffit d'insérer la balise :

**#FORMULAIRE_RECHERCHE**

Le formulaire renvoit par défaut vers `spip.php?page=recherche` ; vous devez donc réaliser un squelette `recherche.html` permettant d'afficher les résultats.

Vous pouvez décider d'utiliser une autre page d'affichage des résultats. Pour cela, il faut utiliser la balise de la manière suivante :

```
[(#FORMULAIRE_RECHERCHE{#URL_PAGE{xxx}})]
```

où `spip.php?page=xxx` est la page vers laquelle vous désirez envoyer l'utilisateur, et `xxx.html` est son squelette.

### Le squelette des résultats

Les boucles permettant d'afficher les résultats de la recherche sont, en réalité, des boucles déjà abordées ici : ARTICLES, RUBRESIQU, BREVES et FORUMS. Vous pouvez en effet effectuer des recherches non seulement sur les articles, mais aussi sur les rubriques, les brèves et les forums.

- La seule différence, par rapport à ce qui est documenté dans le manuel de référence des boucles, est le choix du critère de sélection, qui doit être **`{recherche}`**. Les autres critères d'affichage et les balises de ces boucles restent ici valables.
- Cependant, afin de classer les résultats par pertinence, on utilisera de préférence ce nouveau critère d'affichage : **`{par points}`**. Les résultats sont en général affichés par ordre décroissant de score (`{par points}{inverse}`), c'est-à-dire de pertinence.
- Enfin, on pourra utiliser la balise **`#POINTS`**, qui affiche la pertinence des résultats (attention, dans l'absolu cette valeur n'est pas très explicite, elle est surtout utile pour le classement des résultats).
- La balise **`#RECHERCHE`** affiche la requête formulée par le visiteur.

Exemple de boucle complète :

```
<h1><:resultats_recherche:>[(#RECHERCHE)]</h1>
<B_articles>
<h2><:info_articles_trouves:></h2>
<ul>
      <BOUCLE_articles(ARTICLES) {recherche} {par points}{inverse}>
      <li>#POINTS <a href="#URL_ARTICLE">#TITRE</a></li>
      </BOUCLE_articles>
</ul>
</B_articles>
```

## Surligner la requête dans les autres pages

La mise en évidence des termes de la recherche se fait par le biais de la bibliothèque de fonctions **jquery**.

- Si la variable `recherche` existe dans l'url, elle est d'abord nettoyée pour éviter toute attaque par injection, puis, si le terme de la recherche se trouve directement dans un bloc de **`class="surlignable"`** (ou dans un *bloc descendant* d'un bloc de `class="surlignable"`), il s'affiche en *highlight* dans un **`<span class="spip_surligne">.../span>`**.
- S'il se trouve directement dans un bloc de **`class="pas_surlignable"`** (ou dans un *bloc descendant* d'un bloc de `class="pas_surlignable"`), il s'affiche normalement (*nohighlight*).
- L'apparence visuelle du surlignement peut être changée en modifiant la définition de style de **`.spip_surligne`** (voir : « Modifier l'habillage graphique »).

Pour bénéficier de la mise en évidence des termes de la recherche *dès la première page d'affichage* (par exemple dans le cas où la recherche ne renvoit pas vers *recherche.html*), ajouter dans le fichier **mes_options.php** :

```
if (isset($_REQUEST['recherche'])) {
  $_GET['var_recherche'] = $_REQUEST['recherche'];
}
```

*Les balises **`#DEBUT_SURLIGNE`** et **`#FIN_SURLIGNE`** ainsi que la variable de personnalisation **`$nombre_surligne`** sont **obsolètes depuis SPIP 2.0.3**.*

# 6. Optimisation / Système

## Mutualisation du noyau SPIP

Février 2007 — maj : Janvier 2009

Il s'agit de pouvoir gérer plusieurs sites SPIP sur un seul serveur ou hébergement en n'utilisant qu'une seule fois les fichiers du noyau de SPIP.

Cela permet un gain d'espace disque important, ainsi qu'une possibilité de mise à jour de SPIP simple de l'ensemble des sites en ne mettant à jour que le noyau.

C'est avec SPIP 1.9.2 et ses améliorations [1] que la mutualisation devient plus robuste permettant la mise en place d'un partage du noyau de SPIP.

Cet article explique la procédure pour SPIP 1.9.2, sur des serveurs Apache [2] autorisant la réécriture d'url (url_rewriting).

Il y a plusieurs méthodes pour arriver aux mêmes résultats, selon que l'on souhaite configurer directement la mutualisation depuis un hébergement ou depuis un serveur.

### Le concept...

Les dossiers nécessaires au fonctionnement du noyau SPIP (ecrire, squelettes-dist, oo), et ceux marquant l'activité d'un site (config, IMG, tmp, local) sont clairement identifiables et séparés. C'est cette séparation qui permet d'avoir plusieurs sites SPIP autonomes pour un même noyau de SPIP.

Cette autonomie repose sur l'existence d'un quatuor de répertoires par site, dans lesquels Spip va écrire les données résultant de l'activité du site. Ces répertoires sont au nombre de quatre, car on distingue les données temporaires et permanentes d'une part, et les données accessibles par http et celles qui ne le sont pas d'autre part, d'où quatre types de données. Les deux répertoires inaccessibles par http seront protégées par un .htaccess installé automatiquement par SPIP (les administrateurs du serveur pourront même mettre ces répertoires en dehors de l'arborescence servie par http).

Avant SPIP 1.9.2, ces quatre types de données n'étaient pas clairement distingués, et se retrouvaient dans les répertoires IMG, CACHE, ecrire et ecrire/data. Avec SPIP 1.9.2, ces quatre répertoires sont distincts et nommés par les constantes PHP suivantes :

```
define('_NOM_TEMPORAIRES_INACCESSIBLES', "tmp/");
define('_NOM_TEMPORAIRES_ACCESSIBLES', "local/");
define('_NOM_PERMANENTS_INACCESSIBLES', "config/");
define('_NOM_PERMANENTS_ACCESSIBLES', "IMG/");
```

Dans une installation de Spip non mutualisée, ces répertoires sont créés à la racine du site, lors de l'application de la fonction `spip_initialisation` sur les quatre valeurs ci-dessus. Pour obtenir la mutualisation des sources de SPIP, il suffit de savoir associer une URL spécifique à son quatuor de répertoires spécifiques, en appelant la fonction `spip_initialisation` sur quatre autres valeurs, déduites de l'URL.

## Créer les bons répertoires

Pour démarrer la mutualisation, il faut tout d'abord partir d'un site fonctionnel. Pour les exemples, nous partirons d'un site appelé par l'url http://example.org/ stocké physiquement dans /home/toto/public_html/.

Il faut créer un répertoire (par exemple nommé 'sites') qui va contenir les répertoires des sites mutualisés, à la racine de la distribution (au même niveau que /ecrire).

- mutualisation d'un répertoire :
Si l'on souhaite que les adresses http://example.org/premier_site/ et http://example.org/deuxieme_site/ appellent chacun un site spip, il faut créer alors les sous-répertoires /premier_site et /deuxieme_site dans le répertoire /sites, puis dans chacun d'eux créer les répertoires /config, /IMG, /tmp, /local. Ces quatre répertoires doivent être accessibles en écriture. Il sera possible par la suite d'ajouter un répertoire /squelettes aussi, à côté de ces répertoires.

- mutualisation sous-domaine et domaine (quelques idées) :
Si l'on souhaite que l'adresse http://toto.example.org/, http://exemple.tld/ et http://utilisateur.example.com/ appellent chacun un site spip, on peut envisager de créer dans /sites les répertoires /toto, /exemple.tld, et /example.com/utilisateur

**Remarque :** Toutes les url doivent pointer à la racine de la distribution SPIP, c'est-à-dire dans /home/toto/public_html/. C'est le rôle que vont remplir soit un fichier .htaccess soit la configuration du serveur Apache expliqués plus loin.

## Des redirections bien faites !

Pour que SPIP reconnaisse qu'un site est mutualisé, il faut que le script spip.php (appelé par index.php) soit executé. Celui-ci cherchera, grâce à un code ajouté dans /config/mes_options.php si l'URL appelée correspond à un site mutualisé ou non. Pour cela, il faut qu'une adresse http://example.org/premier_site/ ou http://example.org/deuxieme_site/ soit redirigée vers http://example.org/ pour executer alors index.php...

C'est le rôle attribué au fichier .htaccess (ou directement dans la configuration du serveur Apache)

Il faut copier et renommer le fichier htaccess.txt à la racine de la distribution en .htaccess, puis le modifier :

Pour autoriser la réécriture d'URL (rien ne change normalement) : `RewriteEngine On`

Si la distribution SPIP est dans un sous-répertoire, modifier rewritebase. Ici, le site est à la racine donc : `RewriteBase /`

Enfin, dans réglages personnalisés, ajouter le code suivant pour que les répertoires /premier_site , /deuxieme_site et /troisieme_site soient traités depuis la racine de la distribution :

```
#Mutualisation
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)$ /$1/ [R,L]
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)/(.*) /$2 [QSA,L]
```

Le premier rewriterule redirige les adresses http://example.org/premier_site vers http://example.org/premier_site/. Le deuxième redirige tout ce qu'il y a derrière /premier_site/ à la racine, par exemple : http://example.org/premier_site/art... est redirigé vers http://example.org/article112. (Cependant, cette redirection est transparente, le nom de l'url ne change pas, il est toujours http://example.org/premier_site/art..., même et surtout pour php !)

**Et si SPIP est dans un sous répertoire ?**
Dans ce cas là, les fichiers de spip se trouvent dans /home/toto/public_html/spip/ , SPIP est appelé par http://example.org/spip/, les sites mutualisés par http://example.org/spip/premier_site/ ou http://example.org/spip/deuxieme_site/.

Il faut alors mettre dans le .htaccess :

```
RewriteEngine On
RewriteBase /spip/

#Mutualisation
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)$ /spip/$1/ [R,L]
RewriteRule ^(premier_site|deuxieme_site|troisieme_site)/(.*) /spip/$2
[QSA,L]
```

**Et pour rediriger tous les répertoires comme des sites mutualisés ?**

Il vous est possible **à la place** des rewriterules ci-dessus d'utiliser un code générique, utilisable quelque-soit le nom du répertoire du site mutualisé :

- à la racine

```
RewriteCond %{REQUEST_URI} !^/(config|squelettes-
dist|ecrire|IMG|oo|plugins|sites|squelettes|tmp|local)/(.*)
RewriteRule ^[^/]+/(.*) /$1 [QSA,L]
```

- ou dans un dossier /spip :

```
RewriteCond %{REQUEST_URI} !^/spip/(config|squelettes-
dist|ecrire|IMG|oo|plugins|sites|squelettes|tmp|local)/(.*)
RewriteRule ^[^/]+/(.*) /spip/$1 [QSA,L]
```

**Et pour les domaines et sous domaines ?**
Par un heureux hasard, il n'y a rien à faire ici puisqu'ils pointent normalement déjà vers la racine de la distribution SPIP...

## Mutualiser selon l'URL grâce à mes_options.php

C'est le fichier /config/mes_options.php à la racine de la distribution qui va réaliser la majeure partie du travail : il doit chercher si une URL est à mutualiser ou non, et initialiser SPIP en fonction.

De nombreux cas peuvent se présenter, entre les répertoires, les domaines et sous domaines. PHP peut utiliser deux variables pour tester les URLs qui ont appelé le script :

```
$_SERVER['REQUEST_URI']; // contient tout ce qu'il y a derrière le nom de
domaine : /premier_site/article112 par exemple...
```

```
$_SERVER['SERVER_NAME']; // contient le nom du domaine et sous domaine :
utilisateur.example.org par exemple
```

Ce sont ces deux variables qui vont être comparées à une expression régulière pour extraire le nom du répertoire qui contient la mutualisation.

Un moyen simple de mutualiser tous les répertoires est de copier le code suivant :

```
<?php
if ( preg_match(',/([a-zA-Z0-9_-]+)/?,',$_SERVER['REQUEST_URI'],$r)) {

   if (is_dir($e = _DIR_RACINE . 'sites/' . $r[1]. '/')) {

       $cookie_prefix = $table_prefix = $r[1];

       define('_SPIP_PATH',
           $e . ':' .
           _DIR_RACINE .':' .
           _DIR_RACINE .'squelettes-dist/:' .
            _DIR_RACINE.'prive/:'.
           _DIR_RESTREINT);

       spip_initialisation(
           ($e . _NOM_PERMANENTS_INACCESSIBLES),
           ($e . _NOM_PERMANENTS_ACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_INACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_ACCESSIBLES)
           );

      $GLOBALS['dossier_squelettes'] = $e.'squelettes';

       if (is_readable($f =
$e._NOM_PERMANENTS_INACCESSIBLES._NOM_CONFIG.'.php')) include($f);
   }
}
?>
```

La ligne preg_match récupère le nom d'un dossier dans l'arborescence de l'URL, par exemple 'premier_site' dans http://example.org/premier_site/ ... Puis le script tente une mutualisation si le répertoire /sites/premier_site/ existe.

Si SPIP est dans un répertoire /spip, il faut modifier la première ligne par : `if (preg_match(',/spip/([a-zA-Z0-9_-]+)/?,', $_SERVER['REQUEST_URI'], $r)) {`

Il faut dire dire à SPIP quel est le préfixe de table de base de données utilisé, ce que fait `$cookie_prefix = $table_prefix = $r[1];`
Cette ligne va créer des tables MySQL avec des préfixes ayant le nom du répertoire contenant le site : mon_site_article à la place de spip_article par exemple. Cela permet d'héberger tous les sites sur une seule et même base de données. Vous pouvez garder le préfixe par défaut (spip), mais il faut penser à avoir plusieurs bases de données différentes pour chaque site. Pour cela, mettre à la place :

```
$cookie_prefix = $r[1];
$table_prefix='spip';
```

La fonction spip_initialisation admet quatre paramètres qui sont l'adresse de chacun des répertoires nécessaires au fonctionnement du site mutualisé.

`$GLOBALS['dossier_squelettes'] = $e.'squelettes';` définit l'emplacement du dossier squelettes du site mutualisé.

Enfin les dernières lignes chargent un éventuel sites/premier_site/config/mes_options.php.

**Information :**

Toute modification du fichier config/mes_options.php du noyau SPIP affectera les options de tous les sites hébergés.

Par exemple, mettre dans ce fichier : `$type_urls = 'propres' ;`

Donnera par défaut à tous les sites ce type d'url... Mais chaque site peut le changer dans son propre /sites/mon_site/config/mes_options.php.

## Configurer apache pour les domaines et sous-domaines

La mutualisation côté serveur, pour ce qui concerne la gestion des sous-domaines ou des domaines reste simple, mais nécessite de créer quelques redirections d'URL dans la configuration du serveur Apache pour tenir compte de ces sites.

Voici un exemple de configuration pour un serveur nommé 'exemple.tld' (ici un SPIP mutualisé) utilisant des sous-domaines (SPIP appelé par http://utilisateur.example.org/spip/).

Le noyau SPIP est dans '/home/toto/public_html/spip/'.

Il faut alors créer les répertoires /sites/exemple.tld/ , /sites/exemple.tld/utilisteur/.

Le fichier de configuration se situe dans Apache 2/linux dans /etc/apache2/sites_availables/default (vous pouvez aussi créer un nouveau fichier dans le dossier sites_availables et l'activer).

```
# SERVEUR exemple.tld
# SPIP par sous domaine...
<VirtualHost *>
       ServerName exemple.tld
        ServerAlias *.exemple.tld

       # Redirection vers le SPIP noyau
        DocumentRoot "/home/toto/public_html"
        <Directory "/home/toto/public_html/">
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>

        # Seule l'adresse http://utilisateur.exemple.tld/spip/* doit être
redirigée

        # (utilisateur.exemple.tld/spip/* -> /home/toto/public_html/*)
        RewriteCond %{SERVER_NAME} (www\.)?([^.]+)\.example\.net$
        RewriteRule ^/spip/(.*) /home/toto/public_html/spip/$1 [QSA,L]

        # (utilisateur.exemple.tld/* ->
/home/toto/public_html/sites/exemple.tld/utilisateur/*)
        RewriteCond %{SERVER_NAME} (www\.)?([^.]+)\.example\.net$
```

```
        RewriteRule (.*) /home/toto/public_html/sites/exemple.tld/%1/$1
[QSA,L]

</VirtualHost>
```

Il est par contre nécessaire de tester ces mutualisations possibles dans /config/mes_options.php :

```
<?php
# pour utilisateur.exemple.tld/spip/
if (
preg_match(',(.*)\.exemple\.tld/spip/?,',$_SERVER['SERVER_NAME'].$_SERVER['
REQUEST_URI'],$r)) {

   if (is_dir($e = _DIR_RACINE . 'sites/exemple.tld/' . $r[1]. '/')) {

      $cookie_prefix = $table_prefix = $r[1];

      define('_SPIP_PATH',
           $e . ':' .
           _DIR_RACINE .':' .
           _DIR_RACINE .'squelettes-dist/:' .
           _DIR_RACINE.'prive/:'.
           _DIR_RESTREINT);

       spip_initialisation(
           ($e . _NOM_PERMANENTS_INACCESSIBLES),
           ($e . _NOM_PERMANENTS_ACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_INACCESSIBLES),
           ($e . _NOM_TEMPORAIRES_ACCESSIBLES)
           );

      $GLOBALS['dossier_squelettes'] = $e.'squelettes';

       if (is_readable($f =
$e._NOM_PERMANENTS_INACCESSIBLES._NOM_CONFIG.'.php')) include($f);
   }
}
?>
```

## Note sur les sauvegardes et les restaurations

Chaque site copie ses fichiers de sauvegardes dans le répertoire /sites/premier_site/tmp/dump (ou /sites/premier_site/tmp/upload/*login* pour les sauvegardes d’un administrateur restreint). Les restaurations se font par le même dossier.

**Attention :**

A l’heure actuelle, SPIP enregistre dans la base de données, pour ce qui concerne le dossier /IMG des sites mutualisés une adresse sites/premier_site/IMG/* au lieu de IMG/* comme pour un site SPIP seul.

Une restauration ne fonctionnera donc que si elle s’effectue dans le site qui a créé la sauvegarde.

Une astuce pour restaurer le site ailleurs consiste à éditer le fichier dump.xml et à remplacer toutes les occurrences (à l'exception de celles des déclarations dir_img et dir_logo dans l'ouverture de la balise spip) :

- sites/premier_site/IMG/
- par (SPIP seul) : IMG/
- ou (SPIP mutualisé) : sites/mon_nouveau_site/IMG/

Inversement, pour passer un SPIP seul dans un répertoire mutualisé, il faut remplacer toutes les occurrences de :

- IMG/
- par : sites/mon_site/IMG/

(Cette difficulté sera corrigée dans la prochaine version de SPIP.)

## Notes

[1] Nouvelle distribution des répertoires pour clarifier la mutualisation, correction des problèmes de logins et de noms de sites qui se mélangeaient parfois entre les sites mutualisés, fonction d'initialisation d'un site à mutualiser plus claire

[2] Pour information : ces procédures ont été testées avec Apache 2.0.55 et PHP 5.1.2 sur serveur Ubuntu Dapper Drake ainsi que sur PHP 5.1.6 sur serveur Ubuntu Edgy Eft

## Le validateur XML intégré

Février 2007 — maj : Décembre 2007

Depuis que le World Wide Web Consortium a lancé la Web Accessibility Initiative (http://www.w3.org/WAI/PF/), les problématiques de validation XML et d'accessibilité du Web aux déficients visuels ont convergé.

SPIP s'est très tôt intéressé aux problèmes d'accessibilité, avec la version 1.5 en 2002. En revanche, il a longtemps récusé XML eu égard à la rareté des pages conformes : l'abondance de pages HTML non XHTML fait que les navigateurs ont leur propre analyseur, et les langages basés sur XML, comme SVG, MathML et XQuery, ont connu un démarrage très lent.

Toutefois, la convergence des problématiques d'accessibilité et de validation d'une part, et l'implémentation en natif de SVG dans plusieurs navigateurs de l'autre, a amené SPIP, dans sa version 1.8.1, à offrir une interface avec des outils de validation comme Tidy ou le validateur officiel du W3C.

Ces outils, comme l'indiquent sans détour les pages d'accueil de leur site, souffrent de quelques limitations, révèlent des divergences dans leurs messages d'erreur, et ne sont pas installables par l'internaute moyen [1] Ils sont de plus inopérants face aux nouvelles technologies comme Ajax, qui modifient incrémentalement le contenu d'une page Web.

Plus grave encore, les *Document Type Definition* du W3C ont elles aussi des limitations, y compris la DTD XHTML 1.0 dite *stricte* : alors que la spécification officielle interdit l'emboîtement des balises `a`, `label` ou `form`, la définition formelle de la grammaire décrite considère comme valides les constructions `<label for='x'><label...` ou `<form action='.'><div><form...` par exemple [2].

Face à cette situation, SPIP innove radicalement en proposant un *validateur extensible, intégré, incrémental et optionnel*, fondé sur le *Simple Analyzer for XML* fourni en standard par PHP. Cette pluralité d'aspects le destine autant aux webmestres qu'aux graphistes et aux développeurs d'extensions de SPIP avec évidemment des manipulations différentes.

### Validation pour les webmestres

Pour les webmestres, mettre simplement dans le fichier `mes_options.php` l'instruction `$xhtml = 'sax';` provoquera une remise en page du code HTML produit par un squelette, chaque ligne comportant une seule balise ouvrante et au plus un attribut, avec renfoncement de la marge gauche proportionnellement au nombre de balises ouvrantes non fermées. Dans le cas où la page se révèle invalide, ce travail sera finalement ignoré, et le texte initial sera envoyé au client HTTP. Dans les ceux cas, la mise en cache éventuelle a évidemment lieu après l'action du validateur, qui se contente donc ici d'une simple mise en page.

Quelques précisions en ce qui concerne le traitement des attributs. Leur valeur sera systématiquement entourée d'une apostrophe, sauf si elle contient une apostrophe auquel cas elle sera entourée de guillemets (et si elle contient aussi des guillemets, ils seront écrits `&quot;`). En raison d'une erreur de conception de SAX [3], les entités XML seront préalablement converties dans le *charset* du site à l'exception de `&amp;` `&lt;` `&gt;` et `&quot;` qui sont correctement traitées par SAX (car c'est indispensable). La liste des entités et leur valeur seront déduites du DOCTYPE indiqué par le squelette ; à défaut SPIP prendra un

ensemble prédéfini équivalent à la DTD du latin1 enrichie de `&euro;`, `&oelig;` et `&OElig;` définis dans la DTD des symboles spéciaux.

## Validation pour les graphistes

Pour les créateurs de squelettes, le validateur est disponible à travers le débusqueur.

Cet outil n'est visible que des administrateurs du site, car ils disposent des boutons d'administration lorsqu'ils visitent les pages de l'espace public. L'un de ces boutons se nomme *Analyse XML* et déclenche explicitement la demande de validation de la page visitée. Cette analyse est d'abord confiée à SAX qui, même en l'absence de `DOCTYPE`, repère :

- les mauvais emboîtements de balises ouvrantes et fermantes ;
- les attributs sans valeur ;
- les attributs sans délimiteur ;
- les attributs sans espace avant le suivant ;
- les entités XML mal écrites (notamment un `&` littéral, alors que XML impose d'écrire `&`).

