Documentation SPIP

Contribuer

Contribuer

Traducteurs et développeurs, contribuez à améliorer SPIP !

Traduire SPIP 4
Les auto-références multilingues de la documentation 4
Comment traduire SPIP ? 5
Traduire l’aide en ligne 6
Conseils généraux pour la traduction de ce site 7
Traduction des images 9
Le développement de SPIP et ses outils 11
Participer à la documentation des balises, critères, filtres de SPIP 11
Étendre SPIP 13
Pourquoi réaliser un plugin ? 20
Réaliser un premier plugin 22

Traduire SPIP

Si vous désirez participer aux traductions en cours ou à venir, aider à traduire ou relire la documentation ou l'aide en ligne, vous êtes les bienvenus !

Les auto-références multilingues de la documentation

Mars 2009

La version SPIP 2.0 ayant introduit la possibilité de surcharger le traitement du `hreflang` des raccourcis, le présent site consacré à la documentation multilingue de SPIP offre à présent une méthode toute simple pour référencer ses propres articles.

Auparavant, lorsqu'un article A référençait un article B de la documentation, les traducteurs étaient devant une situation insatisfaisante et propice à l'oubli. Ils traduisaient d'abord A et obtenait un autre article, disons C, dans lequel ils laissaient la référence à B, n'ayant pas encore l'URL de la traduction de B, puisqu'elle n'existait pas encore. Le temps que cette traduction, disons D, soit effectuée puis publiée, le souvenir que C contient une référence à B qu'il faut remplacer par D a toutes les chances d'être oublié.

A présent, les rédacteurs de la documentation doivent impérativement utiliser la notation `[{}->artN]` pour référencer l'article N de la documentation, et les traducteurs peuvent recopier ce raccourci sans plus avoir à le modifier un jour. En effet, si l'article N n'existe pas dans la langue du texte qui utilise ce raccourci, celui-ci produira l'URL de l'article N (ce qui est le mieux que l'on puisse faire), mais si une traduction existe, ce sera automatiquement l'URL de cette traduction qui sera produite. Plus précisément, ce comportement sera strictement observé lors d'une référence à partir de l'espace privé, tandis que dans l'espace public on ajoute la contrainte supplémentaire que la traduction soit publiée, afin de produire une URL disponible à coup sûr.

Comment est-ce possible ? En fait, depuis longtemps les raccourcis de SPIP autorisaient l'usage des accolades pour spécifier la valeur de l'attribut `hreflang`. Mais SPIP interprète de plus une paire d'accolades vides comme un `hreflang` égal à la langue du texte où figure le raccourci. La surcharge introduite pour ce site consiste simplement à chercher si l'article désigné par le raccourci existe dans la langue indiquée par le `hreflang`. Si c'est le cas, le raccourci `->artN` est remplacé par `->artP`, où `P` est la traduction de `N`.

Comment traduire SPIP ?

Janvier 2003 — maj : Décembre 2007

Si vous voulez traduire SPIP, il y a deux manières de procéder :

Avec l'interface de traduction en ligne

La méthode qui a notre préférence est de procéder en ligne sur le site de la documentation de SPIP, **à travers l'interface de traduction** (http://www.spip.net/trad-lang/trad_lang.php). Pour vous servir de cette interface il est nécessaire que vous vous inscriviez d'abord comme rédacteur (http://www.spip.net/rubrique4.html) sur ce site. Ensuite, voyez si votre langue est déjà en cours de traduction, auquel cas il faudra vous joindre à l'équipe de traduction correspondante (dans l'espace privé, ou via la liste spip-trad - http://listes.rezo.net/mailman/listinfo/spip-trad/) ; sinon, vous pouvez créer la langue en question et démarrer sans attendre.

Attention : lorsqu'une variable est présente dans une chaîne, elle est notée entre deux « `@` ». Vous aurez, par exemple, à traduire la chaîne : `'Erreur dans la sauvegarde (@type@ @id_objet@) !'`. Il ne faut pas modifier les caractères entre les `@`, car ils seront remplacés pour former la chaîne finale : « Erreur dans la sauvegarde (article 7) ».

C'est à peu près tout ce qu'il faut savoir pour traduire SPIP. Pour le reste, les qualités de la traduction sont celles qu'on attend de SPIP : lisibilité, accessibilité à un public non initié (éviter tant que possible le jargon), correction de la langue (éviter les fautes de frappe ou d'orthographe), etc. Et bien sûr, il faut comprendre et connaître SPIP un minimum, et réfléchir au sens de ce que l'on traduit dans son contexte d'utilisation, car un faux sens est vite arrivé...

Il y a au total environ un millier de lignes à traduire (ça va plus vite qu'on ne pense...).

Hors connexion, directement dans les fichiers de langue

Si ce fonctionnement en ligne vous pose problème, et que vous préférez effectuer toute la traduction chez vous, hors connexion, vous pouvez le faire en téléchargeant SPIP et en travaillant à partir des fichiers de langue que vous éditerez dans un simple éditeur de texte, puis que vous nous enverrez par mail pour intégration au système de développement.

La distribution de SPIP contient un répertoire `ecrire/lang/` dans lequel se trouvent, pour chaque langue, trois fichiers **`spip_xx.php`**, **`ecrire_xx.php`** et **`public_xx.php`**, où « `xx` » est le code de la langue. `spip_xx.php` est utilisé par le logiciel lui-même, `ecrire_xx.php` pour les chaines utilisées uniquement dans l'espace privé et `public_xx.php` par les squelettes de la distribution.

Ces fichiers sont organisés d'une manière relativement simple. Chaque chaîne de caractères disponible à la traduction y est représentée par une ligne du type :

```
'avis_chemin_invalide_1' => 'Le chemin que vous avez choisi',
```

qu'il faut traduire en :

```
'avis_chemin_invalide_1' => 'Li camino chi you have chost',
```

etc.

Quelques particularités qui méritent votre attention :

- comme pour la traduction en ligne, lorsqu'une variable est présente dans une chaîne, elle est notée entre deux « @ ». Par exemple : `'avis_erreur_sauvegarde' => 'Erreur dans la sauvegarde (@type@ @id_objet@) ! '`. Il ne faut pas modifier les caractères entre les @, car ils seront remplacés pour former la chaîne finale : « Erreur dans la sauvegarde (article 7) » ;
- Les caractères accentués sont notés indifféremment en entités HTML (`&eacute;`) ou entités unicode (`&#233;`) ;
- Le caractère apostrophe (`'`) doit être précédé d'un backslash (`\'`).

Traduire l'aide en ligne

Avril 2003 — maj : Janvier 2007

L'aide en ligne est constituée d'une collection d'articles définis dans l'espace privé de ce site, et qui suivent un format précis. Pour la traduire adressez-vous à la liste des traducteurs (http://listes.rezo.net/mailman/listinfo/spip-trad/), qui sauront vous guider sur le format.

Conseils généraux pour la traduction de ce site

Janvier 2007 — maj : Décembre 2007

Le site www.spip.net a changé de présentation en janvier 2007. Voici quelques recommandations relatives à la façon dont ces nouveaux squelettes organisent et présentent le contenu de ce site.

Le rubriquage de www.spip.net

Le rubriquage du secteur français « Documentation en français » sert de référence. Il est visible *in extenso* à la page « plan de site ». Il convient donc de le reproduire dans chaque secteur de langue.
Cependant les secteurs peu traduits contiennent peu d'articles : mieux vaut alors ranger ces quelques articles directement à la racine du secteur, sans s'embarasser de rubriques.

Sur le site public, les rubriques et sous-rubriques sont classées par leurs numéros de titre.
Les quatres premières rubriques à la racine du secteur constituent les onglets de la barre de navigation.

Seules les rubriques principales bénéficient de logos. Ils ne contiennent pas de texte, ce qui permet de les utiliser dans toutes les langues. Il est donc conseillé de récupérer les icones de la version française pour les installer dans les rubriques correspondantes des autres langues. Le site n'utilise pas les « logos pour survol ».