À la première erreur rencontrée, le débusqueur de SPIP essaiera immédiatement de retrouver de quel squelette provient l'erreur (il y en a plusieurs en cas d'inclusion) et à quelle ligne, en donnant des liens vers les fichiers sources (cette détermination ne peut pas toujours aboutir, une marge d'erreur est à prendre en compte).

Si cette première analyse est correcte, cette nouvelle version de SPIP va plus loin en prenant en compte le DOCTYPE de la page. Il peut être de type `PUBLIC` ou `SYSTEM`, et dans le premier cas la DTD sera mise en cache pour accélérer les analyses ultérieures. Chaque DTD peut en inclure d'autres, qui seront chargées pareillement. Afin d'éviter les redondances de calcul tout en tenant compte du système d'inclusion conditionnelle des DTD, SPIP met également en cache une structure de données spécifique qu'il déduit de la lecture de toutes les définitions d'éléments, d'attributs et d'entités issus du DOCTYPE indiqué. A l'aide de cette structure, SPIP *valide* la page, autrement dit vérifie que :

- tous les noms d'attributs utilisés dans une balise sont acceptés par la DTD ;
- tous les attributs obligatoires d'une balise sont présents ;
- toutes les valeurs d'attributs sont conformes au *motif* indiqué éventuellement dans la DTD ;
- tous les attributs de type ID ont une valeur composée de lettres, chiffres, souligné, tiret et deux-points ;
- tous les attributs de type IDREF ou IDREFS référencent des ID effectivement présents dans la page ;
- toutes les entités XML utilisées sont définies dans la DTD ;
- tous les noms de balises utilisées sont définis dans la DTD ;
- toute balise non vide est autorisée à l'être par la DTD (par exemple un `img` non vide sera dénoncé) ;
- toute balise vide est autorisée à l'être par la DTD (par exemple un `tr` vide sera dénoncé) ;
- toute balise utilisée est fille d'une balise l'admettant comme telle dans la DTD ;
- toute balise devant apparaître avant une de ces sœurs apparaît effectivement ainsi (par exemple `head` avant `body`)
- toute balise devant apparaître un nombre fixe de fois apparait effectivement ainsi (par exemple `title` une unique fois dans `head`).

Si des erreurs sont détectées, le validateur affichera un tableau de toutes les erreurs, avec leur nombre d'occurrences, des liens vers les lignes incriminées et des suggestions de corrections déduites automatiquement des constructions autorisées par la DTD. En l'absence d'erreur, le débusqueur affichera le code selon la mise en page décrite précédemment.

## Validation pour les développeurs

Les pages Web en accès restreint ont particulièrement besoin d'un validateur intégré pour être mises au point, puisque les outils de validation externes n'ont par définition pas accès à ces pages. En ce qui concerne les scripts, standards ou venant d'extension, de l'espace de rédaction de SPIP on leur applique la validateur XML en plaçant `$GLOBALS['transformer_xml'] = 'valider_xml';` dans le fichier mes_options.php. L'espace privé de SPIP 1.9.2 a été rendu conforme XHTML 1.0 grâce à ce mécanisme. Affecter cette variable globale à `'indenter_xml'` provoquera l'indentation du source HTML si celui-ci est conforme XML, sans chercher à le valider.

Il est également possible de lancer, avec la souris, l'analyse XML du résultat d'un script Ajax activé dans l'espace de rédaction. Un tel script ne retournant pas une page HTML complète mais seulement un fragment, le validateur intégré fabriquera une page avec le DOCTYPE courant, un en-tête réduit à la balise Title, et une balise Body composée de ce fragment. Une fenêtre s'ouvrira avec le résultat de l'analyse, comme pour une page normale, pendant que la fenêtre initiale recevra le résultat du script Ajax comme d'habitude. En raison d'une spécification du W3C inutilisable quant au modèle d'événements [4], ce lancement du validateur n'est pas provoqué par un bouton spécifique de la souris, mais par un clic pendant lequel une au moins des deux touches Alt ou Meta est enfoncée.

Par ailleurs, le validateur de SPIP peut être appliqué à n'importe page présente sur le Web. Tout site SPIP installé à l'URL `http://`*u* contient la page `http://`*u*`ecrire/valider_xml` que les auteurs du site peuvent invoquer explicitement pour valider des pages non limitées à celle du site. Ce validateur ne s'applique cependant qu'à des documents XML ; en l'absence de DOCTYPE, la DTD XHTML1.0 transitionnal sera prise.

Comme le suggèrent les lignes ci-dessus, le validateur s'applique à n'importe quel DOCTYPE, y compris ceux référençant des DTD résidantes sur le site. Rendre accessibles des pages Web, c'est être plus rigoureux dans l'utilisation des attributs et des balises, on peut donc facilement se construire un outil de validation d'accessibilité en écrivant des DTD moins laxistes que la XHTML 1.0 dite stricte. Remplacer `#IMPLIED` par `#REQUIRED` dans les attributs jugés indispensables est immédiat. Forcer `input, select, textarea` et `button` a être exclusivement des filles de la balise `label` est à peine plus difficile. En outre, le validateur accepte n'importe quel motif (au sens de PCRE) comme type d'attribut, il sera alors appliqué à chaque occurrence de cet attribut dans la page analysée.

Enfin, il est possible de définir ses propres règles de validation associées à un attribut. Les types usuels ID, IDREF et IDREFS sont implémentés par les fonctions validerAttribut_ID, validerAttribut_IDREF, validerAttribut_IDREFS. Il suffit d'introduire de nouveau type S1 ... Sn dans la DTD, et de définir les fonctions associées dans le fichier mes_options pour provoquer des contrôles personnalisés. En fin d'analyse, la fonction surchargeable `inc_valider_passe2` est appelée, afin d'effectuer les contrôles rétrospectifs (c'est ici que les attributs IDREF sont vérifiés comme référençant des attributs effectivement présents). Cette interface de programmation est encore un peu frustre et sera revisée après expérimentation. Mais elle permet dès à présent de mettre sur pied très rapidement de nouvelles spécifications d'accessibilité.

## Notes

[1] Le validateur du W3C se présente sous la forme d'un archive de plus de 1Mo, comportant essentiellement un script CGI en Perl, nécessitant plusieurs modules de ce langage et son interface avec le serveur HTTP. En outre, l'analyse syntaxique et la validation ne sont pas effectuées par ce programme, mais déléguées à l'analyseur *onsgmls* programmé en C++ et nécessitant donc d'être compilé après chargement des sources du projet OpenSP de 1Mo également (certains systèmes d'exploitation incluent cependant déjà l'exécutable de 128Ko résultant). Cette difficulté explique sans doute la rareté de ses installations, qui du coup sont souvent saturées.

[2] Cette carence est justifiée par ce passage de la recommandation XHTML qui affirme : *The HTML 4 Strict DTD forbids the nesting of an 'a' element within another 'a' element to any descendant depth. It is not possible to spell out such prohibitions in XML.* Cette affirmation n'est etayée par aucune démonstration ou référence scientifique. Les théorèmes établissant les pouvoirs de l'analyse syntaxique automatique ont été énoncés et démontrés dès les années 1950, et contredisent une telle affirmation. W3C, go to school ?

[3] À la rencontre de ce qu'il considère comme un lexème, SAX appelle un *preneur d'événements*. Faute de considérer les attributs comme des lexèmes, il provoquera la même séquence d'appel pour les 3 textes suivants :

```
&eolig;&EOlig;<a ...
&eolig;<a title='&EOlig;' ...
<a title='&eolig;' href='&EOlig;' ....
```

En conséquence, le preneur d'événements *Entité XML* ne saura pas si l'entité provoquant son appel fait partie de l'élément de texte précédant la prochaine balise, ou si elle fait partie de l'un des attributs de cette balise, et lequel. L'ambiguïté ne peut même pas être levée à l'aide des numéros de ligne et de colonne (ou de caractère du flux), qui indiquent dans tous les cas l'emplacement précédant la balise.

[4] Alors que le système d'exploitation le plus répandu dans le monde suivait pour une fois sagement les leçons d'Unix et plus précisément de X-Window, en décrivant un bouton de souris par un masque de bits, le W3C a cru bon de produire une spécification incompatible et mal conçue, que ne suit pratiquement aucun navigateur.

## Tidy : Correction XHTML 1.0

Avril 2005 — maj : Décembre 2008

Les pages produites par SPIP peuvent être analysées par le Le validateur XML intégré. Il existe aussi une interface avec Tidy ; elle exige une installation séparée mais a l'avantage d'essayer, souvent avec succès, de corriger automatiquement les erreurs de validation.

### Principe général

Tidy est un outil (externe à SPIP) qui permet de transformer du code HTML 4 *relativement propre* en code XHTML 1.0 transitional valide. Cet outil aide les webmestres à conformer leurs sites aux recommandations XHTML, même dans les messages de forums librement composés par les visiteurs.

**Important :** Tidy n'est pas un outil « magique » : il est incapable de transformer du code « très sale » en code conforme. Face à certaines « erreurs » de code, il refuse purement et simplement de fonctionner. Des interventions manuelles sur les erreurs qu'il dénonce sont donc à prévoir..

- **Étape 1 :** il convient, avant tout, de créer du code aussi propre et conforme que possible, avant même le passage par Tidy. Cela se fait à plusieurs niveaux :

  - tout d'abord, SPIP produit, dans ses traitements typographiques, du code propre, et à chaque version plus conforme (« *compliant* ») ; l'espace privé et les squelettes de SPIP sont conformes XHTML strict.

- **Étape 2 :** si Tidy est présent sur le serveur, et si le webmestre active cette option (voir plus loin), alors SPIP fait passer les pages produites par Tidy, qui va alors tenter de nettoyer les pages et de les transformer en pages conformes.

- **Étape 3 :** si le traitement a bien fonctionné (Tidy n'a pas rencontré d'erreur « bloquante »), alors la page affichée est bien du XHTML 1.0 ; en revanche, si Tidy n'a pas fonctionné (voir plus loin), alors c'est la page d'origine qui est affichée — dans ce cas (c'est un outil important à prendre en compte), SPIP affiche un bouton d'administration signalant l'« erreur Tidy », et créé un fichier récapitulant les différentes pages ayant rencontré des erreurs.

Encore une fois, afin de ne pas préter à l'outil des possibilités « magiques » qu'il n'a pas, et à ses développeurs des intentions qui ne sont pas les leurs, il est important de noter qu'il ne s'agit pas de se reposer entièrement sur Tidy, dont les limites sont connues, mais de l'intégrer dans une logique plus large de mise en conformité :

- d'abord en améliorant le code produit par SPIP,
- en utilisant ensuite Tidy à la fois comme outil de « nettoyage » du code, mais aussi *en proposant une indication des erreurs* pour permettre au webmestre d'améliorer son code.

La mise en conformité du code ne peut reposer sur une solution purement technique, mais sur un travail personnel pour lequel SPIP offre un outil de suivi.

### Installer Tidy

- **Tidy, extension de PHP**

Tidy existe en tant qu'extension de PHP. C'est la façon la plus simple de l'utiliser, l'outil étant alors directement utilisable par le webmestre. Pour déterminer sa présence, on pourra consulter la page `ecrire/?exec=info` de son site pour obtenir la configuration de son serveur et la liste des extensions de PHP disponibles.

*N.B. :* Tout retour d'expérience de la part de webmestres intéresse les développeurs — sur la liste spip-dev. (Nous avons besoin de retours d'expérience sur les versions 1 et 2 de Tidy, c'est-à-dire sur des sites en PHP 4, en PHP 5, mais également des installations via PEAR.)

- **Tidy comme programme indépendant**

Il est par ailleurs possible d'utiliser Tidy en ligne de commande (c'est-à-dire en tant que programme indépendant de PHP s'exécutant directement sur le serveur).

Cette version est particulièrement pratique, puisque :

- il existe des versions de Tidy déjà compilées pour la plupart des systèmes d'exploitation,
- il est souvent possible et simple d'installer ces versions de Tidy sur un hébergement sans avoir d'accès *root*,
- certains administrateurs de sites ont subi des incompatibilités lors de l'installation de Tidy en extension PHP (avec, semble-t-il, ImageMagick) ; la version en ligne de commande ne provoque pas ce genre de problème.

Avant toute chose, vérifiez que Tidy n'est pas déjà présent sur votre serveur. Pour cela, installez dans le fichier `mes_options.php` les lignes suivantes :

```
define('_TIDY_COMMAND', 'tidy');
$xhtml = true;
```

Et vérifiez sur votre site public si les pages sont modifiées (soit transformées en XHTML, soit affichage du message « Erreur tidy »). Si cela ne fonctionne pas, pensez à supprimer ces deux lignes, et essayez d'installer Tidy selon la méthode suivante (ou demandez à la personne responsable de votre hébergement de le faire).

Vous pouvez installer une version déjà compilée de Tidy correspondant à votre système.

- On trouvera ces versions sur le site officiel de Tidy ; il en existe pour Linux, divers *BSD, MacOS X, etc.
- Décompressez l'archive téléchargée, et installez le fichier « tidy » sur votre site.
- Vérifiez les droits d'exécution de ce fichier sur le serveur (si nécessaire, passez les droits en « 777 »). (Si vous avez un accès SSH à votre serveur, vous pouvez tester le programme directement depuis le terminal. Si vous n'avez pas un tel accès, rien de grave, passez aux étapes suivantes, sachant que vous serez plus démuni si cela ne fonctionne pas du premier coup.)
- Configurez l'accès à ce fichier en indiquant le chemin d'accès en le définissant ainsi :

  ```
  define('_TIDY_COMMAND', '/usr/bin/tidy');
  $xhtml = true;
  ```

Si le chemin indiqué dans `_TIDY_COMMAND` est correct, alors Tidy sera déclenché lors des affichages des pages de votre site public.

**Important.** La définition de `_TIDY_COMMAND` doit se trouver dans `/ecrire/mes_options.php` et non à la racine du site dans `/mes_fonctions.php`. Cela est dû au fonctionnement assez spécifique du système (post-traitement des fichiers tirés du cache de SPIP).

La partie `$xhtml = true;`, en revanche, fonctionne comme une « variable de personnalisation » ; vous pouvez, si vous souhaitez faire des essais ou restreindre le fonctionnement à une partie du site, définir cette variable au niveau du fichier d'appel, par exemple `article.php3` si vous ne voulez passer tidy que sur les articles.

## Nettoyez votre code...

Encore une fois, il faut bien comprendre que Tidy n'est capable de rendre conforme que du code qui est déjà, à l'origine, très propre. Avec les squelettes livrés avec SPIP et le code produit par défaut par SPIP, cela ne pose aucun problème : le code est très proche du HTML 4 conforme (« *compliant* »), aussi Tidy n'a aucun mal à en faire du XHTML 1.0 transitional parfaitement conforme.

Lorsque Tidy a bien fonctionné, vous constatez dans le code source de vos pages :
— que le « DOCTYPE » de votre page est devenu « XHTML... »,
— que le code est joliment indenté,
— que l'ensemble passe la validation W3C sans difficulté.

Si le DOCTYPE n'est pas modifié, c'est que Tidy a renoncé à corriger la page, dans laquelle il a rencontré des erreurs impossibles (pour lui) à corriger.

Les erreurs peuvent avoir deux sources : les squelettes, et le texte des articles.

- **Vos squelettes ne sont eux-mêmes pas conformes ;** dans ce cas, il faut les corriger. C'est le cas le plus fréquent.

Vous pouvez, ici, commencer par désactiver Tidy (passer la variable `$xhtml` à `false`), et passer vos pages à un validateur (le Le validateur XML intégré ou le W3C Validator) en visant la conformité HTML 4.01 transitional au minimum.

Une fois vos squelettes aussi proches que possible du HTML 4, Tidy n'aura pas de difficulté à produire du XHTML conforme. Si ces pages sont complètement conformes, c'est encore mieux ! (Et pas impossible : les squelettes livrés avec SPIP le sont déjà).

- **Certaines articles contiennent des codes erronés**

SPIP laissant les rédacteurs travailler en « code source », ceux-ci peuvent insérer du code non conforme à l'intérieur de leurs articles. (Par exemple, dans la documentation de `www.spip.net`, on trouve à certains endroits l'utilisation de balises HTML `<tt>...</tt>` que Tidy considère comme inacceptables.)

Aussi, une fois les squelettes nettoyés, on pourra chercher à corriger les textes de certains articles (cela concerne, donc, les insertions de HTML directement dans les articles ; encore

une fois, le code produit par SPIP est essentiellement conforme, et ne provoque pas de « blocage » de Tidy).


Pour cela, outre l'apparition, sur les pages concernées, d'une bouton d'aministration intitulé « Erreur Tidy », SPIP tient à jour en permanence un fichier `/ecrire/data/w3c-go-home.txt` qui contient la liste des pages impossibles à valider [1]. Une fois vos squelettes rendus propres (paragraphe précédent), le nombre de pages défaillantes devrait être limité aux articles contenant du code HTML non accepté par Tidy.

Il est assez difficile de définir précisément ce que Tidy considère comme une « erreur insurmontable ». Par exemple, les balises mal fermées ne sont pas réellement considérées comme impossible à corriger (par exemple, passer en italique dans un paragraphe et fermer l'italique dans un autre paragraphe fabrique du code HTML non conforme, que Tidy parvient cependant à bien corriger). Le plus souvent, il s'agit de balises insérées à la main totalement inexistantes (par exemple : taper `<bt>` à la place de `<br>`), ou de balises HTML considérées comme obsolètes dans le HTML 4 (telles que `<tt>` ou `<blink>`), que Tidy refusera purement et simplement de traiter.

## Conclusion

Encore une fois, l'outil Tidy ne doit surtout pas être considéré comme un produit « miracle » : il ne transforme pas du code sale en code conforme. Son intégration dans SPIP suit donc une logique d'accompagnement d'un processus de nettoyage :

- SPIP lui-même continue à produire du code de plus en plus propre ;
- Tidy sert alors à mettre la dernière touche de nettoyage sur du code déjà très « compliant » (au passage, en résolvant quelques incompatibilités difficiles à gérer avec un code unique entre le HTML et le XHTML, telles que certaines balises auto-fermantes en XHTML, et non fermées en HTML, telles que `<br />`) ;
- Tidy est ensuite utilisé pour identifier les erreurs de codage dans le code source des articles eux-mêmes (lors de l'insertion, assez fréquent, de code HTML « à la main » dans le corps des articles).

## Notes

[1] Ce fichier titre son nom de l'article W3C go home! (http://www.uzine.net/article1979.html), publié sur uZine, qui critiquait l'acharnement des prophètes de la *compliance* à emm... les webmestres qui se contentent de faire des pages Web ; l'essentiel, rappelons-le, est de publier sans se casser la tête. Si en plus on peut le faire de manière conforme, tant mieux, et c'est l'objet de cette passerelle SPIP-Tidy.

## Sécurité : SPIP et IIS

Empêcher l'accès aux données confidentielles de SPIP sous Microsoft IIS

Août 2004 — maj : Décembre 2007

### Sécurité par défaut de SPIP

Il existe deux dossiers « sensibles » dans SPIP, ce sont `CACHE` et `ecrire/data`. Le premier comporte tous les fichiers qu'utilise votre cache pour accélérer l'affichage des pages, il est donc moyennement sensible, mais le deuxième stocke les journaux d'activité de spip (les `spip.log`) et vous permet notamment de créer `dump.xml`, le fichier de sauvegarde de la base de données.

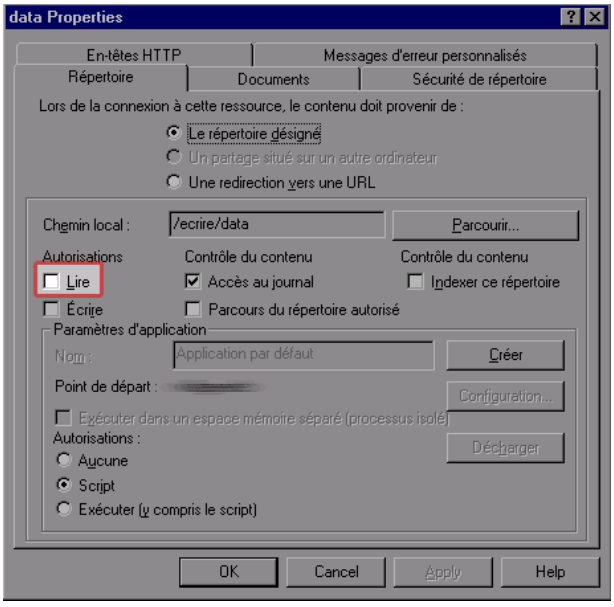
Or le fichier `dump.xml` contient des données très sensibles : en particulier on peut y voir *tous* les articles, même s'ils ne sont pas rendus publics sur le site, sans compter qu'il liste également les identifiants et les mots de passe[5] [1] des rédacteurs et administrateurs du site.

La sécurité de tous ces fichiers est assurée traditionnellement par des fichiers de configuration d'accès nommés `.htaccess`. SPIP génère automatiquement ces fichiers pour empêcher l'accès aux données sensibles stockées sur le serveur : vous pouvez vérifier que `CACHE` et `ecrire/data` contiennent chacun un fichier `.htaccess`. Hélas, ces fichiers fonctionnent sous Apache (le serveur Web libre faisant tourner la majorité des sites Web de l'Internet) mais pas sous IIS (Internet Information Services, le serveur Web de Microsoft).

### Protéger ses données sous IIS : une étape de plus

Si votre site SPIP est installé sur un IIS, n'importe qui peut donc voir les dossiers censément sécurisés via `.htaccess` : il faut donc les protéger.

Pour protéger un dossier sur votre site : allez dans le panneau d'administration de votre serveur Web, faites un clic droit sur le dossier concerné, cliquez sur « propriétés », et dans l'onglet « Répertoire » décochez la case « Lire ».




[5] Les mots de passe sont chiffrés par SPIP, mais gardez bien à l'esprit qu'aucune protection n'est inviolable.

**Le panneau de propriétés du dossier /ecrire/data/**

Décocher la case "Lire" suffit à protéger le dossier exactement comme le fait Apache avec les fichiers .htaccess

Faites cette opération pour chacun des deux dossiers `CACHE` et `ecrire/data`. Si la manipulation est bonne, vous ne devriez plus pouvoir accéder aux fichiers de ces dossiers à travers le serveur web. Testez votre configuration en essayant d'afficher `http://www.votresite.com/ecrire/data/spip.log` avec votre navigateur. Vous devriez obtenir un message du type « Accès refusé ».

## Le support LDAP

Décembre 2001 — maj : Octobre 2009

Attention, cet article est vraiment destiné à des utilisateurs avancés, qui maîtrisent l'usage de LDAP et souhaitent appuyer SPIP sur un annuaire LDAP existant.

LDAP (Lightweight Directory Access Protocol) est un protocole permettant d'interroger un annuaire contenant des informations d'utilisateurs (nom, login, authentification...). Il est possible de vérifier si un rédacteur est dans la base LDAP avant de lui donner accès à l'espace privé.

À l'installation, SPIP détecte si PHP a été compilé avec le support LDAP. Si oui, à la cinquième étape (« créer un accès »), un bouton permet d'ajouter un annuaire LDAP à la configuration SPIP. La configuration qui suit est relativement simple, elle essaie de deviner les paramètres au maximum. Notamment, elle permet de choisir le statut par défaut des auteurs venant de l'annuaire : ceux-ci peuvent être rédacteurs (conseillé), administrateurs ou simples visiteurs.

*Note : par défaut, l'extension LDAP de PHP n'est généralement pas activée, donc SPIP n'affichera pas le formulaire correspondant lors de l'installation. Pensez à activer l'extension LDAP dans votre installation de PHP si vous voulez utiliser LDAP avec SPIP.*

Si SPIP est déjà installé et que vous voulez configurer l'annuaire LDAP, il faudra reprendre l'installation en effaçant le fichier `config/connect.php`.

Une fois la configuration correctement effectuée, tous les utilisateurs de l'annuaire LDAP seront identifiés en tapant leur login (ou nom) dans l'annuaire LDAP, puis leur mot de passe. Notez que cela n'empêche pas de créer directement des auteurs dans SPIP ; ces auteurs ne seront pas recopiés dans l'annuaire mais gérés directement par SPIP. D'autre part les informations personnelles (biographie, clé PGP...) des auteurs authentifiés depuis LDAP ne seront pas non plus recopiées dans l'annuaire. Ainsi SPIP n'a besoin que d'un accès *en lecture seule* à l'annuaire LDAP.

**Important :** créez toujours un premier administrateur « normal » (non LDAP) lors de l'installation de SPIP. C'est préférable pour éviter d'être bloqué en cas de panne du serveur LDAP.

### Pour en savoir plus

Les infos de connexion au serveur LDAP sont écrites dans `connect.php`. Corollaire : il faut supprimer ce fichier et relancer l'installation pour activer LDAP sur un site SPIP existant.

Dans la table spip_auteurs, est ajouté un champ "source" qui indique d'où viennent les infos sur l'auteur. Par défaut, c'est "spip", mais ça peut aussi prendre la valeur "ldap". Ca permet de savoir notamment quels champs ne doivent pas être changés : en particulier, on ne doit pas autoriser la modification du login, car sinon il y a une perte de synchronisation entre SPIP et LDAP.

À l’authentification, les deux méthodes sont testées à la suite : SPIP puis LDAP. En fait un auteur LDAP ne pourra pas être authentifié par la méthode SPIP (méthode standard avec challenge md5) car le pass est laissé vide dans la table spip_auteurs. Un auteur SPIP, quant à lui, sera authentifié directement depuis la table spip_auteurs. D’autre part, si le login entré ne vient pas de SPIP, le mot de passe est transmis en clair.

Quand un auteur LDAP se connecte pour la première fois, son entrée est ajoutée dans la table spip_auteurs. Les champs remplis sont : nom, login et email qui viennent de LDAP (champs ’cn’, ’uid’ et ’mail’ respectivement) et le statut dont la valeur par défaut a été définie à l’installation (rédacteur, admin ou visiteur). Important : on peut modifier le statut par la suite, afin de choisir ses admins à la main par exemple.

Une fois un auteur connecté, il est authentifié par la voie classique, c’est-à-dire simplement avec le cookie de session. De même, les infos prises en compte dans l’affichage et les boucles sont celles de spip_auteurs, pas celles de l’annuaire.

Pour les auteurs SPIP, rien ne change. On peut les créer et les modifier comme à l’habitude.

## Rapidité du site public

Juin 2001 — maj : Décembre 2007

Contrairement à la plupart des systèmes de publication gratuits, SPIP intègre un système de cache permettant d'accélérer l'affichage du site public. Quelques pistes pour comprendre ce qui influe sur la rapidité de votre site...

### Optimiser un site

Si vous vous inquiétez pour la rapidité de votre site, il est bon de vous intéresser aux pistes suivantes :

- Votre hébergement Web offre-t-il des performances de bonne qualité ? Evidemment, c'est subjectif. L'expression « mauvaise qualité » recouvre à coup sûr la plupart des hébergeurs gratuits (notamment Free). « Bonne qualité » inclut forcément une machine dédiée (i.e. qui ne sert qu'à votre site) de fabrication récente, mais aussi des hébergeurs commerciaux pas trop au rabais. Entre les deux, ça devient très subjectif, en fonction de vos exigences, de la taille de votre site....
- Si la qualité de votre hébergement laisse à désirer, vous aurez intérêt à ne pas créer de squelettes trop complexes, i.e. qui demandent à SPIP d'afficher trop d'informations différentes. Cela vaut pour tout type d'informations : tout ce qui, dans les squelettes, est susceptible d'être transformé par SPIP en données affichables. Notez, en particulier, que les squelettes fournis par défaut démontrent au maximum les possibilités de SPIP, et par conséquent génèrent des pages assez lourdes.
- N'oubliez pas non plus de régler les délais d'expiration des différents types de pages. Ainsi, si votre site contient un grand nombre d'articles en archives, vous avez peut-être intérêt à augmenter la durée d'expiration des articles, sinon les articles consultés peu souvent ne bénéficieraient pas du système de cache.

### L'influence du cache

La présence du cache change quelque peu la donne en matière de rapidité. Ce n'est pas tant le nombre de visites de votre site qui sera le point critique, que la capacité de votre serveur à recalculer les pages dans le temps imparti au script PHP (en effet, sur la plupart des serveurs, une limite de durée d'exécution par appel de script est fixée afin d'éviter les abus et les erreurs de programmation). Par contre, si la page demandée est dans le cache et n'a pas expiré, la réponse du serveur devrait être quasi-instantanée (dans le cas contraire, votre serveur est vraiment très chargé).

La qualité des performances devient ainsi objectivement mesurable si, lors du recalcul d'une page du site, on obtient un « *timeout* », c'est-à-dire que le serveur a dépassé le temps maximal d'exécution d'un script PHP. Alors il faut soit changer d'hébergement, soit se résoudre à afficher des pages plus simples : pour cela, modifier les squelettes pour afficher moins d'informations sur une même page.

## Sur une machine dédiée

Si vous utilisez votre propre machine, il faut vous assurer qu'elle pourra tenir la charge. N'importe quelle machine pas trop vieille (moins de trois ans environ) devrait en être capable.

Par contre, l'utilisation de SPIP, par rapport à d'autres systèmes de publication, permet de mutualiser les ressources techniques entre plusieurs sites. En effet, tant que le cache est utilisé, la machine est peu sollicitée, donc plusieurs sites peuvent cohabiter sans problème (sauf s'il y a vraiment un très grand nombre de visites). Le problème est donc surtout de prévenir qu'il y ait trop de passagers à bord, c'est-à-dire qu'un trop grand nombre de « services » hébergés (sites Web, boîtes à e-mail...) mette en péril la qualité du service.

## Utiliser des URLs personnalisées

Mai 2001 — maj : avril 2010

Après l'installation, les pages générées par SPIP utilisent des adresses relatives ressemblant à `spip.php?article765`, donnant des URLs du type `http://www.spip.net/spip.php?article765`.

Il y a possibilité d'avoir des adresses plus à votre goût — par exemple `article123.html` ou `Titre-de-l-article.html`, et SPIP vous aide en partie dans cette tâche.

Cette fonctionnalité fait appel à la distinction entre deux types d'URLs :

- *l'URL apparente* d'une page, c'est-à-dire telle qu'elle est tapée et/ou affichée dans la barre d'adresse du navigateur. Par exemple `http://www.spip.net/fr_article765.html`. Ce sont ces URLs qu'on cherche à rendre plus « jolies » ou plus « signifiantes » ;

- *l'URL réelle* de la page, c'est-à-dire l'URL qui est « vue » par SPIP lorsque la page est calculée sur le serveur. Par exemple `http://www.spip.net/spip.php?article765` ; en général, cette URL peut aussi être tapée directement dans le navigateur (vous pouvez vérifier).

### Choisir le type d'URLs apparentes

Depuis SPIP 2.1, le choix est faisable via l'interface privé. Allez dans Configuration du site->fonction avancées, et choisissez le type d'URL souhaité.

Avant SPIP 2.1 il faut suivre la procédure inscrite en dessous.

Dans le fichier `ecrire/mes_options.php`[6] (à créer le cas échéant), vous pouvez déclarer une variable PHP contenant le type d'URLs à utiliser. En l'absence de ce réglage, SPIP utilisera :

```
$type_urls = "page";
```

La variable `$type_urls` détermine le nom du fichier PHP qui est appelé pour gérer les URLs. Avec la déclaration par défaut ci-dessus, c'est le premier `urls/page.php` trouvé dans SPIP_PATH.

Vous remarquerez que SPIP propose aussi les fichiers `urls/standard.php`, `urls/html.php`, `urls/propres.php`, `urls/propres-qs.php` et `urls/propres2.php` dans le répertoire `ecrire/`

- Le fichier `urls/html.php` permet de traiter des adresses du type (« article123.html »). Vous pouvez décider d'utiliser les « URLs "html" » en mettant dans `ecrire/mes_options.php` la ligne :
`$type_urls = "html";`

- Le fichier `urls/propres.php` permet de traiter des adresses du type (« Titre-de-l-article »). Il faut alors ajouter :
`$type_urls = "propres";`

[6] Remarque : les versions précédentes de SPIP incluaient le fichier `inc-urls.php3` à la racine du site s'il était présent ; cette méthode est encore valable mais est considérée comme obsolète...

- Le fichier `urls/propres2.php` est une variation du précédent, qui donne des adresses du type (« Titre-de-l-article.html »). Il faut alors ajouter :
`$type_urls = "propres2";`

- Le fichier `urls/propres-qs.php` est une variation du précédent, qui donne des adresses du type (« `./?Titre-de-l-article` »). Il faut alors ajouter :
`$type_urls = "propres-qs";`. Ce dernier est à utiliser si votre hébergeur ne vous permet pas d'utiliser le module de réecriture d'urls d'apache (cf. mod_rewrite)

- Enfin, le fichier `urls/standard.php` permet aux nostalgiques des versions précédentes de spip de donner des adresses du type (« `article.php?id_article=765` »). Il faut alors ajouter :
`$type_urls = "standard";`

Si vous voulez plutôt utiliser vos propres adresses (ce pour quoi vous devez savoir programmer en PHP), il est fortement conseillé de partir d'un des fichiers existants et de le recopier sous le nom que vous aurez choisi : `urls/XXX.php`. Il est par exemple très aisé de modifier la fonction `_generer_url_propre()` dans `urls/propres.php` pour obtenir des variations très intéressantes ; si vous faites cela, merci de partager vos modifications sur le site SPIP Contrib (http://www.spip-contrib.net/).

## Programmer la traduction des adresses apparentes en adresses réelles

Pour que l'adresse `article123.html` appelle bien en réalité le fichier PHP spip.php avec comme paramètre `id_article=123`, il va falloir configurer le serveur Web qui héberge votre site, soit dans un fichier *.htaccess* (ça ne marche pas toujours), soit dans le fichier de configuration centrale du serveur si vous y avez accès. Cela utilise, sous le serveur Apache (le plus utilisé), ce qu'on appelle des *Rewrite Rules* : des règles de réécriture d'adresses Web.

Savoir écrire ces règles n'est pas simple pour les non-programmeurs, et nous ne pouvons pas vous donner de solutions infaillibles car cela dépend de votre configuration : cette partie est entièrement entre vos mains (ou celles de votre hébergeur).

Un fichier `htaccess.txt` à titre d'exemple, qui fonctionne sur la plupart des hébergeurs avec les types d'URLs cités précédemment (« standard », « html », « propres » et « propres2 »). Pour l'activer il faut le recopier à la racine du site sous le nom `.htaccess`. Il est fortement conseillé de l'ouvrir au préalable pour vérifier quelques aspects de configuration.

Vous devrez ensuite tester la validité de ces adresses, en appelant la page « Voir en ligne » sur un article, un auteur, une brève, une rubrique, etc.

## Générer les URLs apparentes dans les pages SPIP

Afin d'afficher partout les URLs du type choisi, utilisez dans vos squelettes les balises `#URL_ARTICLE`, `#URL_RUBRIQUE`, `#URL_BREVE`, etc.

## Transition d'un type d'URLs à l'autre

Tout est prévu pour que la transition d'un type d'adresses à l'autre se fasse en douceur : installez le fichier `htaccess.txt`, et vous pouvez ensuite librement basculer des adresses « standard » aux adresses « propres2 », « propres » ou « html », et vice-versa, sans jamais

provoquer d’erreur 404 pour les visiteurs (ou les moteurs de recherche) qui auraient mémorisé les anciennes adresses.

Dernier détail pour faciliter la transition, si vous choisissez les URLs propres ou propres2, les visites des pages portant les anciennes adresses (standard ou html) sont redirigées automatiquement vers les nouvelles adresses.

# 7. Bases de données

## Les bases de données en SPIP

Décembre 2008 — maj : Juillet 2009

SPIP peut être vu comme un outil de mise en page dans un format MIME (HTML, RSS, ICS ...) d'extraits de bases de données. Une des nouveautés de SPIP 2.0 est de pouvoir rassembler dans une page des données en provenance de plusieurs bases SQL, accessibles par des serveurs éventuellement différents et distants les uns des autres. SPIP se charge de gérer les différentes connexions et déclarations de tables sans imposer de programmation à ses utilisateurs. Le présent article expose toutes les manipulations de bases de données offertes par cette dernière version.

### Installation minimale

Voici d'abord tous les scénarios possibles d'installation de SPIP avec une seule base.

Lors de l'installation de SPIP, celui-ci teste la configuration de PHP et propose, lorsque c'est possible, un choix parmi plusieurs types de serveurs SQL (actuellement MySQL, PostgreSQL ou SQLite), qui offrent tous les mêmes fonctionnalités. À ce stade, il faut également fournir l'adresse Internet du serveur SQL, un identifiant de connexion à ce serveur et son mot de passe associé. Ces informations sont en général à saisir dans le formulaire d'installation. Toutefois, si vous voulez mettre en œuvre la Mutualisation du noyau SPIP, vous pouvez indiquer une ou plusieurs de ces valeurs dans le fichier de configuration `mes_options.php`. SPIP ne les demandera alors pas, ce qui dispense de les saisir et ne vous oblige pas à les communiquer à ceux qui utiliseront votre installation mutualisée. Voici les noms des constantes PHP à définir pour cela :

`_INSTALL_SERVER_DB` *type du serveur SQL* (Mysql ou PG ; casse indifférente)
`_INSTALL_HOST_DB` *nom Internet du serveur SQL (par exemple : localhost)*
`_INSTALL_USER_DB` *un nom d'utilisateur du serveur SQL*
`_INSTALL_PASS_DB` *son mot de passe*

SPIP se connecte alors au serveur choisi, avec les identifiants de connexion. En cas de réussite, il teste si l'utilisateur a le droit de créer des bases, ou seulement le droit de voir celles présentes et d'en choisir une, voire n'a le droit que d'utiliser une base homonyme de son nom d'utilisateur. Vous pouvez également fixer d'avance dans le fichier `mes_options.php` le nom de la base de données à utiliser, en définissant la constante PHP :

`_INSTALL_NAME_DB` *nom de la base*

SPIP poursuit alors l'installation en créant ses tables SQL (ou, en cas de réinstallation, en vérifiant que celles présentes sont utilisables). Ces tables commencent toutes par un *préfixe de table* (qui vaut par défaut `spip_`) et peut être indiqué dans `mes_options.php` de deux manières :

*la variable globale* `$table_prefix`
*la constante* `_INSTALL_TABLE_PREFIX`

Ce préfixe de table est ce qui permet d'écrire des boucles abrégées dans les squelettes, comme

```
<BOUCLE1(ARTICLES)....
```

alors que le nom SQL de la table est en fait `spip_articles`.

SPIP demande ensuite de créer le premier utilisateur du site en fournissant un nom et un mot de passe (sans rapport avec ceux du serveur SQL) ou en proposant de déléguer le service d'authentification à un serveur LDAP. Après cette déclaration, la procédure minimale d'installation est terminée, et la page suivante est la page habituelle d'entrée dans l'espace de rédaction, qui demande les identifiants venant d'être saisis. Ceux-ci ont été sauvés dans un fichier qui par défaut est `config/connect.php`.

## Compléments d'installation

Voici à présent comment indiquer à SPIP plusieurs bases à exploiter, sous le même serveur ou sous des serveurs différents, et même sur des machines différentes.

Une fois le site installé, choisir le sous-menu *maintenance du site* dans le menu *configuration* en haut de l'espace privé. Ce menu fournit une page à trois onglets ; cliquer sur celui de droite, nommé *déclarer une autre base*. On accède alors à un formulaire quasiment identique au formulaire d'installation : il propose d'indiquer un type de serveur SQL, son adresse Internet, un nom d'utilisateur et son mot de passe. En cas de connexion réussie, SPIP demandera de fournir le nom d'une base, en listant celles présentes s'il en a la possibilité. Si elle existe bien, il créera un fichier de connexion supplémentaire, qui par défaut se trouve dans le répertoire `config` et porte le nom de la base de données indiquée.

Cela fait, SPIP revient à nouveau à ce formulaire, et liste les fichiers de connexion présents, autrement dit les bases accessibles. SPIP propose donc de déclarer encore une base supplémentaire : il n'y a pas de limitation au nombre de bases déclarables.

L'intérêt de ces déclarations est de pouvoir appliquer la notation des squelettes de mise en pages à ces bases supplémentaires. S'il existe un fichier de connexion nommé *B*`.php`, permettant d'interroger une base possédant une table nommée *T*, alors le squelette

```
<BOUCLE1(B:T) />#TOTAL_BOUCLE<//B1>
```

donnera le nombre de lignes de cette table. De plus, SPIP demande au serveur SQL de lui *décrire* cette table, ce qui lui permet de connaître ses champs et, éventuellement, sa *clé primaire*. Si la boucle *T* a une clé primaire nommée `id` et un champ nommé `nom`, alors le squelette :

```
<BOUCLE1(B:T){id}>#NOM</BOUCLE1>
```

donnera le champ `nom` de l'unique ligne ayant l'`id` donné par le contexte.

Enfin, il est possible d'appliquer un squelette en demandant que toutes ses boucles utilisent une autre base que celle par défaut, en ajoutant dans l'URL de la page le paramètre `connect` indiquant le nom de la base. Par exemple `http://monsite?connect=autre_site` appliquera le squelette `sommaire` standard du site `monsite` à la base indiquée par le fichier de connexion `config/autre_site.php` dans le répertoire d'installation de `mon_site`.

Il est recommandé de respecter la casse du nom du fichier de connexion, lorsqu'on l'utilise dans une boucle ou comme paramètre de `connect` : aucune conversion majuscules / minuscules n'est effectuée par SPIP (mais certains systèmes de fichiers le font).

*Remarque :* L'accès à des bases supplémentaires est exclusivement en lecture ; en particulier, si une base est par exemple un forum (administré par SPIP ou non) il reste impossible de poster dans ce forum autrement qu'à partir de son site d'origine. La levée de cette restriction est à l'étude.

## Installations croisées

Les extraits de squelette ci-dessus reposent donc sur l'existence d'un fichier de connexion nommé *B*`.php`. Un point important est que le fichier de connexion principal et ceux des bases supplémentaires ont le même format, ce qui permet en particulier à un site SPIP d'utiliser directement le fichier de connexion d'un autre site SPIP pour exploiter ses tables. Une manière de procéder est de copier le fichier `connect.php` du site B sous le nom B`.php` dans le répertoire `config` du site A.

Une manière plus astucieuse, dans le cas de sites partageant une même installation de SPIP, est d'avoir un seul répertoire `config` pour tous les sites, et d'y nommer le fichier de connexion du site A non pas `connect.php` mais A`.php`, de même pour le site B etc. De la sorte, dès qu'un site est installé sous SPIP, il est connu comme base supplémentaire de tous les autres sites partageant cette installation de SPIP, et il verra tous les sites utilisant le même répertoire `config` que lui comme autant de bases supplémentaires. Pour obtenir cet optimum dans les déclarations implicites, il faut définir soigneusement les constantes `_DIR_CONFIG` et `_FILE_CONNECT_INS`. On trouvera plus bas un exemple de mutualisation offrant cette fonctionnalité et quelques autres.

Pour un site installé sur un serveur SQL éloigné, la copie de son fichier de connexion sur le site local peut suffire en théorie. Toutefois beaucoup d'hébergeurs refusent que leurs serveurs SQL soient interrogés par des machines en dehors de leur réseau local. Il faut aussi penser que bien souvent le « nom » indiqué pour le serveur SQL à l'installation est `localhost`, qui est une adresse relative au serveur HTTP, alors qu'ici il y a nécessité d'une adresse absolue. En bref, SPIP peut opérer dans une telle architecture, mais un minimum de connaissances, voire d'autorisation d'intervention, sur la topographie des sous-réseaux en jeu est pratiquement indispensable.

## Les bases supplémentaires sous SPIP

Lorsqu'un fichier de connexion est celui d'un autre site sous SPIP (que ce soit une copie ou l'original accessible par installations croisées), il contient l'indication que ce site est sous SPIP (la globale `spip_connect_version` y est affectée). Dans ce cas, les squelettes du site principal vont s'appliquer avec un comportement spécial :

- les boucles avec nom abrégé seront interprétées comme abrégées aussi dans la base supplémentaire ; autrement dit, `<BOUCLE1(B:ARTICLES)...` (ou `<BOUCLE1(ARTICLES)...` avec une URL comportant `connect=B.`) référencera la table `spip_articles` dans la base B, plus précisément la table *prefixe*`articles` où *prefixe* est celui utilisé par le site B (ce préfixe est indiqué dans le fichier de connexion) ;

- à l'intérieur d'une boucle, les balises `#URL_` (voir « Utiliser des URLs personnalisées ») utiliseront le type d'URL du site principal, et non du site distant, et l'URL produite référencera le site principal par `connect=`*site_distant* ; cette stratégie permet de naviguer dans la base du site distant sans cesser d'utiliser les squelettes du site principal, autrement dit de tester l'apparence que donne ce jeu de squelettes et son type d'URL sur le site distant sans modifier l'installation de celui-ci ou travailler sur une copie.
- Les raccourcis SPIP des balises #URL qui peuvent figurer dans les champs de la base de données distante sont eux aussi interprétés de cette manière : `[->art3681]` référencera bien l'article 3681 de la base distante, mais à travers une URL qui fournira la mise en page par les squelettes du site principal.

Ce mode de fonctionnement permet donc de présenter d'autres bases SPIP sans même avoir à écrire des squelettes de mises en page, puisque les noms des tables dans ceux-ci sont les mêmes que ceux du site distant. En revanche, une base qui n'est pas sous SPIP a besoin de squelettes de mises en page nommant explicitement les tables de cette autre base et ses champs. Pour répondre à ce besoin, les administrateurs du site bénéficient d'un traitement spécial : lorsqu'ils donnent à leur navigateur l'URL de leur site suivi de `?page=table:`*table* où *table* est le nom d'une table de la base, SPIP va automatiquement construire un squelette spécifique à cette table, permettant d'en examiner le contenu avec une grande ergonomie. Le squelette produit est affichable à travers le lien *squelette* en bas de page. Il est possible alors de le copier à la souris (cliquer en haut de colonne pour cacher la numérotation des lignes) et de l'améliorer en le travaillant sous un éditeur approprié.

À noter qu'avec cette production automatique, SPIP pourrait, à la limite, fonctionner sans aucun squelette prédéfini.

## Sauvegardes et Fusions

Avec le sous-menu **maintenance du site**, SPIP donne accès depuis toujours à deux outils de sauvegarde et restauration de la base de données locale (voir Sauvegarder vos données).