Classement des articles dans les rubriques

Il y a deux façons de « classer » les articles sur www.spip.net :

- `(par num titre)` dans certaines rubriques, il est nécessaire de « forcer » le classement des articles selon une certaine logique (par exemple dans les tutoriels et le manuel de référence), qui est celle d'une explication qui se déroule sur plusieurs articles successifs ; il suffit de numéroter les titres des articles, avec un numéro suivi d'un point et d'un espace (« 1. Premier article », « 2. Deuxième article »...) ;

Les traducteurs et traductrices de la version espagnole ont adopté une numérotation de 10 en 10 très pratique : « 10. Premier article », « 20. Deuxième article ». Il est conseillé de procéder ainsi, cela facilitant l'insertion d'articles à l'intérieur de ce classement par la suite.

- `(par date inverse)` dans les rubriques qui ne nécessitent pas une lecture « à la suite » des articles (FAQ, trucs et astuces... et « Évolutions et mises à jour »), on ne numérote pas les articles, ce qui permet de présenter en haut de liste les derniers articles publiés, les plus anciens en bas.

Dans les squelettes, la présentation s'adapte automatiquement selon la présence ou non de la numérotation.

Rythmer la présentation de la page d'accueil

Le site est doté d'une page d'accueil principale et d'une page d'accueil par langue. Celles-ci ne sont pas qu'une simple liste de rubriques et d'articles : elles contiennent des pavés qui rythment la page et mettent en valeur certains éléments, de façon à présenter SPIP, mais aussi les ressources et les nouveautés dans chaque langue.

Le premier texte affiché est celui de la rubrique-secteur de la langue. Il présente SPIP et l'objet de ce site. Pour le modifier : « modifier la rubrique », champ « texte ».

Référencement de ressources externes

Les principaux sites de référence sont présentés dans les rubriques du secteur français « Documentation en français », et sont visibles à cette page « Liens utiles ». Bien qu'ils soient principalement francophones, on peut (mais ce n'est pas obligatoire) référencer ces sites en traduisant leurs descriptifs dans les autres secteurs de langue.
Mieux cependant vaut référencer les ressources disponibles dans votre langue.

Chaque langue dispose d'une liste de discussion (spip-#`lang`@rezo.net). Ces listes sont signalées au cas par cas dans la rubrique « Questions et réponses » de la langue concernée. Celles-ci doivent être dotées du mot-clé « spip-#lang@rezo.net » pour être présentées en bonne place en page d'accueil.
Les forums disponibles dans chaque langue sont référencés dans la même rubrique, et dotés du mot-clé « Forum ».

Pour faire un lien dans le corps d'un texte, on préférera employer le raccourci SPIP correspondant au site référencé. Par exemple, plutôt que d'écrire `[SPIP Contrib->http://www.spip-contrib.net]`, on écrira `[->site57]` (ce qui s'affichera ainsi : SPIP - Contrib).

Le glossaire

Les boucles, balises, filtres, etc. de SPIP sont listés sous forme de mots-clés, qui sont associés aux articles qui les documentent en français, afin de constituer un glossaire complet du langage SPIP. Dans les autres langues, ces mots-clés sont inutiles, le squelette du glossaire proposant automatiquement les traductions disponibles.

Traduction des images

Décembre 2003 — maj : Janvier 2007

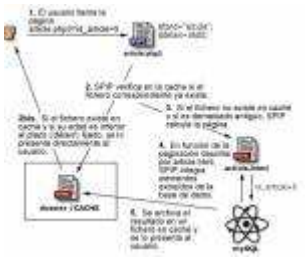
Utiliser les mêmes fichiers image

Idéalement, on ré-utilisera le même fichier image que dans l'article d'origine. Par exemple, pour la traduction de l'article 3336 (http://www.spip.net/ca_article3336.html), les images seront conservées telles qu'elles, c'est-à-dire appelées avec les mêmes raccourcis (`<img2048|center>`) que dans l'article d'origine. Pour qu'elles soient ainsi utilisables dans toutes les langues, on ne leur attribue ni titre, ni description.