Il est possible d'installer une sauvegarde issue d'un vieux site, dans une installation de SPIP de numéro de version supérieur ; toutefois ce cas exige un espace mémoire qui n'est pas toujours disponible, aussi il est toujours conseillé d'essayer de mettre à jour le site d'origine, puis d'en faire la sauvegarde, et enfin de la faire relire par le nouveau site.

D'autre part, jusqu'à la version 1.8, une sauvegarde était totale, et la restauration également. Après quelques essais fondés sur le statut d'administrateur restreint, qui se sont révélés finalement malcommodes, SPIP 2.0 offre une nouvelle fonctionnalité plus harmonieuse de sauvegarde partielle et de restauration par fusion.

Le formulaire de sauvegarde propose de limiter la sauvegarde à une rubrique donnée. Le fichier produit contiendra les articles, sites, brèves et sous rubriques contenu dans la rubrique indiquée, ainsi que les mots-clés et groupes de mots-clés utilisés par toutes ces données. Le nom de la sauvegarde sera par défaut le nom de la rubrique, mais cela peut être changé. La création des fichiers de sauvegarde peut être si longue que le serveur HTTP peut vous déconnecter pour cause d'inactivité apparente. Recharger simplement la page : SPIP retrouvera à quel endroit il en était et poursuivra son travail. Le grand nombre de fichiers que vous pourrez temporairement apercevoir dans un sous-répertoire du répertoire `tmp` est normal, car lié à l'anticipation de cette situation de reprise après déconnexion.

Symétriquement, le formulaire de restauration propose à présent deux fonctionnalités. La première est l'habituel remplacement de la base courante par la base sauvegardée. La deuxième consiste à considérer le fichier de sauvegarde comme une base incomplète venant s'ajouter à la base déjà installée. Comme il peut y avoir conflit entre la base installée et la base incomplète en ce qui concerne la numérotation de leurs tables (d'articles, de brèves etc), SPIP commence par renuméroter les éléments de la base incomplète en leur affectant des numéros immédiatement supérieurs aux maxima de la base installée. Dans une deuxième passe, SPIP importe effectivement les éléments renumérotés, mais lors de la première passe, il a comparé les éléments à importer avec ceux de la base installée et a ignoré les doublons définis par les règles suivantes :

- s'il existe dans la base installée un groupe de mots-clés de même nom qu'un groupe de mots-clés à importer, ce groupe est ignoré ;
- si un mot-clé dont le groupe de mot-clés n'a pas été importé possède un homonyme dans un groupe de mot-clés portant le même nom que son groupe d'origine, ce mot-clé est ignoré ;
- s'il existe dans la base installée une rubrique de secteur portant même nom qu'une rubrique de secteur à importer, cette rubrique est ignorée ;
- si une sous-rubrique dont la rubrique parente n'a pas été importée possède un homonyme de même parenté, cette sous-rubrique est ignorée ; il en est de même pour les articles, les brèves et les sites référencés ;
- s'il existe dans la base installée un document portant même nom qu'un document à importer et qu'ils ont même taille, ce document est ignoré.

Dit de manière plus synthétique, l'opération de fusion est l'union de deux bases, le contenu de la base installée ayant priorité sur la base importée en ce qui concerne les doublons (par exemple, pour deux secteurs de même nom, les champs *texte* et *descriptif* de celui déjà installé seront conservés et les autres seront ignorés).

En ce qui concerne les documents joints, ceux-ci seront importés en tant que documents distants, ce qui dispense de recenser et copier la partie du répertoire IMG/ concernée par la sauvegarde partielle. On peut ensuite utiliser, sur chaque document, le bouton de copie locale pour s'affranchir du site d'origine. Si celui-ci a déjà disparu ou possède des restrictions d'accès non contournables par le site récepteur, il faudra installer le répertoire IMG/ originel à une URL accessible, et la donner dans la dernière case de saisie du formulaire d'importation.

On notera la possibilité de rabattre les statuts de valeur *publié* à la valeur *proposé*, ce qui permet de garder une politique de publication au cas par cas, malgré cette importation massive.

## Un exemple complet

L'installation mutualisée de SPIP ci-dessous permet d'avoir un seul répertoire de configuration (`_DIR_CONNECT`) et un seul de sauvegardes (`_DIR_DUMP`). Chaque site peut ainsi donner des aperçus de la base de chacun des autres, et utiliser leurs sauvegardes pour les fusionner avec sa propre base. N'est utilisé aussi qu'un seul répertoire pour le cache de l'aide en ligne (`_DIR_AIDE`) car SPIP l'importe au coup par coup à partir du serveur http://www.spip.net/, et un seul répertoire pour les *Document Type Definitions* que Le validateur XML intégré va chercher sur le site du W3C et autres (`_DIR_XML`).

Les fichiers de droits et de journalisation sont également rassemblés dans seulement deux répertoires (`_DIR_CHMOD` et `_DIR_LOG`).

Seront donc créés à la racine config/connect, config/chmod, tmp/log, tmp/dump, tmp/xml et tmp/aide. Le premier contiendra A.php pour la connexion à la base A, B.php etc.

```
if ( preg_match(',/([a-zA-Z0-9_-]*)[/?],',$_SERVER['REQUEST_URI'],$r)) {
        if (is_dir($e = _DIR_RACINE . 'Ajouts/' . $r[1]. '/')) {
                $cookie_prefix = $table_prefix = $r[1];

                define('_SPIP_PATH',
                _DIR_RACINE. 'Ajouts/' . $table_prefix  . '/dist/:' .
                _DIR_RACINE .'Ajouts/' . $table_prefix  . '/:' .
                _DIR_RACINE .'dist/:' .
                _DIR_RACINE .'dist/javascript/:' .
                _DIR_RESTREINT);

                $pi = $e . _NOM_PERMANENTS_INACCESSIBLES;
                $pa = $e . _NOM_PERMANENTS_ACCESSIBLES;
                $ti = $e . _NOM_TEMPORAIRES_INACCESSIBLES;
                $ta = $e . _NOM_TEMPORAIRES_ACCESSIBLES;

                $pig = _DIR_RACINE . _NOM_PERMANENTS_INACCESSIBLES;
                $tig = _DIR_RACINE . _NOM_TEMPORAIRES_INACCESSIBLES;

                define('_DIR_DUMP', $tig . 'dump/');
                define('_DIR_AIDE', $tig . 'aide/');
                define('_DIR_CACHE_XML', $tig . "xml/");
                define('_DIR_LOG', $tig . 'log/');
                define('_DIR_CONNECT', $pig . 'connect/');
                define('_DIR_CHMOD', $pig . 'chmod/');
                define('_FILE_CONNECT_INS', $table_prefix);
                define('_FILE_CHMOD_INS', $table_prefix);
                define('_FILE_LOG_SUFFIX',
                         '_' . $table_prefix . '.log');

                $GLOBALS['test_dirs'] =
                  array($pa, $ta, $ti, $pig, $tig,
                _DIR_DUMP, _DIR_LOG, _DIR_CONNECT, _DIR_CHMOD);

                spip_initialisation($pi, $pa, $ti, $ta);
        }
)
```

## P.-S.

*Point d'histoire*. Dans ses premières versions, SPIP ne donnait accès qu'à certains champs d'une unique base de données (celle qu'il créait à sa première installation). Tout juste était-il possible de simuler plusieurs bases SPIP dans une seule (mais qui s'ignoraient mutuellement) en préfixant le nom des tables de chaque base simulée avec un préfixe spécifique.

SPIP 1.8 et le nouveau compilateur de squelettes, ont ouvert l'accès à tous les champs de toutes les bases accessibles par le serveur HTTP. Mais cette possibilité exigeait d'écrire des fichiers PHP par imitation des fichiers standards, peu intuitifs. De plus, SPIP ne distinguait pas les notions de *type de serveurs* et de *base de données* : pour adresser plusieurs bases d'un même serveur, il fallait dupliquer ces fichiers et les adapter. Cette architecture très insatisfaisante explique pourquoi ces fonctionnalités n'ont jamais été documentées, bien que plusieurs extensions de SPIP s'en soient servi.

## L’interface de SPIP avec SQL

Décembre 2008 — maj : août 2010

Rappelons d’abord que le *Structured Query Langage* ne s’est standardisé que très progressivement, chaque implémentation comblant les lacunes de la spécification avec ses propres solutions. Les premières versions de SPIP ne connaissaient qu’une seule implémentation de SQL ce qui lui permettait d’en centraliser l’utilisation à travers une seule fonction (spip_query) dont l’unique argument était la requête. Le portage de SPIP sur différentes implémentations de SQL a imposé de renoncer à ce modèle ; cette fonction reste disponible par souci de compatibilité, mais son usage doit être désormais évité. On lui préférera le jeu de fonctions suivant, qui a de plus l’avantage de neutraliser la plupart des techniques d’attaque par injection de code.

Le premier portage de SPIP sur un autre serveur que MySQL3 a été réalisé pour le serveur PostGreSQL version 8.2. Il a été immédiatement suivi d’un double portage en SQLite 2 et 3. Ces portages se fondent sur le remplacement de la fonction spip_query par autant de fonctions que d’*instructions* SQL (select, update etc) ayant chacune autant d’arguments que l’instruction admet de clauses (where, limit etc), plus un argument optionnel précisant la base SQL. Les *opérandes* SQL (en particulier les dates et les nombres en hexadécimal) restent écrits en syntaxe MySQL, les fonctions d’interface se chargeant de les réécrire au besoin. Quelques fonctions supplémentaires sont proposées : des indispensables (l’accès aux lignes successives d’un résultat de **select**) et des abréviations (décompte, accès à une ligne qu’on sait unique etc). Des informations générales sont également fournies : alphabets utilisés, présence d’un LDAP etc.

Le double portage en SQLite n’a pas nécessité de réviser le jeu de fonctions défini d’abord pour le seul portage PostGres, ce qui tend à prouver la perennité de l’interface ainsi définie. Toutefois, de tels outils sont tôt ou tard condamnés à évoluer, aussi cette nouvelle l’interface proposée avec SPIP 2.0 intègre dès à présent un gestionnaire de ses propres versions, lui permettant d’utiliser une connexion SQL avec plusieurs versions de l’interface *simultanément*. Ainsi, les extensions de SPIP respectant cette nouvelle interface sont-elles assurées de fonctionner dans les versions ultérieures de SPIP.

Signalons enfin que le choix de développer cette architecture a été imposé par l’inexistence d’une bibliothèque de niveau d’abstraction équivalent disponible aussi librement que SPIP : les extensions de PHP couramment offertes se contentent d’uniformiser les appels aux fonctions de base de celui-ci, ce qui n’est que la partie émergée de l’iceberg auquel est confronté le développeur d’applications sur bases de données hétérogènes.

Cet article comporte trois parties. Les deux premières sont destinées aux développeurs désireux d’écrire des extensions de SPIP ; elles présentent l’architecture générale, puis les fonctions disponibles. La troisième détaille l’implémentation et est destinée aux contributeurs de SPIP, désireux de le porter sur d’autres implémentations de SQL ou d’améliorer les portages existants. Dans tout l’article, on parlera de *serveur SQL* pour désigner une implémentation de SQL utilisée par SPIP, bien qu’à proprement parler certaines implémentations ne sont pas des serveurs.

### Architecture générale

Dans un premier temps, on peut considérer que l’interface de SPIP aux implémentations de SQL se réduit à l’unique fonction suivante, définie dans le fichier `ecrire/base/abstract_sql.php` :

```
sql_serveur($ins_sql, $serveur='', $continue=false)
```

Cette fonction commence, si elle ne l'a pas déjà fait auparavant, par se connecter au serveur spécifié en deuxième argument. Cet argument est souvent omis, ce qui désigne alors l'implémentation SQL choisie à l'installation, et mémorisé par SPIP dans un *fichier de connexion* (voir Les bases de données en SPIP). Sinon, l'argument doit indiquer explicitement le nom du fichier de connexion à utiliser, l'extension `.php` étant omise. Le résultat retourné est une autre fonction, réalisant le type d'action demandée par la chaîne passée en premier argument (par exemple `select` ou `fetch`) sur la base SQL indiquée par le fichier de connexion. Le troisième argument indique ce qu'il faut faire lorsqu'une telle fonction ne peut être retournée. S'il est absent ou égal à `false`, une erreur fatale sera déclenchée. Autrement, deux autres situations sont distinguées. Si la connexion indiquée est inconnue ou inopérante, la valeur `false` sera retournée. Autrement, sera retournée une structure de données décrivant la connexion (elle sera décrite dans la dernière partie), ce qui permet d'une part de vérifier qu'une fonction existe sans risquer l'erreur fatale, et d'autre part d'obtenir plusieurs informations avant utilisation.

Cette vision minimaliste de l'interface permet de tout faire, mais avec une syntaxe assez opaque. Si par exemple `$db` est le nom d'une base dans le serveur principal, on ne le sélectionnera pas

```
$f = sql_serveur('selectdb');
$f($db);
```

Pour clarifier l'écriture, il existe un jeu de fonctions enchaînant les deux instructions ci-dessus pour les cas les plus fréquents. Par exemple, il existe

```
sql_selectdb($nom, $serveur='')
```

Ce qui permet de réécrire plus simplement l'opération précédente :

```
sql_selectdb($db)
```

De manière générale, l'interface de SPIP aux serveurs SQL est un jeu de fonctions dont le nom est `sql_`*f* et dont le dernier argument, optionnel, est le nom du serveur. Elles appellent `sql_serveur` pour obtenir une fonction *f* qu'elles appliquent sur leurs arguments, y compris le nom du serveur. Toutes les fonctions dont le nom est ainsi construit sont réservées à l'interface de SPIP aux serveurs SQL.

Dans une vision *orientée objet*, ce jeu de fonctions représente les *méthodes* de l'objet `sql`, mais on écrit `sql_`*f* à la place de `sql->`*f*, et il n'est pas nécessaire d'*instancier une classe*. La présence du nom du serveur dans tous les appels permet de simuler les opérations impliquant l'objet *moi-même*.

Ce jeu de fonctions dispense donc la plupart du temps d'utiliser `sql_serveur`. Signalons également la fonction (présente dans SPIP de longue date) :

```
spip_connect($serveur='')
```

qui simplement ouvre la connexion au serveur, et est donc équivalente à

```
sql_serveur('', $serveur, true)
```

et qui retourne *false* si le serveur est indisponible, sinon la structure de données décrivant les possibilités du serveur.

## Fonctions disponibles

Les fonctions de l'interface SQL peuvent se classer en plusieurs groupes, pour lesquels on donnera à chaque fois un tableau présentant leurs arguments. Pour les exemples on se reportera au code de SPIP.

Un premier groupe de fonctions concernent la lecture des tables SQL. Les fonctions incontournables sont :

- `sql_select` dont les arguments sont les clauses SQL habituelles de cette instruction, et dont le résultat est une *ressource Select* ;

- `sql_fetch`, utilisée presque toujours dans une boucle, qui permet de récupérer les lignes successives d'une ressource Select ; son résultat est un tableau indexé par le nom des champs ;

- `sql_free` qui signale au serveur de libérer une ressource.

D'autres fonctions offrent des compositions fréquentes de ces opérations :

- `sql_fetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée ; pratique pour les requêtes dont le résultat ne comporte qu'une ligne ;

- `sql_getfetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée, et enfin extrait du tableau retourné le champ dont le nom est donné comme premier argument (la liste des champs n'en comporte donc qu'un) ; pratique pour les requêtes dont le résultat ne comporte qu'une ligne d'un seul champ ;

- `sql_allfetsel` de mêmes arguments que `sql_select`, qui applique celle-ci sur ses arguments puis `sql_fetch` sur la ressource retournée tant que celui-ci retourne un tableau non vide ; `sql_allfetsel` retourne pour finir le tableau de tous les tableaux retournés par `sql_fetch` (attention à ne pas saturer la mémoire avec cette fonction) ;

- `sql_countsel` de mêmes arguments que `sql_select` moins le premier, qui applique celle-ci sur `COUNT(*)` et ses autres arguments et retourne le nombre calculé ; pratique pour connaître le nombre de lignes que retournerait la requête Select ainsi décrite.

- `sql_count` qui retourne le nombre de lignes d'une ressource Select, comme si on avait ajouté `COUNT(*)` dans la requête ;

- `sql_get_select` utilise les mêmes arguments que `sql_select` , mais retourne le code SQL de la requête, sans l'exécuter. Cette fonction peut-être utile pour les besoins de certains plugins ou pour créer facilement des vues SQL.

Les quatre premières appellent pour finir `sql_free`, et sont donc à préférer autant que possible au trio `select-fetch-free` dont on oublie facilement le dernier membre.

Le tableau ci-dessous précise le contenu et l'ordre des arguments attendus par ces fonctions.

| Fonction | Arguments |
|---|---|
| sql_select | 1. liste des champs : chaîne ou tableau<br>2. liste des tables : chaîne ou tableau<br>3. clause Where : chaîne ou tableau<br>4. clause Groupby : chaîne avec virgule séparatrice ou tableau<br>5. clause Orderby : chaîne avec virgule séparatrice ou tableau<br>6. clause Limit : un ou deux entiers séparés par des virgules<br>7. clause Having : chaîne ou tableau<br>8. serveur |
| sql_fetch | 1. ressource<br>2. serveur |
| sql_free | 1. ressource<br>2. serveur |
| sql_count | 1. ressource<br>2. serveur |
| sql_countsel | 1. liste des tables : chaîne ou tableau<br>2. clause Where : chaîne ou tableau<br>3. clause Groupby : chaîne ou tableau<br>4. clause Orderby : chaîne avec virgule séparatrice ou tableau<br>5. clause Limit : un ou deux entiers séparés par des virgules<br>6. clause Having : chaîne ou tableau<br>7. serveur |
| sql_fetsel | 1. liste des champs : chaîne ou tableau<br>2. liste des tables : chaîne ou tableau<br>3. clause Where : chaîne ou tableau<br>4. clause Groupby : chaîne avec virgule séparatrice ou tableau<br>5. clause Orderby : chaîne avec virgule séparatrice ou tableau<br>6. clause Limit : un ou deux entiers séparés par des virgules<br>7. clause Having : chaîne ou tableau<br>8. serveur |
| sql_allfetsel | 1. liste des champs : chaîne ou tableau<br>2. liste des tables : chaîne ou tableau<br>3. clause Where : chaîne ou tableau<br>4. clause Groupby : chaîne avec virgule séparatrice ou tableau<br>5. clause Orderby : chaîne avec virgule séparatrice ou tableau<br>6. clause Limit : un ou deux entiers séparés par des virgules<br>7. clause Having : chaîne ou tableau<br>8. serveur |
| sql_getfetsel | 1. nom d'un champ : chaîne<br>2. liste des tables : chaîne ou tableau<br>3. clause Where : chaîne ou tableau |

4. clause Groupby : chaîne ou tableau
5. clause Orderby : chaîne avec virgule séparatrice ou tableau
6. clause Limit : un ou deux entiers séparés par des virgules
7. clause Having : chaîne ou tableau
8. serveur

sql_get_select

1. liste des champs : chaîne ou tableau
2. liste des tables : chaîne ou tableau
3. clause Where : chaîne ou tableau
4. clause Groupby : chaîne avec virgule séparatrice ou tableau
5. clause Orderby : chaîne avec virgule séparatrice ou tableau
6. clause Limit : un ou deux entiers séparés par des virgules
7. clause Having : chaîne ou tableau
8. serveur

Dans les fonctions ci-dessus, si la clause Select est fournie sous forme de tableau, ses éléments seront concaténés, séparés par des virgules. En cas de tableau pour les clauses Where et Having, les éléments doivent être des chaînes (des sous-tableaux en notation préfixée sont également pris en charge, mais réservés au compilateur). Ces chaînes seront réunies en une grande conjonction (i.e, elles seront concaténées avec AND comme séparateur).

La clause From est une chaîne (le cas du tableau est réservé au compilateur de SPIP). Attention : s'il est nécessaire de référencer les tables dans les autres clauses, il faut en définir des alias dans ce paramètre et les utiliser systématiquement. Ainsi, on écrira :

```
sql_countsel('spip_articles AS a, spip_rubriques AS r',
"a.id_secteur=r.id_rubrique AND r.titre='monsecteur')
```

ou

```
sql_countsel('spip_articles AS a JOIN spip_rubriques AS r ON
a.id_secteur=r.id_rubrique", "r.titre='monsecteur'")
```

alors que l'écriture suivante ne sera pas comprise :

```
sql_countsel('spip_articles, spip_rubriques',
"spip_articles.id_rubrique=spip_rubriques.id_secteur AND
spip_rubriques.titre='monsecteur'")
```

Un deuxième groupe de fonctions est constitué par celles modifiant le contenu des tables. Ces fonctions sont délicates à définir car la syntaxe des valeurs à introduire dans les tables change d'un serveur SQL à un autre (notamment les dates). Pour cette raison, ces fonctions doivent disposer de la description de la table à modifier, afin de connaître le type des valeurs attendues par le serveur SQL. SPIP retrouve automatiquement ces informations (données au moment de la création de la table) mais il est possible de fournir une description arbitraire (avant-dernier argument de ces fonctions, optionnel et d'ailleurs rarement utile).

SPIP fournit donc une fonction d'insertion, `sql_insertq`, et une fonction de mise à jour, `sql_updateq`, qui prennent un tableau *champ=>valeur* et s'occupent de *citer* les valeurs en fonction du type (avec la fonction `sql_quote` spécifiée ci-dessous). Est également disponible `sql_insertq_multi` permettant de faire des insertions de plusieurs entrées en prenant un

tableau de tableau *champ=>valeur*. Pour les mises à jour où les nouvelles valeurs dépendent des anciennes (comme dans `cpt=cpt+1`), utiliser `sql_update` où les valeurs seront prises littérallement, mais il faudra interdire soigneusement les possibilités d'attaque par injection de code. Il existe également `sql_replace`, fonction effectuant une mise à jour sur une ligne correspondant à une clé primaire, ou insérant les valeurs si cette ligne n'existe pas ainsi qu'une fonction `sql_replace_multi` pour des mises à jour ou insertions multiples. Enfin, `sql_delete` efface d'une table les lignes répondant à une clause Where.

| Fonction | Arguments |
|---|---|
| sql_updateq | 1. table<br>2. tableau *champ=>valeur à citer*<br>3. clause Where<br>4. description<br>5. serveur |
| sql_update | 1. table<br>2. tableau *champ=>valeur*<br>3. clause Where<br>4. description<br>5. serveur |
| sql_insertq | 1. table<br>2. tableau *champ=>valeur à citer*<br>3. description<br>4. serveur |
| sql_insertq_multi | 1. table<br>2. tableau de tableau *champ=>valeur à citer*<br>3. description<br>4. serveur |
| sql_replace | 1. table<br>2. tableau *champ=>valeur à citer*<br>3. description<br>4. serveur |
| sql_replace_multi | 1. table<br>2. tableau de tableau *champ=>valeur à citer*<br>3. description<br>4. serveur |
| sql_delete | 1. table<br>2. clause Where<br>3. serveur |

Un groupe un peu à part est formé de fonctions traitant spécifiquement des opérandes. : elles ne se connectent pas au serveur, mais retournent des chaînes dépendant de celui-ci :

- `sql_quote` prend une chaîne ou un nombre, retourne un nombre si l'argument était un nombre ou une chaîne représentant un entier, sinon retourne la chaîne initiale entourée d'apostrophes et avec les apostrophes protégées selon la syntaxe propre au serveur (un \ devant pour certains, une deuxième apostrophe pour d'autres) ;
- `sql_hex` prend une chaîne de chiffres hexadécimaux et retourne sa représentation dans le serveur SQL visé ;
- `sql_in` construit un appel à l'opérande IN, en traitant les éventuelles valeurs hexadécimales y figurant ;
- `sql_test_int` prédicat retournant Vrai si le type SQL fourni désigne un entier ;
- `sql_test_date` prédicat retournant Vrai si le type SQL fourni désigne une date ;
- `sql_multi` applique une expression SQL sur champ contenant un *bloc multi* (voir Réaliser un site multilingue) pour y prendre la partie correspondant à la langue indiquée ; l'intérêt d'effectuer cette opération au niveau SQL est essentiellement de demander simultanément un tri sur cette colonne.

| Fonction | Arguments |
|---|---|
| sql_quote | 1. valeur<br>2. serveur |
| sql_hex | 1. valeur<br>2. serveur |
| sql_in | 1. colonne<br>2. valeurs<br>3. vrai si négation souhaitée<br>4. serveur |
| sql_multi | 1. colonne<br>2. langue<br>3. serveur |
| sql_test_date | 1. type<br>2. serveur |
| sql_test_int | 1. type<br>2. serveur |

Un groupe important est constitué par les fonctions manipulant les déclarations de bases et de tables. Pour des raisons historiques, cette première version de l'interface reprend quasi littéralement la syntaxe de MySQL3 et devra certainement être revue, en particulier pour y faire apparaître la déclaration des jointures. Les fonctions `sql_create`, `sql_alter`, `sql_showtable` et `sql_drop_table` permettent de créer, modifier, voir et supprimer une table. Les fonctions `sql_create_view`, `sql_drop_view` permettent de créer ou supprimer une vue. Les fonctions `sql_listdbs`, `sql_showbase` et `sql_selectdb` permettent de voir les bases accessibles, de voir leur contenu et d'en sélectionner une. À noter que tous les hébergeurs n'autorisent pas forcément de telles actions ; SPIP essaiera de le deviner, en notant ses essais dans le fichier `spip.log`.

| Fonction | Arguments |
|---|---|
| sql_create | 1. nom de la table<br>2. tableau nom de colonne => type SQL et valeur par défaut<br>3. tableau nom d'index => colonne(s)<br>4. vrai si auto-incrément<br>5. vrai si temporaire<br>6. serveur |
| sql_alter | 1. requête MYSQL Alter<br>2. serveur |
| sql_showtable | 1. RegExp<br>2. vrai si table déclarée par SPIP<br>3. serveur |
| sql_drop_table | 1. nom de la table<br>2. vrai s'il faut insérer la condition *existe*<br>3. serveur |
| sql_create_view | 1. nom de la vue<br>2. requête de sélection de champs (créé par exemple avec `sql_get_select`)<br>3. serveur |
| sql_drop_view | 1. nom de la vue<br>2. vrai s'il faut insérer la condition *existe*<br>3. serveur |
| sql_listdbs | |
| sql_selectdb | 1. nom de la base<br>2. serveur |
| sql_showbase | 1. RegExp<br>2. serveur |

Deux fonctions permettent de régler le codage des caractères lors des communications avec le serveur :

- `sql_set_charset`, demande d'utiliser le codage indiqué ;
- `sql_get_charset`, demande si un codage de nom donné est disponible sur le serveur.

| Fonction | Arguments |
|---|---|
| sql_get_charset | 1. RegExp<br>2. serveur |
| sql_set_charset | 1. codage<br>2. serveur |

Un dernier groupe de fonctions offre quelques outils de gestion des requêtes et des tables ; on se reportera à leurs homonymes dans la documentation des serveurs SQL. Signalons toutefois que `sql_explain` est utilisée implicitement par le débusqueur de SPIP, accessible par les boutons d'administration de l'espace public, lorsqu'on lui demande le plan de calcul d'une boucle (ou de tout un squelette).

| Fonction | Arguments |
|---|---|
| sql_optimize | 1. requête<br>2. serveur |
| sql_repair | 1. table<br>2. serveur |
| sql_explain | 1. requête<br>2. serveur |
| sql_error | 1. requête<br>2. serveur |
| sql_errno | |
| sql_version | |

Hors groupe, la fonction générale `sql_query`, nom qu'aurait dû porter l'historique `spip_query` ; leur utilisation est de toute façon à éviter.

## Réalisation des portages

Cette section est destinée à ceux souhaitant porter SPIP sur d'autres serveurs SQL, ou ayant besoin d'informations plus techniques, notamment sur la gestion des versions de l'interface. Les fonctions du fichier `ecrire/base/abstract_sql.php` étudiées ci-dessus se contentent d'offrir une interface homogène aux différents serveurs, mais ne procèdent elles-mêmes à aucun calcul. C'est dans le fichier `ecrire/base/connect_sql.php` que se situe le travail effectif.

La fonction essentielle est `spip_connect` qui ouvre la connexion au serveur SQL indiqué par son argument (ou, s'il est omis, le serveur principal) en repérant les connexions déjà faites. Cette ouverture consiste à inclure un *fichier de connexion* créé lors de l'installation de SPIP par les scripts présents dans le répertoire `install`. Un fichier de connexion se réduit pour l'essentiel à appliquer la fonction `spip_connect_db` aux valeurs fournies lors de l'installation.

La fonction `spip_connect_db` reçoit en particulier comme argument le type du serveur SQL. Ce type doit être le nom d'un fichier présent dans le répertoire `req`. Ce fichier est chargé et doit définir toutes les fonctions d'interfaces définies à la section précédente, plus la fonction `req_`*type*`_dist` qui sera immédiatement appliquée sur les mêmes arguments que `spip_connect_db`, type excepté. C'est cette fonction qui doit établir effectivement la connexion.

Porter SPIP sur d'autres serveurs SQL consiste donc à définir ce jeu de fonctions et à le placer dans le répertoire `req`.

Le gestionnaire de versions d'interface repose sur le deuxième argument de `spip_connect` qui indique la version, la version courante étant prise par défaut. Toutes les fonctions de l'interface sont définies dans le fichier `abstract_sql`, se nomment sql_*X* et sont les seules à se nommer ainsi. Elles se connectent toutes en appelant une variante de `spip_connect` dont le premier argument est le numéro de version de l'interface. Au cas où le fichier `abstract_sql` nécessiterait une révision, il sera renommé abstract_sql_*N*, et le Sed suivant lui sera appliqué (*N* désigne le numéro de version) :

```
s/\(sql_[A-Za-z_0-9 ]*\)/\1_N/
```

En appliquant également ce script aux extensions de SPIP fondées sur cette version, on leur permettra d'en appeler ses fonctions, qui seront chargées sans collision de noms, le Sed ayant préfixé le nom des anciennes avec leur numéro de version. Il faudra juste rajouter une instruction include portant sur le fichier abstract_sql_*N*. Du côté du portage, il faudra renommer pareillement les fichiers du répertoire `req`, et écrire les nouvelles versions.

La coexistence de plusieurs versions de l'interface pendant l'exécution de SPIP repose sur la structure décrivant le serveur. Celle-ci est en fait un tableau, contenant :

- `link`, *ressource* indiquant la connexion ;

- `db`, nom de la base de données ;

- `prefixe`, nom du préfixe de table ;

- `ldap`, nom de l'éventuel fichier décrivant le serveur LDAP.

Les autres entrées sont les numéros de versions disponibles, et leur valeur est le tableau des fonctions implémentant cette version de l'interface.

Les autres fonctions du fichier `connect_sql` concernent essentiellement la gestion des versions et le traitement de quelques cas particuliers de déclarations des tables standards de SPIP.

## La structure de la base de données

Juin 2001 — maj : Décembre 2008

La structure de la base de données est assez simple. Certaines conventions ont été utilisées, que vous repérerez assez facilement au cours de ce document. Par exemple, la plupart des objets sont indexés par un entier autoincrémenté dont le nom est du type `id_objet`, et qui est déclaré comme clé primaire dans la table appropriée.

*NB : cet article commence à dater, et personne n'a encore pris la peine d'en faire la mise à jour. Il faut le lire comme un élément permettant de comprendre le fonctionnement de SPIP, mais plus comme un outil de référence. Si vous souhaitez contribuer à la documentation en refondant cet article, surtout n'hésitez pas !*

### Contenu rédactionnel

#### Les rubriques : spip_rubriques

| | |
|---|---|
| id_rubrique | bigint(21) |
| id_parent | bigint(21) |
| titre | text |
| descriptif | text |
| texte | longblob |
| id_secteur | bigint(21) |
| maj | timestamp(14) |
| export | varchar(10) |
| id_import | bigint(20) |

- Chaque rubrique est identifiée par son **id_rubrique**.
- **id_parent** est l'*id_rubrique* de la rubrique qui contient cette rubrique (zéro si la rubrique se trouve à la racine du site).
- **titre**, **descriptif**, **texte** parlent d'eux-mêmes.
- **id_secteur** est l'*id_rubrique* de la rubrique en tête de la hiérarchie contenant cette rubrique. Une rubrique dépend d'une rubrique qui dépend d'une rubrique... jusqu'à une rubrique placée à la racine du site ; c'est cette dernière rubrique qui détermine l'*id_secteur*. Cette valeur précalculée permet d'accélérer certains calculs de l'espace public (en effet, les brèves sont classées par secteur uniquement, et non selon toute la hiérarchie).
- **maj** est un champ technique mis à jour automatiquement par MySQL, qui contient la date de la dernière modification de l'entrée dans la table.
- **export**, **id_import** sont des champs réservés pour des fonctionnalités futures.

#### Les articles : spip_articles

| | |
|---|---|
| id_article | bigint(21) |
| surtitre | text |
| titre | text |
| soustitre | text |
| id_rubrique | bigint(21) |
| descriptif | text |
| chapo | mediumtext |
| texte | longblob |
| ps | mediumtext |
| date | datetime |
| statut | varchar(10) |
| id_secteur | bigint(21) |
| maj | timestamp(14) |
| export | varchar(10) |
| images | text |
| date_redac | datetime |
| visites | int(11) |
| referers | blob |
| accepter_forum | char(3) |

- Chaque article est identifié par son **id_article**.
- **id_rubrique** indique dans quelle rubrique est rangé l'article.
- **id_secteur** indique le secteur correspondant à la rubrique susmentionnée (voir le paragraphe précédent pour l'explication de la différence entre les deux).
- **titre**, **surtitre**, **soustitre**, **descriptif**, **chapo**, **texte**, **ps** parlent d'eux-mêmes.
- **date** est la date de publication de l'article (si l'article n'a pas encore été publié, c'est la date de création).
- **date_redac** est la date de publication antérieure si vous réglez cette valeur, sinon elle est égale à « `0000-00-00` ».
- **statut** est le statut actuel de l'article : `prepa` (en cours de rédaction), `prop` (proposé à la publication), `publie` (publié), `refuse` (refusé), `poubelle` (à la poubelle).
- **accepter_forum** : permet de régler manuellement si l'article accepte des forums (par défaut, oui).
- **maj** : même signification que dans la table des rubriques.

- **export** est un champ réservé pour des fonctionnalités futures.
- **images** est un champ contenant la liste des images utilisées par l'article, dans un format particulier. Ce champ est généré par `spip_image.php3`.
- **visites** et **referers** sont utilisés pour les statistiques sur les articles. Le premier est le nombre de chargements de l'article dans l'espace public ; le deuxième contient un extrait de hash des différents referers, afin de connaître le nombre de referers distincts. Voir `inc-stats.php3`.

**Les auteurs : spip_auteurs**

| | |
|---|---|
| id_auteur | bigint(21) |
| nom | text |
| bio | text |
| email | tinytext |
| nom_site | tinytext |
| url_site | text |
| login | tinytext |
| pass | tinyblob |
| statut | tinytext |
| maj | timestamp(14) |
| pgp | blob |
| htpass | tinyblob |

- Chaque auteur est identifié par son **id_auteur**.
- **nom**, **bio**, **nom_site**, **url_site**, **pgp** sont respectivement le nom de l'auteur, sa courte biographie, son adresse e-mail, le nom et l'URL de son site Web, sa clé PGP. Informations modifiables librement par l'auteur.
- **email**, **login** sont son e-mail d'inscription et son login. Ils ne sont modifiables que par un administrateur.
- **pass** est le hash MD5 du mot de passe.
- **htpass** est la valeur cryptée (i.e. générée par crypt()) du mot de passe pour le .htpasswd.
- **statut** est le statut de l'auteur : `0minirezo` (administrateur), `1comite` (rédacteur), `5poubelle` (à la poubelle), `6forum` (abonné aux forums, lorque ceux-ci sont réglés en mode « par

abonnement »).
- **maj** a la même signification que dans les autres tables.

**Les brèves : spip_breves**

| | |
|---|---|
| id_breve | bigint(21) |
| date_heure | datetime |
| titre | text |
| texte | longblob |
| lien_titre | text |
| lien_url | text |
| statut | varchar(6) |
| id_rubrique | bigint(21) |
| maj | timestamp(14) |

- Chaque brève est identifiée par son **id_breve**.
- **id_rubrique** est la rubrique (en fait, le secteur) dans laquelle est classée la brève.
- **titre**, **texte**, **lien_titre**, **lien_url** sont le titre, le texte, le nom et l'adresse du lien associé à la brève.
- **date_heure** est la date de la brève.
- **statut** est le statut de la brève : `prop` (proposée à la publication), `publie` (publiée), `refuse` (refusée).
- **maj** : idem que dans les autres tables.

**Les mots-clés : spip_mots**

| | |
|---|---|
| id_mot | bigint(21) |
| type | varchar(100) |
| titre | text |
| descriptif | text |
| texte | longblob |
| maj | timestamp(14) |

- Chaque mot-clé est identifié par son **id_mot**.
- Le **type** du mot-clé est le type, ou groupe, choisi pour le mot-clé. En définissant plusieurs types, on définit plusieurs classifications indépendantes (par exemple « sujet », « époque », « pays »...).
- **titre**, **descriptif**, **texte** parlent d'eux-mêmes.
- **maj** : idem que dans les autres tables.

**Les sites syndiqués : spip_syndic**

| | |
|---|---|
| id_syndic | bigint(20) |
| id_rubrique | bigint(20) |
| id_secteur | bigint(20) |
| nom_site | blob |
| url_site | blob |
| url_syndic | blob |
| descriptif | blob |

- Chaque site syndiqué est identifié par son **id_syndic**.
- **id_rubrique** et **id_secteur** définissent l'endroit dans la hiérarchie du site où viennent s'insérer les contenus syndiqués.
- **nom_site**, **url_site**, **descriptif** sont le nom, l'adresse et le descriptif du site syndiqué.
- **url_syndic** est l'adresse du fichier dynamique utilisé pour récupérer les contenus syndiqués (souvent il s'agit de *url_site* suivi de `backend.php3`).

**Les articles syndiqués : spip_syndic_articles**

| | |
|---|---|
| id_syndic_article | bigint(20) |
| id_syndic | bigint(20) |
| titre | text |
| url | text |
| date | datetime |
| lesauteurs | text |

- Chaque article syndiqué est identifié par son **id_syndic_article**.
- **id_syndic** réfère au site syndiqué d'où est tiré l'article.
- **titre**, **url**, **date**, **lesauteurs** parlent d'eux-mêmes.

## Eléments interactifs

**Les messages de forums : spip_forum**

| | |
|---|---|
| id_forum | bigint(21) |
| id_parent | bigint(21) |
| id_rubrique | bigint(21) |
| id_article | bigint(21) |
| id_breve | bigint(21) |
| date_heure | datetime |
| titre | text |
| texte | mediumtext |
| auteur | text |
| email_auteur | text |
| nom_site | text |
| url_site | text |
| statut | varchar(8) |
| ip | varchar(16) |
| maj | timestamp(14) |
| id_auteur | bigint(20) |
| id_message | bigint(21) |

- Chaque message de forum est identifié par son **id_forum**.
- L'objet auquel est attaché le forum est identifié par son **id_rubrique**, **id_article** ou **id_breve**. Par défaut, ces valeurs sont égales à zéro.
- Le message parent (c'est-à-dire le message auquel répond ce message) est identifié par **id_parent**. Si le message ne répond à aucun autre message, cette valeur est égale à zéro.
- **titre**, **texte**, **nom_site**, **url_site** sont le titre et le texte du message, le nom et l'adresse du lien y attaché.
- **auteur** et **email_auteur** sont le nom et l'e-mail déclarés par l'auteur. Dans le cas des forums par abonnement, ils ne sont pas forcément identiques aux données enregistrées dans la fiche de l'auteur (i.e. dans la table spip_auteurs).
- **id_auteur** identifie l'auteur du message dans le cas de forums par abonnement.
- **statut** est le statut du message : `publie` (lisible dans l'espace public), `prive` (écrit en réaction à un article dans l'espace privé), `privrac` (écrit dans le forum interne dans l'espace privé), `off` (supprimé ou à valider, selon la modération des forums - a priori ou a posteriori).
- **ip** est l'adresse IP de l'auteur, dans les forums publics.
- **maj** a la même signification que dans les autres tables.

**Les pétitions : spip_petitions**

| | |
|---|---|
| id_article | bigint(21) |
| email_unique | char(3) |
| site_obli | char(3) |
| site_unique | char(3) |
| message | char(3) |
| texte | longblob |
| maj | timestamp(14) |

- **id_article** identifie l'article auquel est associée la pétition (une seule pétition par article).
- **email_unique**, **site_obli**, **site_unique**, **message** définissent la configuration de la pétition : l'adresse e-mail des signataires doit-elle être unique dans les signatures, l'adresse Web est-elle obligatoire, est-elle unique, un message attenant aux signatures est-il autorisé (`oui` ou `non`).
- **texte** est le texte de la pétition.
- **maj** : pareil que dans les autres tables.

**Les signatures de pétitions : spip_signatures**

| | |
|---|---|
| id_signature | bigint(21) |
| id_article | bigint(21) |
| date_time | datetime |
| nom_email | text |
| ad_email | text |
| nom_site | text |
| url_site | text |
| message | mediumtext |
| statut | varchar(10) |
| maj | timestamp(14) |

- Chaque signature est identifiée par son **id_signature**.
- **id_article** identifie l'article, donc la pétition sur laquelle est apposée la signature.
- **nom_email**, **ad_email**, **nom_site**, **url_site** sont le nom, l'adresse e-mail, ainsi que le site Web déclarés par le signataire.
- **message** est le message éventuellement entré par le signataire.
- **statut** est le statut de la signature : `publie` (acceptée), `poubelle` (supprimée) ; toute autre valeur donne la valeur de la clé de validation utilisée pour la confirmation par e-mail.
- **maj** a la même signification que dans les autres tables.

## Les relations entre objets

Ces tables ne gèrent aucun contenu, simplement une relation entre les objets présents dans d'autres tables. Ainsi :

- **spip_auteurs_articles** spécifie la relation entre auteurs et articles. Si un *id_auteur* y est associé à un *id_article*, cela veut dire que l'auteur en question a écrit ou co-écrit l'article (il peut y avoir plusieurs auteurs par article, et vice-versa bien sûr).

- **spip_mots_articles** définit de même la relation de référencement des articles par des mots-clés.

## Gestion du site

La table **spip_meta** est primordiale. Elle contient des couples (`nom`, `valeur`) indexés par le nom (clé primaire) ; ces couples permettent de stocker différentes informations telles que la configuration du site, ou la version installée de SPIP.

La table **spip_forum_cache** est utilisée afin d'adapter le système de cache à l'instantanéité des forums. Pour chaque fichier du cache ayant donné lieu à une requête sur la table spip_forum, la table spip_forum_cache stocke les paramètres de la requête (article, rubrique, brève et éventuel message parent du forum). Lorsqu'un message est posté, les paramètres du message sont comparés avec ceux présents dans spip_forum_cache, et pour chaque correspondance trouvée le fichier cache indiqué dans la table est effacé. Ainsi les messages

n’attendent pas le recalcul régulier de la page dans laquelle ils s’insèrent pour apparaître dans l’espace public.

## Indexation (moteur de recherche)

Six tables sont utilisées par le moteur de recherche. Elles se divisent en deux catégories.

**Le dictionnaire d’indexation : spip_index_dico**

Chaque mot rencontré au cours de l’indexation est stocké dans cette table, ainsi que les 64 premiers bits de son hash MD5. C’est le mot qui sert de clé primaire, permettant ainsi d’effectuer très rapidement des requêtes sur un début de mot ; on récupère alors le(s) hash satisfaisant à la requête, afin d’effectuer la recherche proprement dite dans les tables d’indexation.

**Les tables d’indexation : spip_index_***

Ces tables, au nombre de cinq, gèrent chacune l’indexation d’un type d’objet : articles, rubriques, brèves, auteurs, mots-clés. Une entrée par mot et par objet est stockée. Chaque entrée contient le hash du mot (en fait les 64 bits de poids fort du hash MD5, cf. ci-dessus), l’identifiant de l’objet indexé (par exemple l’*id_article* pour un article), et le nombre de points associé à l’indexation du mot dans l’objet. Ce nombre de points est calculé en fonction du nombre d’occurences du mot, pondéré par le champ où ont lieu les occurences : une occurence dans le titre d’un article génère plus de points qu’une occurence dans le corps de l’article.

# 8. Autres fonctions avancées

**Vous trouverez ici une description détaillée des autres fonctions avancées à la disposition du webmestre.**

## Mutualisation : un SPIP pour plusieurs sites

9 juillet — maj : Décembre 2008

Partager les fichiers de SPIP entre plusieurs sites, ce que l'on appelle une mutualisation, permet un gain d'espace disque important, ainsi qu'une possibilité de mise à jour de SPIP simple de l'ensemble des sites en ne mettant à jour que le noyau.

### Le concept...

Les dossiers nécessaires au fonctionnement du noyau SPIP (ecrire, prive, dist (devenu squelettes-dist en 2.0)), et ceux marquant l'activité d'un site (config, IMG, tmp, local) sont clairement identifiables et séparés. C'est cette séparation qui permet d'avoir plusieurs sites SPIP autonomes pour un même noyau de SPIP.

Cette autonomie repose sur quatre répertoires par site dans lesquels SPIP va écrire les données résultant de l'activité du site. On distingue les données temporaires et permanentes d'une part, et les données accessibles par HTTP et celles qui ne le sont pas d'autre part. Les deux répertoires inaccessibles par HTTP seront protégés par un .htaccess installé automatiquement par SPIP (les administrateurs du serveur peuvent mettre ces répertoires en dehors de l'arborescence servie par HTTP).

Ces quatre répertoires sont distincts et nommés par les constantes PHP suivantes :

```
define('_NOM_TEMPORAIRES_INACCESSIBLES', "tmp/");
define('_NOM_TEMPORAIRES_ACCESSIBLES', "local/");
define('_NOM_PERMANENTS_INACCESSIBLES', "config/");
define('_NOM_PERMANENTS_ACCESSIBLES', "IMG/");
```

Dans une installation simple de SPIP, ces répertoires sont créés à la racine du site avec les valeurs par défaut ci-dessus. La mutualisation des sources de SPIP repose sur l'association de quatre répertoires spécifiques à chaque site, déduits de leurs URL.