Traduire le texte des images

Autant que possible, on évitera de créer des images contenant du texte. Si une explication textuelle est nécessaire, mieux vaut la placer dans les champs prévus (« *titre* » et « *description* ») de façon à faciliter les traductions. Ainsi un schéma nécessitant des explications sera plus aisé à traduire s'il n'affiche que des numéros, et que les explications correspondant à ces numéros sont reportées dans le champ de description : on fait alors une copie de l'image pour en traduire le titre et la description directement dans l'espace privé, sans qu'il soit nécessaire de faire appel à un logiciel de retouche d'images.

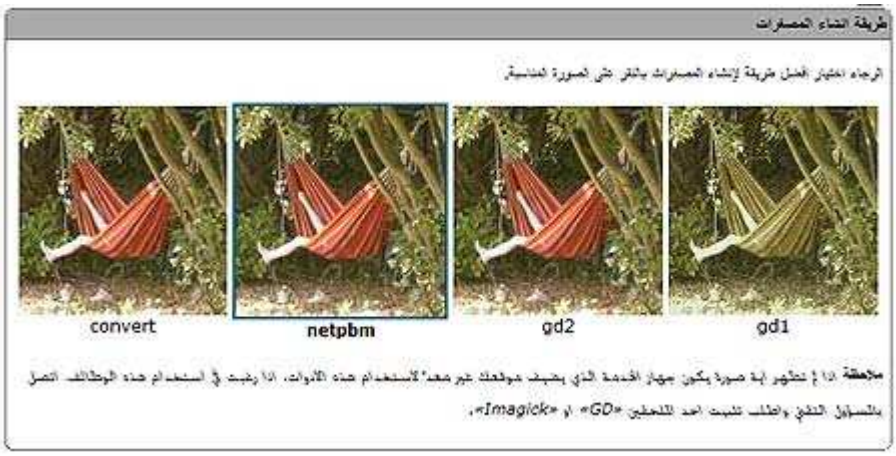
Cependant, il est parfois nécessaire de « traduire » certaines images, lorsqu'elles contiennent du texte. Pour traduire les textes sur des images de la documentation de SPIP, il faut utiliser un logiciel de retouche d'images.



Par exemple, le document ci-dessus a été réalisé avec Fireworks à partir de l'image en français (de l'article 877 - http://www.spip.net/fr_article877.html), où les textes sont traduits en espagnol (pour l'article 93 - http://www.spip.net/es_article93.html). Vous pouvez reprendre ce document pour en faire plus rapidement une traduction vers une autre langue.

Des captures d’écran dans la bonne langue

De nombreuses illustrations de la documentation sont des captures d’écran de l’espace privé de SPIP. Lorsque le texte y est lisible, mieux vaut réaliser une nouvelle capture d’écran dans la même langue que celle de l’article qu’elle illustre, afin de faciliter la compréhension.



Ainsi la capture d’écran ci-dessus est utilisée dans l’article 3049 (http://www.spip.net/ar_article3049.html) (en langue arabe), pour remplacer avantageusement celle de l’article 3024 (http://www.spip.net/fr_article3024.html) (en langue française).

Le développement de SPIP et ses outils

Les différents outils utilisés pour développer SPIP.

Participer à la documentation des balises, critères, filtres de SPIP

Juillet 2010

Toute personne peut participer. Les pré-requis sont réduit à l'essentiel :

- avoir un compte sur spip.net
- connaitre une balise, un critère, ... non documenté et connaitre un cas d'utilisation associé.
- avoir lu cet article pour comprendre les quelques spécificités liées à la documentation

Titrer l'article

- un filtre : **`|nom_du_filtre`**
- une balise : **`#NOM_DE_LA_BALISE`**
- un critère : **`{nom_du_critère}`**

Résumer l'élément documenté

Le **chapeau** sert à introduire le concept présenté :