### Initialisation de la mutualisation

Cette association est effectuée par une fonction, `spip_initialisation`. Elle admet quatre arguments indiquant la localisation des quatre répertoires fondamentaux, et construit à partir de ceux-ci les constantes servant au fonctionnement de SPIP. Dans une installation simple, ces quatre arguments sont les quatre constantes ci-dessus. La fonction `spip_initialisation` est appelée juste après le chargement de `mes_options.php`, et refuse silencieusement de s'exécuter si elle a déjà été appelée. En conséquence, si dans ce fichier cette fonction est appliquée sur des arguments différents déduits de l'URL du site, on aura autant d'utilisation des sources de SPIP que d'URL de site. On peut aussi définir dans ce fichier les constantes servant au fonctionnement de SPIP, ces définitions ayant priorité sur celles qu'essaiera de définir `spip_initialisation`.

Le code ci-dessous, placé dans le fichier `config/mes_options.php` prend le nom de domaine et exécute une mutualisation sur les quatre répertoires *tmp, local, config* et *IMG* placés dans le dossier `sites/nom_du_domaine/`. Auparavant, la définition de quelques constantes permet de centraliser les fichiers de journalisation de tous les sites dans un unique répertoire, `log`, et non dans `sites/nom_du_domaine/tmp/` pour chacun. Cette centralisation n'a rien d'obligatoire mais se révèle utile ; elle peut s'appliquer aussi aux répertoires d'aide et de sauvegarde.

```
<?php
$rep = 'sites/';
$site = $_SERVER['HTTP_HOST'];
$path = _DIR_RACINE . $rep . $site . '/';

// ordre de recherche des chemins
define('_SPIP_PATH',
      $path . ':' .
      _DIR_RACINE .':' .
      _DIR_RACINE .'squelettes-dist/:' .
      _DIR_RACINE .'prive/:' .
      _DIR_RESTREINT);

// ajout du dossier squelette
if (is_dir($path . 'squelettes'))
      $GLOBALS['dossier_squelettes'] = $rep . $site . '/squelettes';

// exemple de logs a la racine pour tous les sites
define('_FILE_LOG_SUFFIX', '_' . $site . '.log');
define('_DIR_LOG',  _DIR_RACINE . 'log/');

// prefixes des cookie et des tables :
$cookie_prefix = str_replace('.', '_', $site);
$table_prefix = 'spip';

// exectution du fichier config/mes_option.php du site mutualise
if (is_readable($f = $path . _NOM_PERMANENTS_INACCESSIBLES . _NOM_CONFIG
 . '.php'))
      include($f);

// demarrage du site
spip_initialisation(
      ($path . _NOM_PERMANENTS_INACCESSIBLES),
      ($path . _NOM_PERMANENTS_ACCESSIBLES),
      ($path . _NOM_TEMPORAIRES_INACCESSIBLES),
      ($path . _NOM_TEMPORAIRES_ACCESSIBLES)
);
?>
```

## Mutualiser des domaines ou des sous-domaines

Pour mutualiser plusieurs domaines ou sous-domaines, vous devez tous les diriger vers le répertoire physique contenant SPIP. Par exemple, `http://example.org/`, `http://example.net/` et `http://sous_domaine.example.net/`, pour les mutualiser, doivent tous pointer sur le même répertoire, comme `/var/www/spip/`.

Ce dossier contient donc SPIP ainsi qu'un fichier `config/mes_options.php` pour activer la mutualisation. En reprenant l'exemple de code donné au dessus, il faudra alors créer les dossiers *sites* et *log* ainsi que *sites/example.org*, *sites/example.net* et *sites/sous_domaine.example.net* avec dans chacun d'eux les répertoires *config, local, IMG* et

*tmp* accessibles en écriture. Il n'y a rien d'autre à réaliser. En allant sur l'un des sites, SPIP devrait vous proposer son installation.

Si vous souhaitez pouvoir activer des types d'url différents, comme les « url arborescentes » ou les « url propres », il sera nécessaire de créer un fichier d'accès Http, qui est fourni dans la distribution sous le nom de `htaccess.txt` : il suffit de le renommer en `.htaccess` pour le mettre en œuvre. Il s'appliquera à tous les sites mutualisés, mais il est possible d'avoir des accès différenciés en configurant astucieusement le serveur (voir plus bas). Dans les deux cas, tout cela ne fonctionnera que si votre serveur accepte d'exécuter toutes les directives présentes dans ce fichier, notamment celles concernant la réécriture d'url (« url rewriting »).

## Mutualiser les répertoires d'un domaine

Chaque dossier (virtuel) d'un domaine, en analysant correctement l'URL, peut aussi devenir un site SPIP mutualisé. Il faut pour cela que l'URL du dossier soit redirigée de façon transparente vers la racine du SPIP. C'est le rôle du fichier `.htaccess` modifié. Ainsi `http://example.com/premier_site/` et `http://example.com/second_site/` peuvent être chacun des sites SPIP mutualisés (et `http://example.com/` aussi).

Dans un premier temps, il faut renommer le fichier *htaccess.txt* en *.htaccess* et le modifier pour indiquer que les répertoires qui sont virtuels doivent pointer à la racine de SPIP. Pour cela, ajouter dans la partie « Réglages personnalisés » soit le code générique qui traite tous les répertoires virtuels (mais *http://example.com/premier_site* sans / final renverra une erreur), soit un code indiquant les noms des dossiers (pas d'erreur si aucun / final) :

```
// code générique
RewriteCond %{REQUEST_URI}
!^/(config|ecrire|IMG|prive|plugins|sites|squelettes-
dist|squelettes|tmp|lib|local|mutualisation)/(.*)
RewriteRule
 ^[^/]+/(.*) /$1 [QSA,L]

// OU code spécifique
RewriteRule ^(premier_site|second_site)$ /$1/ [R,L]
RewriteRule ^(premier_site|second_site)/(.*) /$2 [QSA,L]
```

Dans un second temps, il faut créer une mutualisation en analysant dans le fichier `config/mes_options.php` l'url transmise. Voici un exemple pour avoir les sites mutualisés dans *sites/exemple.com*, *sites/premier_site* et *sites/second_site* :

```
<?php
if (
	(
		preg_match(',^/([\.a-zA-Z0-9_-]+)/,', $_SERVER['REQUEST_URI'],
$r)
		AND !is_dir(_DIR_RACINE . $r[1])
	)
) {
	$site = $r[1];
} else {
	$site = $_SERVER['HTTP_HOST'];
}

$rep = 'sites/';
$path = _DIR_RACINE . $rep . $site . '/';

// ordre de recherche des chemins
define('_SPIP_PATH',
	$path . ':' .
	_DIR_RACINE .':' .
	_DIR_RACINE .'squelettes-dist/:' .


	_DIR_RACINE .'prive/:' .
	_DIR_RESTREINT);

// ajout du dossier squelette
if (is_dir($path . 'squelettes'))
	$GLOBALS['dossier_squelettes'] = $rep . $site . '/squelettes';

// prefixes des cookie et des tables :
$cookie_prefix = str_replace('.', '_', $site);
$table_prefix = 'spip';

// exectution du fichier config/mes_option.php du site mutualise
if (is_readable($f = $path . _NOM_PERMANENTS_INACCESSIBLES . _NOM_CONFIG
 . '.php'))
	include($f);

// demarrage du site
spip_initialisation(
	($path . _NOM_PERMANENTS_INACCESSIBLES),
	($path . _NOM_PERMANENTS_ACCESSIBLES),
	($path . _NOM_TEMPORAIRES_INACCESSIBLES),
	($path . _NOM_TEMPORAIRES_ACCESSIBLES)
);
?>
```

## Préfixe des tables

Les exemples présentés nécessitent autant de bases de données que de sites mutualisés. Cependant, il est possible d'installer tous les sites dans une même base de données, en préfixant les noms de leurs tables d'un préfixe unique.

L'exemple ci-dessous crée un préfixe de 8 caractères : un préfixe avec les 4 premières lettres du site, suivi de 4 autres caractères.

```
$table_prefix = substr(str_replace('.', '_', $site),0,4) .
substr(md5($site),0,4);
```

## Fichier config/mes_options.php

Toute modification du fichier *config/mes_options.php* du noyau SPIP affectera les options de tous les sites hébergés.

Par exemple, mettre dans ce fichier : `$type_urls = 'propres' ;`

donnera par défaut à tous les sites ce type d'URL... Mais chaque site peut le changer dans son propre */sites/repertoire_du_site/config/mes_options.php*.

## Configurer Apache pour les domaines et sous-domaines

La mutualisation côté serveur, pour ce qui concerne la gestion des sous-domaines ou des domaines reste simple. Il suffit de créer une redirection entre le nom de domaine et le répertoire physique où est stocké SPIP.

Voici un exemple de configuration (minimale) pour un serveur nommé 'exemple.tld' utilisant des sous-domaines. Le fichier de configuration est à créer dans */etc/apache2/sites_availables/exemple.tld*, qu'il faut ensuite activer

```
sudo a2ensite exemple.tld
sudo /etc/init.d/apache2 reload
```

SPIP est installé dans cet exemple dans '/var/www/spip/'.

```
<VirtualHost *>
      ServerName exemple.tld
      ServerAdmin webmaster@localhost
      ServerAlias *.exemple.tld

      DocumentRoot /var/www/spip/
      <Directory /var/www/spip/>
            Options Indexes FollowSymLinks MultiViews
            AllowOverride All
            Order allow,deny
            allow from all
      </Directory>
</VirtualHost>
```

Si vous voulez en plus avoir des fichiers d'accès différenciés, il suffit d'utiliser la directive *AccessFileName* à l'intérieur de la directive *VirtualHost*, de sorte que le fichier d'accès du site *S* se nomme par exemple `.htaccess-S`.

## Note sur les sauvegardes et les restaurations

Chaque site copie par défaut ses fichiers de sauvegardes dans le répertoire */sites/premier_site/tmp/dump* (ou */sites/premier_site/tmp/upload/login* pour les sauvegardes d’un administrateur restreint).

Les restaurations se font par le même dossier. Le chemin des images est aussi correctement pris en compte.

## Étendre l’aide en ligne

Avril 2010 — maj : juillet 2010

Le script d’aide en ligne (à partir de la version SPIP 2.1) est désormais capable de fusionner plusieurs sources d’informations, et de les présenter de manière unifiée dans la fenêtre à deux frames qui s’ouvre lorsqu’on clique sur les icones de point d’interrogation depuis les pages de l’espace privé.

### Principes

La globale **`help_server`** est à présent non plus l’unique URL d’un répertoire d’aide, mais un *tableau* de telles URL (la compatibilité est assurée néanmoins).
Lorsque de l’aide est demandée dans une langue « *L* », SPIP va récupérer toutes les pages nommées « *L-aide.html* » existantes aux URL indiquées par cette globale, et fusionner les informations. Par défaut, ce tableau global a pour seul élément `http://www.spip.net/aide`, ce qui permet de retomber sur le comportement habituel mais peut donc être désormais complété par d’autres éléments.

On déclarera donc dans le fichier `mes_options.php` [1] l’URL pointant vers ce ou ces nouveaux fichiers d’aide.

### Exemple

L’objectif est d’ajouter un paragraphe personnel d’aide au chapitre « *Les raccourcis typographiques* » de l’aide en ligne.

- Dans le répertoire `squelettes/`, créer le fichier `fr-aide.html` [2] ainsi rédigé :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr"
dir="ltr">
<head>
<link rel='stylesheet' href='http://www.spip.net/prive/spip_admin.css'
type='text/css' />
</head>
<body>
<h2>raccourcis/Les raccourcis typographiques</h2>
<!-- ajout d'aide personnalisée -->
<h3 class="spip">Mes raccourcis personnels</h3>
<p style="color:red;">
Ici un paragraphe d'aide personnalisée sur les raccourcis typographiques.
</p>
<p style="color:navy;">
Ici un autre paragraphe d'aide personnalisée.
</p>
<!-- fin ajout d'aide personnalisée -->
</body>
</html>
```

- Dans le fichier `mes_options.php`, déclarer l’URL pointant vers le répertoire contenant ce fichier :

```
// URL supplémentaire où trouver de l'aide en ligne
$GLOBALS['help_server'][] = url_de_base(1) . 'squelettes/';
```

- Désormais, dans la fenêtre habituelle d'aide, un clic sur « *Les raccourcis typographiques* », dans le menu de gauche, affiche le texte de l'aide par défaut de spip.net *complété* (à sa fin) par le texte personnalisé [3].

## Points d'entrée par défaut du menu de l'aide

Pour pouvoir *greffer* un supplément d'aide il faut baliser sa page d'aide personnelle en utilisant l'une des sections `<h2>clef/titre_menu</h2>` disponibles sur spip.net.

En voici la liste :

**Installation de SPIP**

- install0/Régler les droits d'accès
- install1/Votre connexion SQL
- install2/Choix de votre base
- install5/Informations personnelles
- ftp_auth/Vérification par FTP
- erreur_mysql/Un problème de squelette ?

**Les articles**

- raccourcis/Les raccourcis typographiques
- arttitre/Titre, surtitre, soustitre
- artrub/Choisir la rubrique
- artdesc/Descriptif rapide
- artchap/Chapeau
- artvirt/Redirection d'article
- arttexte/Texte
- artdate/Date
- artdate_redac/Date de rédaction antérieure
- artauteurs/Les auteurs
- logoart/Logo de l'article
- artstatut/Le statut de l'article
- artprop/Proposer son article

**Les rubriques**

- rubhier/Une structure hiérarchisée
- rubrub/Choisir la rubrique
- rublogo/Logo de la rubrique

**Les brèves**

- breves/Les brèves
- brevesrub/Choisir la rubrique
- breveslien/Le lien hypertexte
- brevesstatut/Le statut de la brève
- breveslogo/Le logo de la brève

**Images et documents**

- ins_img/Insérer des images
- ins_doc/Joindre des documents
- ins_upload/Installer des fichiers par FTP ou SSH

**Les mots-clés**

- mots/Principe des mots-clés
- artmots/Les mots-clés
- motsgroupes/Les groupes de mots

**Les sites référencés**

- reference/Référencer un site
- rubsyn/Sites syndiqués
- artsyn/Articles syndiqués
- confhttpproxy/Utiliser un proxy

**La messagerie interne**

- messut/ Les messages entre utilisateurs
- messpense/ Les pense-bête
- messcalen/Le calendrier
- messconf/Configuration personnelle de la messagerie

**Suivi des forums**

- suiviforum/Suivi des forums

**Configuration du site**

- confnom/Nom et adresse de votre site
- confart/Contenu des articles
- confdates/Articles post-datés
- confforums/Fonctionnement des forums
- confbreves/Système de brèves
- confmessagerie/Messagerie interne
- confstat/Statistiques des visites
- confmails/Envoi automatique de mails

**Configuration de l'interface personnelle**

- cookie/Le cookie de correspondance
- deconnect/Se déconnecter

**Usages avancés**

- artmodif/Articles en cours de « modification »
- suivimodif/Suivi des révisions
- previsu/La prévisualisation
- latex/Formules mathématiques
- confurl/Type d'adresses URL

**SPIP, un logiciel libre**

- licence/Licence et conditions d'utilisation

## Ajouter un bloc supplémentaire dans le menu de gauche

On peut aussi vouloir ajouter un *bloc* supplémentaire dans le menu de gauche (bloc dont le titre est affiché sur un fond coloré vert/beige).
Pour cela il suffit d'ajouter un élément `<h1> </h1>` dans notre fichier ..-aide.html :

```
<body>
<h1>Aide particulière personnalisée</h1>
<h2>bloc_perso/De l'aide sur tel élément</h2>
<h3 class="spip">Utilisation de tel élément</h3>
<p>
Ici un paragraphe d'aide personnalisée.
</p>
<p>
Ici un autre paragraphe d'aide personnalisée.
</p>
<h2>autre_bloc_perso/De l'aide sur tel autre élément</h2>
<h3 class="spip">Utilisation de tel autre élément</h3>
<p>
Encore un paragraphe d'aide personnalisée.
</p>
</body>
```

## Ranger et organiser ses fichiers d'aide

Si vous préférez organiser vos fichiers d'aide supplémentaires dans un sous-répertoire particullier (« *mes_aides/* » par exemple), vous devrez déclarer dans votre fichier « *mes_options.php* » :

```
$GLOBALS['help_server'][] = url_de_base(1) . 'squelettes/mes_aides/';
```

Voir aussi la balise **`#AIDER`**

## Notes

[1] Dans le cadre d'un plugin (« *monplugin* » par exemple, on déclarera cette URL dans le fichier `monplugin_options.php`. Par exemple :
`$GLOBALS['help_server'][] = url_de_base(1) . str_replace("../", "", _DIR_PLUGIN_MONPLUGIN);`
ou encore, si vous préférez ranger vos fichiers d'aide dans un sous-répertoire particulier :
`$GLOBALS['help_server'][] = url_de_base(1) . str_replace("../", "", _DIR_PLUGIN_MONPLUGIN) . "mes_aides/";`

[2] Dans le cadre d'une utilisation multilingue, multiplier ces fichiers : ar-aide.html, en-aide.html, es-aide.html, ...

[3] Ne pas oublier que pour voir effectivement les modifications apportées, il conviendra de vider le cache (pour supprimer le dossier `tmp/cache/aide/`)

## Le fichier mes_options.php

Mars 2010 — maj : avril 2010

Le fichier **`mes_options.php`** offre la possibilité de modifier la configuration par défaut de SPIP [1].

Ce fichier est à créer (s'il n'existe pas déjà) dans le répertoire `config/`.

Lorsqu'il existe, ce fichier est inclu automatiquement à chaque affichage de page (partie privée comme partie publique) ; on veillera donc à ne pas le remplir inutilement.

**`mes_options.php`** étant un fichier PHP, il doit commencer par **`<?php`** et se terminer par **`?>`** (soyez extrêmement vigilant : il ne doit y avoir aucun caractère ni espace ni ligne vierge *avant* la balise ouvrante `<?php` ni *après* la balise fermante `?>`).

- Exemple de fichier `mes_options.php` :

```
<?php

// interdire l'upload de documents de plus de 200 Ko
define('_DOC_MAX_SIZE', 200);

// interdire toute exécution des scripts javascript
// embarqués dans les rédactionnels
$filtrer_javascript = -1;

// forcer l'affichage d'un menu déroulant à partir de 1998 pour
// le champ "Date de première publication"
$debut_date_publication = '1998';

// ne jamais afficher les préfixes numériques des titres
$table_des_traitements['TITRE'][]= 'typo(supprimer_numero(%s))';


?>
```

### Notes

[1] voir la rubrique personnalisation.

## Les aides au débuggage de squelettes

Novembre 2009 — maj : août 2010

SPIP propose nativement quelques fonctionnalités pour venir en aide au webmestre lors de la phase de débuggage des squelettes.
Ces fonctions d'information sont accessibles en passant des variables spécifiques dans l'url de la page appelée.

Les « **var_mode** » et « **var_profile** » s'utilisent en ajoutant soit **`?var_mode=...`** *(ex : -`titre-de-rubrique-?var_mode=...`)*, soit **`&var_mode=...`** *(ex : `spip.php?article3&var_mode=...`)* à l'url de la page appelée.
Leur utilisation n'est fonctionnelle que pour les *administrateurs* loggés.

### var_mode=calcul et var_mode=recalcul

L'appel de « **var_mode=calcul** » regénère le code html (lance l'exécution du code déjà compilé) et rafraichit le cache (crée les fichiers html qui n'auront plus qu'à être lus lors des prochains appels de la page).

L'appel de « **var_mode=recalcul** » regénère le code php (effectue une nouvelle compilation du squelette), puis regénère le code html (lance l'exécution du code qui vient d'être compilé) enfin rafraichit le cache (crée les fichiers html qui n'auront plus qu'à être lus lors des prochains appels de la page).
le **recalcul** de la page regénère aussi les css et scripts javascript compressés.
le **recalcul** ne s'applique pas aux images (vignettes, images-typo, ...)

Ces deux appels peuvent être lancés par un clic sur l'un des boutons d'administration ;


un premier clic appelant « *var_mode=calcul* » (lorsque l'intitulé du bouton est accompagné d'une étoile, c'est que la page affichée est lue depuis le cache) ;


un second clic appelant « *var_mode=recalcul* ».

*Attention* : On pourrait être tenté d'utiliser `&var_mode=recalcul` dans des liens de ses squelettes (pour forcer la mise à jour de css ou javascript par exemple). C'est une *trés mauvaise pratique à éviter* car consommatrice de ressources serveur (re-compilation du squelette).

## var_mode=inclure

L'appel de « **var_mode=inclure** » affiche le nom et le chemin de chaque noisette (inclure ou modèle) qui compose la page.

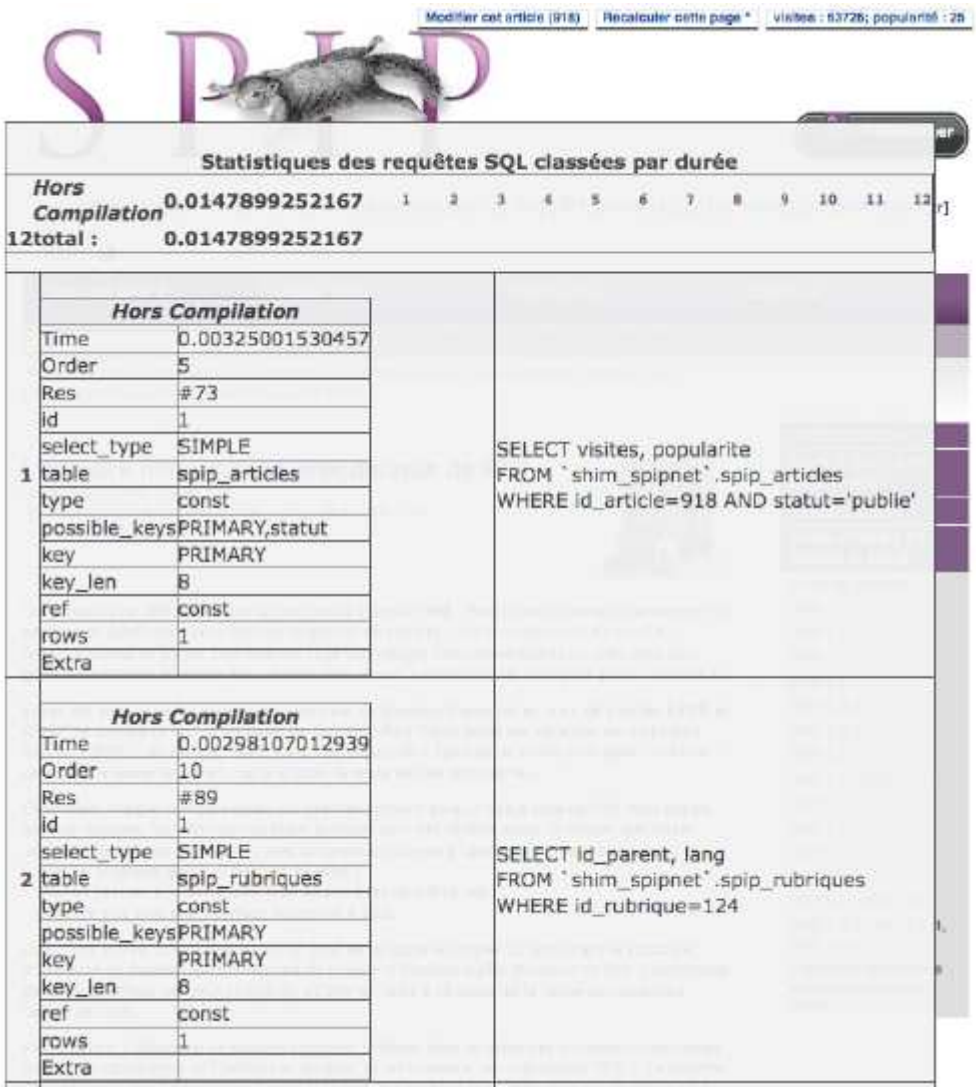
Ceci permet de vérifier que les inclure réellement appelés par le squelette de la page sont bien ceux que l'on y a spécifiés.


## var_profile=1

L'appel de « **var_profile=1** » affiche le détail des requêtes sql et les temps de calcul de chacune. Les requêtes sont classées de la plus gourmande en temps d'exécution à la plus rapide.

Cette fonction est particulièrement utile pour comprendre ce qui peut rendre une page excessivement lente à s'afficher.
Elle permet de visualiser les requêtes sql générées par chaque boucle du squelette (y compris

les squelettes inclus) ainsi que les requêtes *hors boucle* (notées « *Hors Compilation* ») générées par SPIP.



Statistiques des requêtes SQL classées par durée

| | | |
|---|---|---|
| Hors Compilation | 0.0147899252167 | 1 2 3 4 5 6 7 8 9 10 11 12 |
| 12total : | 0.0147899252167 | |

| | Hors Compilation | |
|---|---|---|
| | Time | 0.00325001530457 |
| | Order | 5 |
| | Res | #73 |
| | id | 1 |
| | select_type | SIMPLE |
| 1 | table | spip_articles |
| | type | const |
| | possible_keys | PRIMARY,statut |
| | key | PRIMARY |
| | key_len | 8 |
| | ref | const |
| | rows | 1 |
| | Extra | |

```
SELECT visites, popularite
FROM `shim_spipnet`.spip_articles
WHERE id_article=918 AND statut='publie'
```

| | Hors Compilation | |
|---|---|---|
| | Time | 0.00298107012939 |
| | Order | 10 |
| | Res | #89 |
| | id | 1 |
| | select_type | SIMPLE |
| 2 | table | spip_rubriques |
| | type | const |
| | possible_keys | PRIMARY |
| | key | PRIMARY |
| | key_len | 8 |
| | ref | const |
| | rows | 1 |
| | Extra | |

```
SELECT id_parent, lang
FROM `shim_spipnet`.spip_rubriques
WHERE id_rubrique=124
```

On peut y relever :

- pour chaque boucle
  - le nombre de fois où la boucle (donc la requête sql) a été exécutée,
  - le temps (en *secondes*) pris par la boucle (c'est à dire le temps de la requête correspondante multiplié par le nombre d'exécutions de la boucle),
  - la liste (triée par ordre chronologique de toutes les requêtes générées par la page) de chaque exécution de la requête (qui sert de lien vers le détail de chaque requête).

- les requêtes *hors boucle*

- le temps total pris par l'ensemble des requêtes pour la page

- le détail de chaque requête effectuée avec
  - un tableau statistique affichant l'*explication* de la requête,
  - l'intitulé de la requête.

*Attention* : Penser à **calculer** la page (`&var_profile=1&var_mode=calcul`) pour éviter que ce ne soit le cache qui soit lu, faussant ainsi le résultat affiché.

## var_mode=preview

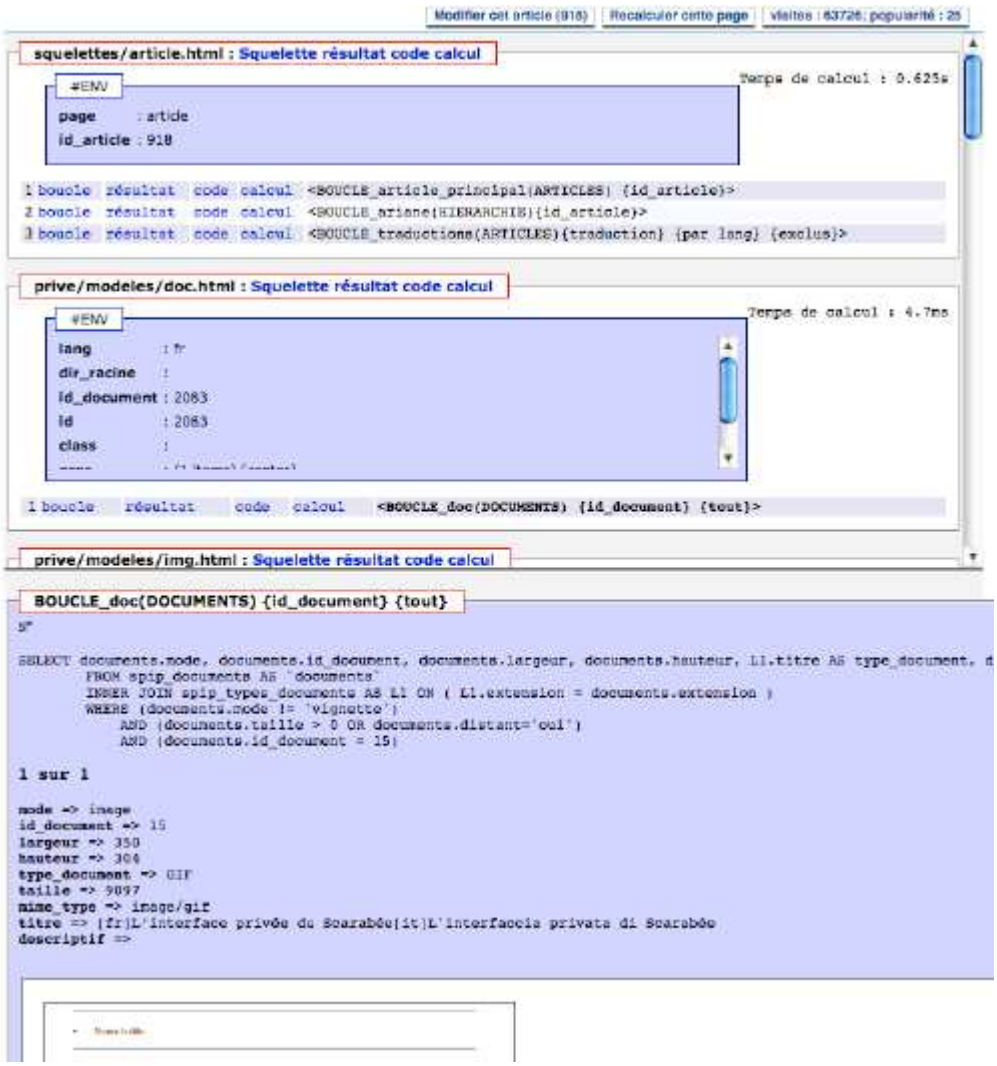
L'appel de « **var_mode=preview** » depuis l'espace privé permet de visualiser dans l'espace public un article de statut « *proposé à la publication* » sans avoir besoin de le publier.


## var_mode=urls

L'appel de « **var_mode=urls** » force la mise à jour de toutes les urls de la page.

## var_mode=debug

L'appel de « **var_mode=debug** » détaille la génération d'une page (boucles spip - requêtes sql - code php - statistiques des requêtes sql).




**var_mode=debug**

Cette page affiche pour le squelette principal ainsi que pour chacun des squelettes inclus

- la liste des valeurs des variables de contexte (#ENV) passées par le squelette *appelant*,
- la liste des boucles.

Les différents liens permettent d'accèder à :

- **Squelette**
  le code texte du squelette (évite ainsi d'avoir accès au fichier .html pour en lire le contenu),
  - **résultat**
    le code généré par ce squelette spécifiquement (hors inclusions éventuelles) qui, une fois *évalué*, retournera du html au navigateur,
  - **code**
    le code php généré par le compilateur (analyse plutôt destinée aux développeurs avertis) qui, une fois *exécuté*, produira le résultat (voir ci-dessus),

    - **calcul**
      le détail des requêtes sql et, pour chacune, les temps de calcul (voir ci-dessus `var_profile`).

- **boucle**
  le code texte de la boucle complète (de `<B_abc>` à `<//B_abc>`),
    - **résultat**
      la requête sql générée par la boucle étudiée,
      une liste des premiers résultats retournés par cette requête[7],
    - **code**
      le code php (généré par le compilateur) de la fonction spécifique associée à cette boucle (analyse plutôt destinée aux développeurs avertis),
    - **calcul**
      le détail et les temps de calcul de la requête sql associée à cette boucle (voir ci-dessus `var_profile`).

## Afficher le contenu d'une table

Un appel de l'url `?page=table:`*`nom_de_la_table`* depuis l'espace public affichera sous forme de tableau le **contenu** de la table nommée.
*par exemple : `?page=table:articles` pour la table `spip_articles`.*

La première ligne de ce tableau liste les champs de la table dans l'ordre alphabétique (attention : ne pas confondre « *N°* » et « *id_...* ») ; un clic sur l'un de ces champs ordonnant les résultats (ascendant/descendant).

La deuxième ligne permet d'effectuer une recherche sur une valeur spécifique d'un champ.

L'ensemble des données est affiché en mode paginé, par groupe de 10 résultats.

*À noter* : Cette fonction permettant d'afficher *l'intégralité* des données de *toutes* les tables de la base de données n'est utilisable **que par un webmestre loggé**.

## Obtenir encore plus d'informations pour le debuggage

- Désactiver le cache de SPIP en ajoutant dans *config/mes_options.php* :

```
define('_NO_CACHE', 1);
```

- Activer les rapports d'erreurs PHP en ajoutant dans *config/mes_options.php* :

```
error_reporting(E_ALL^E_NOTICE);
ini_set ("display_errors", "On");
define('SPIP_ERREUR_REPORT', E_ALL^E_NOTICE);
define('SPIP_ERREUR_REPORT_INCLUDE_PLUGINS', E_ALL^E_NOTICE);
```

[7] il est possible de modifier le nombre de résultats affichés ici en plaçant dans le fichier `mes_options.php` du répertoire `config/` la ligne : `define('_MAX_DEBUG_AFF', 'n');`
(par défaut, pour éviter l'affichage de centaines de retours sur des boucles trop génériques, la valeur « *n* » de cette constante est fixée à 50).

## P.-S.

Voir aussi les vidéos de l'atelier consacré à ce sujet lors des **journées 2009 en Avignon** :

- Bug et debug : niveau 1 (http://videos.spip.org/spip.php?article128)
- Bugs et debug php : niveau 2 (http://videos.spip.org/spip.php?article115).

## #BALISE* et #BALISE**

Novembre 2009 — maj : Octobre 2009

Par défaut, SPIP applique automatiquement sur chaque #BALISE rencontrée dans les squelettes tous les traitements (filtres ou fonctions de transformation) qui lui sont spécifiques, traitements définis dans `#$GLOBALS['table_des_traitements']['BALISE']`.

Dans certains cas, par exemple pour *exécuter un filtre personnel **avant** l'application automatique des fonctions internes de SPIP*, il peut être nécessaire de désactiver ce traitement.

### L'étoile simple : *

On utilisera dans ce cas la notation **`#BALISE*`** à laquelle on appliquera son filtre personnel en prenant bien soin de vérifier s'il est nécessaire d'y ajouter **après** les traitements internes que l'on vient de désactiver.

Par exemple :

```
[(#TEXTE*|mon_filtre|propre)]
```

commencera par *désactiver* les traitements automatiques appliqués à #TEXTE et définis dans `#$GLOBALS['table_des_traitements']['TEXTE']` ;
puis *appliquera* le filtre personnel `|mon_filtre` ;
enfin *appliquera* le filtre `|propre` qui venait d'être désactivé par l'utilisation de `*`.

Illustration :
Le texte de mon article, tel que je l'ai saisi dans l'interface privée et tel qu'il est archivé en base de données est[8] :

```
<img388|left>

Il y a aujourd'hui {trois cent quarante-huit ans six mois et dix-neuf
jours} que les parisiens s'éveillèrent au bruit de toutes les cloches
sonnant à grande volée dans la triple enceinte de la Cité, de l'Université
et de la Ville.
_ Ce n'est cependant pas un jour dont l'histoire ait gardé souvenir que le
{{6 janvier 1482}}.
```

**`#TEXTE`** affiche le source html suivant (avec les traitements par défaut donc) :

```
<p><span class="spip_document_388 spip_documents spip_documents_left"
style="float: left; width: 150px;">
<img src="../local/cache-vignettes/L150xH150/extraits-4f9ff.jpg" alt=""
style="height: 150px; width: 150px;" height="150" width="150"></span></p>
<p>Il y a aujourd'hui <i>trois cent quarante-huit ans six mois et dix-neuf
jours</i> que les parisiens s'éveillèrent au bruit de toutes les cloches
sonnant à grande volée dans la triple enceinte de la Cité, de l'Université
et de la Ville.
<br>Ce n'est cependant pas un jour dont l'histoire ait gardé souvenir que
le <strong>6 janvier 1482</strong>.</p>
```

**`#TEXTE*`** affichera le source html suivant :

[8] merci à Victor Hugo pour cet extrait de *Notre Dame de Paris*.

```
<img388|left>

Il y a aujourd'hui {trois cent quarante-huit ans six mois et dix-neuf
jours} que les parisiens s'éveillèrent au bruit de toutes les cloches
sonnant à grande volée dans la triple enceinte de la Cité, de l'Université
et de la Ville.
_ Ce n'est cependant pas un jour dont l'histoire ait gardé souvenir que le
{{6 janvier 1482}}.
```

Sur ce source *brut*, j'applique mon filtre perso (disons qu'il modifie l'image associée à cet article et ce *pour ce squelette spécifiquement*, ce qui m'interdit de le faire en ré-éditant l'article...). Ce filtre, donc, recherche la séquence `<imgxxx|` pour la remplacer par `<img25|` ; pour simplifier : il remplace les images de l'article par une image unique. Noter que ceci n'aurait pas été possible (du moins, pas *simplement*) à partir du source html retourné par l'utilisation de `#TEXTE`.

Maintenant que mon filtre a fait son travail, il me faut appeler le traitement automatique que SPIP applique à la balise `#TEXTE` pour retrouver le traitement du (nouveau) modèle `<img25|left>` et les traitements typographiques divers : gras, italique...

J'appelle donc dans mon squelette `:[(#TEXTE*|mon_filtre|propre)]` qui me retournera bien le source html :

```
<p><span class='spip_document_25 spip_documents spip_documents_left'
 style='float:left; width:180px;'>
<img src='IMG/png/defaut_cache.png' width="180" height="180" alt=""
/></span></p>
<p>Il y a aujourd'hui <i>trois cent quarante-huit...
```

## L'étoile double : **

En plus de ce que l'on vient de voir, SPIP applique par défaut à toutes les balises rencontrées dans un squelette un **traitement de sécurité** qui interdit l'exécution de script (php ou javascript) susceptible d'être retourné par la balise. Par exemple un `<?php echo 'toto'; ?>`, inclus dans le corps du texte d'un article, ne sera pas exécuté (« toto » ne s'affichera pas).

Dans certains cas, **trés spécifiques**, il peut, cependant, être nécessaire de récupérer la valeur brute, non *désinfectée* (donc potentiellement dangereuse), d'une balise. Par exemple pour traiter certains retours de formulaires, certaines variables d'environnement passées dans `#ENV`.

Dans ces cas là, et en étant conscient des risques posés, on utilisera la notation **`#BALISE**`**. Mais une fois encore, attention à cette écriture (voir à ce propos l'utilisation de la balise #ENV).

## Le système de pagination

Juillet 2006 — maj : mai 2010

Lorsqu'une boucle renvoie plusieurs dizaines d'articles (ou, pour une pétition, plusieurs milliers de signatures), il n'est pas souhaitable, voire impossible, de tout afficher sur une seule page.

On préférera alors répartir les résultats en plusieurs pages, avec un système de navigation page à page. S'il est possible de réaliser cela avec des boucles SPIP ordinaires, c'est tout de même relativement complexe.

### Exemple

Au plus simple, ce système est composé d'un critère et d'une balise :

- le critère **`{pagination}`** s'ajoute sur la boucle à paginer ;
- la balise **`#PAGINATION`**, placée dans une des parties optionnelles (« avant » ou « après ») de la boucle, affiche la « pagination ».

```
Haut du formulaire
<B_page>
    #PAGINATION
    <ul>
<BOUCLE_page(ARTICLES) {par date} {pagination}>
    <li>#TITRE</li>
</BOUCLE_page>
    </ul>
</B_page>
```

Si le site comporte 90 articles publiés, cette boucle affichera la liste des dix plus anciens articles, surplombée de liens conduisant vers la page qui affiche les dix suivants, les dix d'après, etc. Ces liens sont numérotés comme suit :

0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | ...

Le numéro à partir duquel les résultats sont affichés est passé dans l'url via un paramètre `{debut_page=x}` portant le même nom (ici, « page ») que la boucle concernée. (Ce paramètre est exploitable dans une autre boucle via le critère classique `{debut_page,10}`.)

A noter : le nombre total de liens affichés est limité ; des points de suspension permettent, le cas échéant, d'aller directement à la toute fin de la liste, ou de revenir au tout-début.

### Ancres de pagination

La balise `#PAGINATION` comporte une ancre html qui permet au navigateur d'afficher directement la partie de la page qui est paginée ; toutefois si l'on veut mettre les liens de pagination *en dessous* de la liste des articles, il faut pouvoir placer l'ancre *au-dessus* de la liste.

C’est à cela que sert la balise *#ANCRE_PAGINATION*, qui retourne l’ancre en question, et interdit à la balise `#PAGINATION` suivante d’afficher son ancre.

```
<B_page>
#ANCRE_PAGINATION
  <ul>
<BOUCLE_page(ARTICLES) {par date} {pagination}>
   <li>#TITRE</li>
</BOUCLE_page>
   </ul>
#PAGINATION
</B_page>
```

## Le nombre total de résultats

Dans une boucle avec le critère `{pagination}`, #TOTAL_BOUCLE affiche le nombre d’éléments effectivement retournés, c’est-à-dire 10 sur les pages pleines, et 10 ou moins sur la dernière page de résultats.

Pour afficher le nombre d’éléments qui **auraient été retournés** si le critère `{pagination}` n’avait pas été là, utilisez la balise **`#GRAND_TOTAL`**.

```
<B_pagination>
#ANCRE_PAGINATION
<BOUCLE_pagination (ARTICLES) {pagination}>
#TITRE <br />
</BOUCLE_pagination>
Il y a au total #GRAND_TOTAL articles, cette page en affiche
#TOTAL_BOUCLE
</B_pagination>
```

indiquera : « Il y a au total 1578 articles, cette page en affiche 10. »

## Changer le pas de la `{pagination}`

Le nombre standard de 10 éléments par page peut être modifié par un paramètre supplémentaire dans le critère.

Ainsi

```
<BOUCLE_page (ARTICLES) {pagination 5}>
#TITRE <br />
</BOUCLE_page>
```

retournera les titres de cinq articles à partir de *debut_page*.

Le paramètre en question peut lui-même être composé comme on le souhaite à partir d’autres balises, notamment `#ENV{xx}`, ce qui permet de faire un affichage à la demande très complet.

## La pagination dans les squelettes inclus

Si votre pagination doit fonctionner dans un squelette inclus, vous **devez** passer en paramètre de la commande `INCLURE` la formulation `{self=#SELF}` ; ceci permet au squelette inclus de se calculer avec la bonne valeur du paramètre `debut_xxx`, et se justifie qui plus est par un

besoin de sécurité (la balise `#PAGINATION` est en effet calculée à partir de l'URL complète de la page).

Depuis SPIP 1.9 le critère `{self=#SELF}` sera avantageusement remplacé par le couple `{env}` `{ajax}` [1].

**Nommer le critère de pagination**

Lorsque l'on utilise des squelettes inclus plusieurs fois dans la même page comme :

```
<BOUCLE_incluse(ARTICLES) {id_rubrique} {par titre}>
 <INCLURE{fond=motsgroupe5} {id_article} {ajax} {env}>
</BOUCLE_incluse>
```

et que cet *INCLURE* possède une pagination (d'où l'utilisation du critère `{env}`), lorsque vous cliquerez sur une pagination, vous verrez toutes les paginations se modifier en même temps.
Pour éviter celà, il suffit de **nommer le critère pagination** à l'intérieur du fichier inclu par un nom qui sera différent à chaque inclusion :

```
<B_groupe5>
 #ANCRE_PAGINATION
<BOUCLE_groupe5(MOTS) {id_groupe=5} {pagination 15 #ID_ARTICLE}>
 #TITRE
</BOUCLE_groupe5>
 <p class="pagination">#PAGINATION</p>
</B_groupe5>
```

On peut aussi vouloir spécifier dès l'appel de l'INCLURE le nom que l'on veut donner au critère pagination :

```
<INCLURE{fond=page_paginee, env, nom_p=_abc}>
<INCLURE{fond=page_paginee, env, nom_p=_def}>
```

dans page_paginee.html :