- une ou deux phrases de résumé du texte ou de présentation de l'élément. Ce texte est utilisé par les modeles `<critereXX>`,`<baliseXX>` (exemple de cas d'utilisation « La boucle ARTICLES »)

Décrire le concept

Le champ **Texte** sert à la description exhaustive (autant que possible) de l'élément documenté :

- une majuscule en début de phrase
- encadrer par les tags **<code>** et **</code>** le nom de l'élément (exemple : « *la balise `#BALISE` ...* »), les urls de l'espace privé (exemple : « *en allant sur `?exec=config_fonctions` ...* »), parfois le résultat affiché (exemple : « *la date retournée sera du type `1970-01-01 00:00:00`* »)
- privilégier les tags (plus lisibles) **<code>** et **</code>** à **<cadre>** et **</cadre>**
- hiérarchiser son texte (intertitres, paragraphes)

- Globalement

- s'en tenir à documenter le seul élément dont il est question (ne pas vouloir tout dire, éviter les digressions, les trucs et astuces, ...)
- vérifier *a minima* (en lisant le code) les informations fournies

Utiliser les mots clefs

- Indiquer la version de spip ayant introduit l'élément documenté. Pour cela utiliser l'écriture : `(depuis [{}->artxxxx])` où « *xxxx* » est l'id de l'article présentant la version de SPIP et où les `{}` permettent, si nécessaire, un renvoi automatique sur la traduction de l'article, facilitant ainsi le travail des traducteurs (voir : Les auto-références multilingues de la documentation).
- Pour les balises
    - Indiquer si c'est une balise « *statique* » (issue de la base de donnée) ou bien « *dynamique* » (calculée, traitée par une fonction)
- indiquer si nécessaire (pour une balise par exemple) le type de boucle concernée
- faire au mieux `:-)`

... à suivre

Étendre SPIP

Mai 2007 — maj : Mars 2009

Si vous voulez étendre SPIP et en particulier y contribuer, les points importants à retenir sont les suivants : *c'est un outil déjà utilisé par des milliers de personnes, qui a un passé et un avenir, et qui est un travail collectif*. Certains choix ont été opérés, pas toujours heureux, mais qui donnent une cohérence à l'ensemble du code, facilitant son évolution. Lorsqu'un choix se révèle franchement incompatible avec une nouvelle technologie intéressante à intégrer, une réécriture est mise en chantier mais peut prendre plusieurs mois, d'où une incohérence prolongée. On comprend donc qu'il faut éviter autant que possible de telles décisions, et accepter qu'ici où là une nouvelle règle soit violée, la réécriture de cette partie du code n'ayant pas encore eu lieu. On comprend aussi que le style de programmation varie quelque peu d'un endroit à un autre, mais que se construit une identité commune en perpétuel devenir, un logiciel utile et utilisé n'étant pas de saintes écritures figées sur papier bible, mais l'expression d'une communauté vivante et libre.

Organisation des sources

Depuis la version 1.9, il est possible de modifier le comportement de SPIP sans altérer ses sources, grâce au *chemin d'accès* défini par la constante `SPIP_PATH`. Toute extension de SPIP doit opérer ainsi afin qu'un test d'intégration soit débrayable par simple redéfinition du chemin d'accès, la distribution de SPIP étant supposée en lecture seule. Si vous êtes convaincu qu'un certain comportement ne peut être obtenu ainsi, écrire à la liste de discussion spip-dev (http://listes.rezo.net/mailman/listinfo/spip-dev).

Les contributions à SPIP sont à décrire par un article sur Spip-Contrib et leur contenu est à déposer soit en tant que pièces jointes à cet article, soit, ce qui est mieux, sur spip-zone (http://zone.spip.org/trac/spip-zone/) qui fournit un serveur Subversion piloté par Trac.

Pour étendre SPIP ainsi, il faut bien comprendre l'organisation de ces répertoires et le rôle de ses fichiers. C'est le but du présent article, mais on lira aussi avec profit l'article annonçant SPIP 1.9 qui l'a inaugurée.