```
<B_a>
 #ANCRE_PAGINATION
<BOUCLE_a(ARTICLES) {pagination 25 #ENV{nom_p}}>
 #TITRE<br />
</BOUCLE_a>
 #PAGINATION
</B_a>
```

## Styles de la pagination

La pagination est constituée d'une série de liens, et d'un numéro de page correspondant à la page actuelle doté de la class « .on » : on définira donc les styles pour `a` et `.on` pour en personnaliser l'apparence.

## Choisir le modèle de pagination

La balise #PAGINATION accepte un paramètre `{modele}`, qui permet de modifier le résultat de la balise.

Ainsi `#PAGINATION{precedent_suivant}` affichera des liens vers les pages précédentes et suivantes. Les liens seront les suivants

page précédente | page suivante

`#PAGINATION{page}` affichera quelque chose de la forme suivante

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ...

`#PAGINATION{page_precedent_suivant}` affichera quelquechose comme

< 1 | 2 | 3 | 4 | 5 | 6 | >

Il est possible de définir d'autres modèles de pagination, qui devront s'appeler pagination_modele. On pourra, pour les fabriquer, s'inspirer de ceux livrés de base avec SPIP et situés dans le répertoire `prive/modeles/` (jusqu'à la version 1.9.2 de SPIP, ils étaient placés dans `dist/modeles/`) . Pour plus d'info, lire la documentation sur les modèles.

## P.-S.

**Attention :** beaucoup plus simple, ce mécanisme de pagination est aussi *incompatible* avec celui SPIP-Contrib' (http://www.spip-contrib.net/Pagination), qui lui a servi de matrice. Si vous utilisiez la contrib, il vous faudra revoir les squelettes concernés.

## Notes

[1] Rappel : pour que le critère `{ajax}` soit reconnu pour la balise `#PAGINATION`, il faut que cette dernière soit dans un élément de class *pagination*.

## La syndication de contenus

Juillet 2006 — maj : Janvier 2009

Le système de syndication (voir l'article « Les fichiers backend ») s'enrichit : il est désormais possible d'échanger l'adresse des documents joints aux articles (*podcasting*), de transporter d'un site à l'autre les mots-clés (*tags*) associés aux articles ainsi que leur rubrique (ou *catégorie*). On peut aussi, si on le désire, syndiquer le *contenu intégral* des articles.

Dans tout ce qui suit, on considère que le flux de syndication offert par le site source est suffisamment riche pour avoir prévu toutes les possibilités qu'offre notamment le squelette *dist/backend.html* de SPIP.

### Référencement rapide du site

On repère d'abord sur le site source l'URL de son flux de syndication au format RSS (ou Atom). Selon les cas, cette adresse est indiquée directement sur la page, et/ou est « découverte » automatiquement par le navigateur, qui affiche alors une icône caractéristique. Si les squelettes du site source le prévoient, SPIP peut aussi découvrir cette adresse de syndication, et il suffit d'indiquer l'adresse du site pour se voir proposer de le syndiquer.

Une fois la syndication activée, les articles présents dans le flux de syndication du site source sont repris sur le site récepteur.

### Décider ce que l'on veut émettre

Le webmestre du site source peut décider de ce qu'il met dans son flux RSS : ce choix peut bien évidemment se faire en modifiant le squelette *dist/backend.html*, ou en s'en inspirant pour créer un nouveau flux.

Mais la page de configuration dans l'espace privé propose une option très importante pour la syndication : elle permet de décider si le flux RSS du site comportera l'intégralité du contenu des articles, au format HTML, ou seulement un petit résumé (au format texte). Dans le premier cas (qui est la configuration par défaut du système), les articles récents du site sont entièrement lisibles avec un lecteur RSS : ils sont aussi entièrement « recopiables » d'un site à l'autre. Cela permet par exemple de réaliser automatiquement des sites miroirs, ou des sites composés d'un contenu produit sur d'autres sites.

### Décider ce que l'on veut récupérer

Si le site source ne diffuse pas son contenu intégral, il est bien évident que la syndication ne pourra pas en récupérer plus. Mais dans le cas d'une diffusion intégrale, SPIP peut récupérer ce contenu HTML et l'afficher, images comprises.

Site par site, on peut choisir ce que l'on veut récupérer par syndication : le contenu HTML des articles (si disponible) ou un simple résumé au format texte.

On peut également décider de ce que l'on fait des articles qui disparaissent des flux (qui sont en général limités aux 15 articles les plus récents du site) : on peut décider que SPIP les élimine de la base de données (après une période de deux mois), et/ou les passe immédiatement en mode « refusé ». Ces dernières options permettent par exemple de gérer un portail sur des flux très rapides (agences de presse, tags très populaires de sites de

photographie, etc) ; ou encore de maintenir un miroir fidèle d'un site (en éliminant les articles qui seraient dépubliés).

SPIP considère que si la date d'un élément de flux RSS est « très vieux » (plus d'un an) ou trop loin dans le futur (plus de 48 heures), c'est qu'il s'agit probablement d'une erreur. Il inscrit donc la date du jour à la place de celle qu'il a trouvé. On pourrait avoir besoin, cependant, de syndiquer des informations qui, à dessein, fournissent une date située à plus de 2 jours dans le futur. Par exemple, on peut se servir de fils RSS pour annoncer des évènements qui auront lieu dans le futur. C'est le cas, dans la galaxie SPIP, du site SPIP Party.

Pour syndiquer ses rendez-vous via le flux RSS dédié, une variable de personnalisation nommée `$controler_dates_rss`, qui vaut vrai par défaut.

Pour cela, dans votre fichier `config/mes_options.php`, ajouter la ligne `$controler_dates_rss = false;`, le test n'est plus effectué et les dates du futur, ou du « lointain » passé, seront bien prises en compte.

## Décider ce que l'on veut afficher

- **la *source* :**

De manière habituelle, `#NOM_SITE` représente le nom du site syndiqué, qui est donc la « source » de l'article. Cependant avec le développement des agrégateurs de contenu (les portails par exemple), la véritable source de l'article peut être un autre site. Le format RSS a prévu ce cas en définissant un champ `<source>` indiquant la véritable source de l'article. Le moteur de syndication de SPIP récupère ces données quand elles sont présentes, et les balises `#SOURCE` et `#URL_SOURCE` permettent d'afficher respectivement le nom et l'adresse de la source.

- **les *tags* :**

Si les mots-clés affectés aux articles sont correctement renseignés dans le flux RSS, ils sont récupérés par le site récepteur ; ils n'arrivent pas individuellement dans la table des mots-clés, mais sont stockés en vrac dans le champ `tags` de la table `spip_syndic_articles`.

Si le flux de syndication utilise la notation standard `<dc:subject>Tag</dc:subject>`, le tag est noté tel quel. SPIP pousse cette notion un peu plus loin en transmettant aussi l'adresse de la page du mot-clé, grâce aux microformats. Le tag est alors récupéré avec son lien, sous la forme : `<a rel="tag" href="adresse de la page du mot-clé">Mot-clé</a>`.

Sur le site destinataire, ces tags sont affichés dans l'espace privé sous le descriptif de l'article, et sont affichables sur le site public grâce à la balise `#TAGS` (on verra plus loin comment la filtrer pour faire un affichage sélectif des tags).

*Note :* SPIP prévoit une gestion spécifique des tags en provenance des sites http://del.icio.us/ (bookmarks collectifs), flickr (http://www.flickr.com/)(photographie) et connotea (http://www.connotea.org/ )(annotation d'articles scientifiques), de manière à pouvoir leur attribuer un URL (qui n'est pas prévu dans leurs flux RSS respectifs).

- **la rubrique :**

Dans de nombreuses applications (systèmes de blogs, répertoire de liens...) la catégorie (ou encore *directory*) d'un article est l'équivalent de ce que SPIP appelle la *rubrique*. Il était donc naturel d'utiliser la notion RSS standard de `<category>...</category>` pour faire connaître l'appartenance de notre article à telle ou telle rubrique.

De même qu'avec les tags, SPIP récupère cette information et l'affiche via `#TAGS`.

- **les documents joints :**

Les documents qui, dans l'espace privé, figurent en bas de la page de visualisation d'un article, soit dans la section « Documents » soit, pour les images, dans la section « Portfolio », sont eux aussi transmis dans le flux RSS, en utilisant la notion `<enclosure ... />`. C'est ce qu'on appelle le *podcasting*, désormais une fonction standard de SPIP [1].

L'information sur les *enclosures* est elle aussi récupérée dans la balise `#TAGS`, mais elle est affichée de façon différenciée dans l'espace privée, sous forme d'un petit trombone pour chaque fichier.

**Utiliser la balise `#TAGS`**

Comme on l'a vu ci-dessus, la balise `#TAGS` affiche en vrac les liens vers les mots-clés, la rubrique et les documents joints. Mais par la grâce des microformats, les liens vers chacun de ces concepts sont « marqués » de la façon suivante :
- `<a rel="tag" ...>` pour les tags/mots-clés
- `<a rel="directory" ...>` pour la rubrique/catégorie
- `<a rel="enclosure" ...>` pour les documents joints/podcast

Si l'on veut n'afficher que l'un de ces types de tags, on utilisera le filtre `afficher_tags` en précisant le type souhaité en argument :

```
[(#TAGS|afficher_tags{directory})]
[(#TAGS|afficher_tags{tag})]
[(#TAGS|afficher_tags{enclosure})]
```

(Par défaut `[(#TAGS|afficher_tags)]` est équivalent à `[(#TAGS|afficher_tags{'tag,directory'})]`.)

Pour les documents joints, le filtre spécifique `afficher_enclosures` permet d'afficher les trombones plutôt que des liens classiques :

```
[(#TAGS|afficher_enclosures)]
```

**Manipuler le contenu HTML des articles syndiqués**

Cas pratique : notre site syndique un photoblog, qui diffuse systématiquement un petit commentaire suivi de la photographie. Cette dernière se présente sous la forme d'une balise HTML `<img ...>`. Une fois ce photoblog syndiqué en version HTML complète dans notre site, nous pouvons décider de n'afficher que la photo, sans le commentaire. Il nous faut alors extraire la balise `<img />` ; cela peut se réaliser grâce au filtre `extraire_balise{xxx}`, qui récupère la première balise HTML `<xxx />` d'un contenu.

A partir de là tout est possible :
- `[(#DESCRIPTIF|extraire_balise{img})]` affiche la photo ;
- `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{src})]` son URL ;
- `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{width})]` sa largeur ;
- on peut même en modifier le style avec, par exemple :

```
[(#DESCRIPTIF|extraire_balise{img}|
        inserer_attribut{style,'border: double red 4px;'})]
```

Remarque : les contenus HTML provenant d'un site extérieur sont par définition considérés comme « non contrôlés », et donc potentiellement problématiques s'ils contiennent des balises mal fermées, ou du javascript ; SPIP leur applique donc systématiquement le filtre `safehtml` avant affichage.

**Autres filtres liés à la syndication**

Ces filtres permettent de convertir les tags de syndication d'un format à un autre, afin de pouvoir par exemple remettre au format RSS des « tags » récupérés par syndication, et enregistrés dans notre base de données sous forme de microformats : `tags2dcsubject`, `enclosure2microformat`, `microformat2enclosure`.

## Références


- RSS 2.0 (http://www.stervinou.com/projets/rss/)
- microformats (http://microformats.org/)
- rel=tag (http://microformats.org/wiki/rel-tag)
- rel=enclosure (http://microformats.org/wiki/rel-enclosure)
- rel=directory (http://microformats.org/wiki/rel-directory)


## Notes

[1] Attention les fichiers eux-mêmes ne sont pas recopiés : seules leur URL et les données associées (titre, taille, format) sont récupérées. Par ailleurs il faut signaler, pour les puristes, qu'on prend ici quelque liberté avec la norme RSS, qui interdit normalement d'avoir plusieurs *enclosures* dans un même article.

## Le calendrier de SPIP

Août 2005 — maj : Décembre 2007

SPIP permet de visualiser dans l'espace public des calendriers avec le même affichage que celui de l'espace privé, et plus généralement de construire des agendas quelconques bénéficiant des outils de mises en pages de ces calendriers. Cette possibilité est fournie par un nouveau critère de boucle et trois nouveaux filtres.

Le critère `agenda` possède un nombre variable d'arguments et peut donc s'écrire de deux façons :
- {agenda *datetime*, *type*, *AAAA*, *MM*, *JJ*}
- {agenda *datetime*, periode, *AAAA*, *MM*, *JJ*, *AAAA2*, *MM2*, *JJ2*}

Les deux premiers arguments sont :

1. un nom de champ SQL de type *datetime* (par exemple `date` ou `date_redac` pour la table Articles et `date_time` pour les Breves) ; cet argument est obligatoire.
2. un *type* de calendrier (`jour`, `semaine`, `mois` ou `periode`) ; la valeur par défaut est `mois`.

Ces deux arguments doivent être littéraux (autrement dit ils ne peuvent être calculés dynamiquement par `#ENV` ou toute autre balise).

Les trois prochains arguments sont optionnels et peuvent être indiqués par des balises (avec et sans filtre) :

1. *YYYY* une chaîne d'exactement 4 chiffres indiquant une année ;
2. *MM* une chaîne d'exactement 2 chiffres indiquant un mois ;
3. *JJ* une chaîne d'exactement 2 chiffres indiquant un jour.

Si ces valeurs sont omises ou nulles, elles sont remplacées par celle de la date courante. Ces paramètres représentent la date d'un jour dans la période choisie :
- si le type est `jour`, on affichera les éléments dont la date correspond au jour spécifié,
- si le type est `semaine`, on affichera les éléments dont la date est dans la semaine qui contient le jour spécifié,
- si le type est `mois`, on affichera les éléments dont la date est dans le mois qui contient le mois spécifié (dans ce cas, le paramètre de jour *JJ* est facultatif).

Par exemple :

```
<B_semaine>
<ul>
<BOUCLE_semaine(ARTICLES) {agenda date, semaine} {par date}>
<li>#TITRE</li>
</BOUCLE_semaine>
</ul>
</B_semaine>
```

affiche une liste des articles publiés dans la semaine actuelle.

```
Haut du formulaire
<BOUCLE_art_principale(ARTICLES) {id_article}>
<B_mememois>
<ul>
<BOUCLE_mememois(ARTICLES) {agenda date, mois, (#DATE|annee),
(#DATE|mois)} {par date}>
<li>#TITRE</li>
</BOUCLE_mememois>
</ul>
</B_mememois>
</BOUCLE_art_principale>
```

affiche une liste des articles publiés le même mois que l'article actuel.

Dans le cas où l'argument *type* (le deuxième argument) vaut `periode`, trois autres arguments peuvent être spécifiés à la fin du critère. Alors :
- *YYYY*, *MM*, *JJ* correspondront à la date de début de la période,
- *YYYY2*, *MM2*, *JJ2* correspondront à la date de fin de la période.

Si le deuxième trio spécifiant la date de fin est omis, la date courante sera prise comme date de fin, et si le premier trio est également absent, la période de sélection couvrira toute la vie du site (pour les sites avec beaucoup d'articles, le temps d'exécution risque d'être excessif si d'autres critères n'en limitent pas le nombre).

Par exemple :

```
<BOUCLE_art_principale(ARTICLES) {id_article}>
<B_apres>
<ul>
<BOUCLE_apres(ARTICLES) {agenda date, periode, (#DATE|annee),
(#DATE|mois), (#DATE|jour)} {par date}>
<li>#TITRE</li>
</BOUCLE_apres>
</ul>
</B_apres>
</BOUCLE_art_principale>
```

liste les articles qui ont été publiés après l'article actuel.

Pour mettre en page les éléments sélectionnés par une boucle (en particulier une boucle avec un critère **agenda**) sous forme de calendrier, SPIP fournit trois filtres. Ces filtres fournissent un affichage similaire au calendrier de l'espace privé avec le même système de navigation.

- Le filtre `agenda_memo` s'applique sur une balise de date (par exemple `#DATE` ou `#DATE_MODIF`) et prend quatre paramètres :

1. un descriptif
2. un titre
3. une URL représentant l'élément ayant ce titre et ce descriptif (par exemple `#URL_ARTICLE`)
4. un nom de *classe* CSS

Si la balise sur laquelle `agenda_memo` s'applique n'est pas nulle, il se contente de mémoriser la date et les trois premiers arguments dans un tableau indexé par le dernier argument (le nom de classe CSS) et ne retourne rien (aucun affichage).

L'utilisation du dernier argument comme index pour l'élément à mémoriser permet d'avoir plusieurs calendriers par page. De plus, la classe spécifiée ici sera attribuée à cet élément dans l'affichage en calendrier fourni par **agenda_affiche**. Ainsi, on peut donner des styles différents aux éléments. La feuille *calendrier.css* fournit 28 styles différents qui donnent un exemple de différents styles de calendrier.

- Le filtre `agenda_affiche` s'applique sur une balise retournant le nombre d'éléments à afficher (en général `#TOTAL_BOUCLE`) et prend trois paramètres :

1. un texte qui sera affiché si la balise filtrée ne retourne rien (0 élément à afficher) ;
2. un type de calendrier (`jour`, `semaine`, `mois` ou `periode`) ;
3. des noms de *classes* CSS utilisées dans l'appel du filtre précédent qui permettent de filtrer les éléments à afficher.

Si la balise filtrée est nulle, ce filtre retourne son premier argument. Sinon il retourne les éléments mémorisés par le filtre **agenda_memo** mis en page dans un calendrier du type demandé.

Seul les éléments indexés par **agenda_memo** avec une des classes CSS indiquées dans le dernier argument seront affichés (ou alors tous les éléments si ce paramètre est omis). Ainsi on peut filtrer les éléments pour les répartir dans plusieurs calendriers sur la même page.

Le type `periode` restreindra l'affichage à la période comprise entre le plus vieil élément et le plus récent effectivement trouvés, ce qui permet de donner dans le critère une période très large sans récupérer un affichage trop long.

Exemple :

```
<BOUCLE_memorise(ARTICLES) {agenda date, semaine}{par
date}>[
(#DATE|agenda_memo{#DESCRIPTIF,#TITRE,#URL_ARTICLE})
]</BOUCLE_memorise>
[(#TOTAL_BOUCLE|agenda_affiche{<:aucun_article:>,semaine})]
<//B_memorise>
```

affiche les articles publiés dans la semaine actuelle sous forme de calendrier.

- Enfin, le filtre `agenda_connu` teste si son argument est l'un des quatre types de calendriers connus (`jour`, `semaine`, `mois` ou `periode`).

Ce critère et ces filtres sont utilisés par les nouveaux squelettes `agenda_jour.html`, `agenda_semaine.html`, `agenda_mois.html` et `agenda_periode.html`, appelés à partir du squelette `agenda.html` qui indique dans son en-tête les feuilles de style et fonctions javascript nécessaires (mais remplaçables à volonté). Ces squelettes fournissent donc un exemple représentatif d'utilisation.

## Exposer un article dans une liste

Avril 2004 — maj : Janvier 2008

La balise **#EXPOSE** permet de mettre en évidence, dans un menu ou dans une liste, l'objet principal de la page où l'on se trouve.

L'objet qui est « exposé » est l'article (ou la brève, la rubrique, le mot-clé ou l'auteur) qui appartient au « contexte » courant. Dans le cas des rubriques, la traversée de la hiérarchie est gérée, ce qui permet d'« exposer » l'arborescence des rubriques qui contient l'article affiché.

Par défaut, SPIP remplace la balise `#EXPOSE` par « **on** » si l'objet correspond au contexte ; sinon la balise est simplement ignorée.

Toutefois la balise `#EXPOSE` accepte un ou deux arguments, qui permettent de préciser ce qui doit s'afficher pour l'objet exposé, et ce qui doit s'afficher pour les autres objets. Ainsi `[(#EXPOSE{oui,non})]` affichera « oui » pour l'objet exposé, et « non » pour les autres.

### Afficher différemment l'article exposé

Utilisée simplement, la balise `#EXPOSE` permet, par exemple, dans un menu de navigation, de changer l'apparence du lien vers l'article que l'on est précisément en train de consulter. Pour modifier le style des liens de ce menu, on placera la balise `#EXPOSE` de la manière suivante dans le squelette `article.html` :

```
<BOUCLE_principale (ARTICLES) {id_article}>
      <B_menu>
      <ul>
            <BOUCLE_menu (ARTICLES) {id_rubrique}>
            <li>
                  <a href="#URL_ARTICLE"[ class="(#EXPOSE)"]>
                  #TITRE
                  <a>
            </li>
            </BOUCLE_menu>
      </ul>
      </B_menu>
      #TEXTE
</BOUCLE_principale>
```

avec les styles suivants :

```
a { color: blue; }
a.on { color: red; font-weight: bold; }
```

L'article ainsi exposé se distinguera dans la liste par un affichage visuellement différent.

## Désactiver le lien exposé

Avec un peu d'astuce il est possible de désactiver le lien vers l'article exposé et dans le même temps de choisir le style à appliquer :

```
<B_menu>
<ul>
      <BOUCLE_menu(ARTICLES){id_rubrique}>
      <li>
            <#EXPOSE{span,a href="#URL_ARTICLE"}[ class="(#EXPOSE)"]>
            #TITRE
            </#EXPOSE{span,a}>
      </li>
      </BOUCLE_menu>
</ul>
</B_menu>
```

créera le code HTML suivant, où les balises `<a>` sont remplacées par des `<span>` :

```
<ul>
<li><a href="article1.html">Tout sur ma soeur</a></li>
<li><span class="on">Tout sur moi</span></li>
<li><a href="article3.html">Tout sur mon frère</a></li>
</ul>
```

ce qui s'affiche ainsi :

- Tout sur ma soeur
- Tout sur moi
- Tout sur mon frère

## La « popularité » des articles

Juillet 2002 — maj : Décembre 2007

### Comment décompter des visites

Des centaines de méthodes statistiques existent pour décompter des visites sur un site donné. La plupart donnent des courbes horaires, ou par jour, qui permettent de savoir si son site « monte » ou « descend », et de vérifier qu'il y a plus de gens sur le net en fin d'après-midi et dans la semaine, que le week-end ou la nuit...

Notre objectif est un peu différent : il s'agit d'attribuer à chaque article une valeur de « popularité » reflétant assez rapidement une tendance, et permettant de comparer l'activité de différents articles, soit de manière globale sur tout le site (hit-parade), soit à l'intérieur d'une rubrique, soit parmi les articles d'un même auteur, etc.

La méthode retenue est la suivante (rassurez-vous, vous pouvez sauter cette explication si vous n'êtes pas à l'aise en maths) :
- chaque visite sur un article ajoute un certain nombre de points à cet article ; 1 point si c'est un article que l'on consulte depuis le site lui-même en suivant un lien, et 2 points si c'est une « entrée directe » depuis un site extérieur (moteur de recherche, lien hypertexte, syndication...)
- toutes les 10 minutes, le score obtenu est multiplié par un petit facteur d'escompte, qui fait qu'un point attribué par une visite à 10h12 le mercredi ne vaut plus, le lendemain à la même heure, qu'un demi-point, et, le vendredi à 10h12, un quart de point... ;
- le tout est calculé de manière à ce que, dans l'hypothèse où l'article reçoit toujours le même nombre *x* de visites par unité de temps, son score se stabilise sur cette valeur *x*. Autrement dit, si la fréquentation de l'article est stationnaire, sa popularité finira par refléter exactement son nombre de visites par jour (modulo le score 2 donné pour les entrées directes) ;
- cette popularité s'exprime de deux manières : l'une, la popularité_absolue, exprime le score en question (évaluation de la fréquentation quotidienne de l'article) ; l'autre, la popularité_relative, un pourcentage relatif à l'article du site ayant la plus forte popularité (popularité_max) ;
- enfin, la somme de toutes ces valeurs (absolues) sur le site donne la popularité_site, qui permet de comparer la fréquentation de deux sites sous spip...

### Boucles et balises

Des balises permettent de récupérer et d'afficher ces valeurs dans vos squelettes. La boucle ci-dessous résume l'ensemble de ces balises :

```
<BOUCLE_pop(ARTICLES){id_article}{popularite>0}>
<h5>Popularité</h5>
Cet article a une popularité absolue égale à #POPULARITE_ABSOLUE, soit
#POPULARITE % de #POPULARITE_MAX. Au total, ce site fait environ
#POPULARITE_SITE visites par jour.
</BOUCLE_pop>Haut du formulaire
```

La balise la plus utile est **#POPULARITE** puisqu'elle donne un pourcentage représentant la popularité de l'article relativement à l'article le plus populaire du site. Cela permet ainsi de réaliser facilement des classements compréhensibles par tous (avec des valeurs allant de 0 à 100). Les autres balises donnent des valeurs absolues, plus difficiles à interpréter par les visiteurs du site.

*Note :* bien que les données soient représentées, dans la base de spip, sous forme de nombres réels, le rendu de toutes ces balises est toujours donné sous la forme d'un nombre entier, ce qui donnera, sur des sites *très* peu fréquentés (sites de tests, notamment), des choses amusantes du genre :

*« Cet article a une popularité absolue égale à 1, soit 17 % de 2. Au total, ce site fait environ 5 visites par jour. »*

Enfin, un critère de tri peut se révéler utile : **`{par popularite}`**, que l'on utilisera par exemple de la manière suivante pour afficher la liste des 10 articles les plus populaires de la rubrique courante :

```
<BOUCLE_hitparade(ARTICLES){id_rubrique}{par popularite}{inverse}{0,10}>
      <li>#TITRE (popularité : #POPULARITE %)</li>
</BOUCLE_hitparade>
```

(On enlèvera `{id_rubrique}` pour afficher un hit-parade du site entier.)