La racine d'une distribution de SPIP comporte essentiellement :

- un fichier `spip.php`, alias `index.php`, gérant la compatibilité avec les anciennes versions, chargeant le fichier d'initialisation `ecrire/inc_version.php` et passant immédiatement la main au script principal `ecrire/public.php` ;
- un répertoire `ecrire` comportant exclusivement des fichiers interprétables côté serveur (PHP et SQL) ;
- un ou plusieurs (selon les versions) répertoires comportant des fichiers interprétables côté client (HTML, Javascript, feuilles de style, images de différents formats) ainsi que les patrons de mise en page nommés *squelettes*. Ces squelettes sont interprétés des deux côtés : il s'agit de fichiers composés d'un format MIME complété de quelques directives SPIP, directives traitées côté serveur afin d'envoyer au client un texte purement MIME (la plupart du temps du HTML, mais aussi du RSS, du SVG, du ICS etc).
- quatre répertoires vides à l'installation, qui contiendront les données, temporaires ou permanentes, nécessaires à la vie du site.

Rôles des répertoire `prive` et `squelettes-dist` **et leurs sous-répertoires**

Ils contiennent les fichiers déterminant la présentation de SPIP, pour l'espace privé et l'espace public respectivement. Avant SPIP 2.0, ces deux répertoires n'en formaient qu'un seul, nommé `dist`. Ils contiennent beaucoup de sous-répertoires dont voici les significations :

répertoire	rôle
`/`	contient les squelettes. Leur nom se termine par `.html` pour des raisons historiques mais cela ne préjuge pas de leur contenu. Il sufit de donner un tel nom, privé de cette extension, au paramètre `page` de l'URL d'un site sous SPIP pour déclencher l'utilisation de ce squelette. Ce répertoire contient également les feuilles de styles (d'extension `.css`) définissant l'habillage par défaut
`formulaires/`	contient la partie html des *balises dynamiques*, squelettes de formulaires dont le code PHP figure dans le répertoire `ecrire/balise`
`icones_barre/`	contient les images destinées à illustrer la barre typographique
`images/`	images de l'espace privé
`javascript/`	librairies javascript (jQuery, barre typographique, ...)
`modeles/`	squelettes appelables avec la balise #MODELE ou avec les raccourics `<article6|modele>`
`polices/`	polices utilisables pour les images typographiques
`vignettes/`	vignettes standards pour les types de documents pouvant être associés à un article

Rôles du répertoire `ecrire` **et ses sous-répertoires**

Le répertoire `ecrire` comporte plusieurs sous-répertoires composés de fichiers PHP définissant des fonctions et procédant éventuellement, mais rarement, à une initialisation lors de leur chargement (ces exceptions sont appelées à disparaître). Les fichiers situés au niveau du répertoire principal sont les plus importants à comprendre pour contribuer à SPIP, il s'agit de `inc_version.php`, `public.php` et `index.php`.

Le fichier `ecrire/inc_version.php` initialise les constantes et les variables globales nécessaires au fonctionnement de SPIP, notamment celles assurant sa portabilité sur les différentes plates-formes. Assez tôt lors de son chargement, il inclut le fichier `inc/utils.php`, où figurent les fonctions indispensables à SPIP, un fichier hors distribution nommé `mes_options.php` permettant de moduler cette initialisation sans avoir à modifier le fichier `inc_version.php`. En particulier, il est possible dans ce fichier personnel d'invoquer la fonction `spip_initialisation` pour définir les répertoires de données et disposer ainsi de plusieurs sites sous SPIP utilisant une seule distribution (l'appel standard de cette fonction, plus loin dans `inc_version.php`, sera automatiquement neutralisé). Une autre fonction indispensable à SPIP est `find_in_path`, exploitant le *chemin d'accès*, ainsi que `include_spip` qui repose sur `find_in_path`, et `charger_fonction` qui repose sur `include_spip`. Tous les fichiers de SPIP sont chargés par ces deux dernières fonctions.

Le fichier `ecrire/public.php`, appelé par `spip.php`, a pour rôle essentiel de délivrer les pages de l'espace public, commandées lorsque la requête HTTP comporte (après réécriture

éventuelle) le paramètre `page`. Ce script applique alors le squelette ayant pour nom la valeur de ce paramètre. Il envoie les en-têtes HTTP et le contenu obtenus, gère leurs éventuelles erreurs et lance les tâches de fond à l'aide de la fonction `cron`. Contribuer à l'espace public de SPIP consiste donc simplement à fournir de nouveaux squelettes, avec éventuellement leurs feuilles de style et leurs images.

L'autre rôle de `ecrire/public.php` concerne le cas où la requête HTTP comporte l'argument `action`. Il applique alors la fonction `charger_fonction` à la valeur *v* de ce paramètre `action`. Cette application a pour effet de charger le fichier homonyme du répertoire `action`, dont la fonction principale `action_`*v*`_dist` est alors invoquée. Ces scripts effectuent essentiellement des écritures (en base ou sur fichier) et ne retournent en général pas de résultat, échappant ainsi à la problématique de la mise en page.

Le fichier `index.php` est le fichier central d'accès aux formulaires de l'espace privé. Il authentifie l'internaute, initialise ses données personnelles et applique la fonction `charger_fonction` à la valeur *v* du paramètre `exec`. Cette application a pour effet de charger le fichier homonyme du répertoire `exec`, dont la fonction principale `exec_`*v*`_dist` est alors invoquée. Celle-ci a la charge de délivrer l'intégralité du flux de sortie, y compris les en-têtes HTTP. Il est donc possible d'étendre Spip simplement en rajoutant un fichier PHP dans un sous répertoire nommé `exec` d'un répertoire figurant dans `SPIP_PATH`.

Le répertoire `exec` contient exclusivement les fichiers définissant les fonctions directement invocables par le paramètre d'URL `exec`. Le code PHP de ces fichiers ne doit jamais accéder en écriture à la base de données (les exceptions à cette règle sont en voie de disparition). A l'inverse, il y accède abondamment en lecture afin de vérifier les droits du demandeur et déterminer les données à visualiser. Si l'on veut voir SPIP sous l'archétype *Modèle-Vue-Controleur*, les fichiers de `exec` remplissent le rôle de *Controleur*. Si l'on veut voir SPIP sous l'archétype *(Print(Eval(Read)))* de Lisp, c'est la partie *Read*. A terme, ce répertoire devrait devenir un répertoire de squelettes. Il est demandé aux nouvelles contributions de Spip de penser à cet objectif lors de leur rédactions.

Le répertoire `action`, dont il a déjà été question, contient essentiellement les scripts accédant en écriture à la base de données. Si l'on veut voir SPIP sous l'archétype *Modèle-Vue-Controleur*, les fichiers de `action` remplissent le rôle de *Modèle*. Si l'on veut voir SPIP sous l'archétype *(Print(Eval(Read)))* de Lisp, c'est la partie *Eval*. Là encore, contribuer à SPIP consiste à écrire de tels scripts et à les invoquer par des formulaires construits avec la fonction `generer_action_auteur` assurant la sécurisation de l'accès à ces scripts, qui à l'inverse invoqueront la fonction `securiser_action` pour vérifier les droits du demandeur. Cette architecture permet de calculer ces droits seulement à la construction des formulaires appelant les scripts d'accès en écriture : plutot que de recalculer tous les droits, ces scripts vérifieront simplement que la clé figurant dans les arguments est la même que celle qu'ils recalculent à partir des autres arguments, de l'identité du demandeur et d'une valeur aléatoire renouvellée périodiquement. Ces scripts ne retournent en général pas de résultat, il n'y a donc là aussi ni code HTML, ni appel à la fonction `echo` (les exceptions sont appelées à disparaître). En revanche, ils sont souvent appelés avec un paramètre HTTP nommé `redirect`, demandant une redirection qui sera alors automatiquement opérée par `public.php`, qui envoie un statut HTTP 204 en l'absence de ce paramètre. Dans le cas des formulaires construits avec la fonction `ajax_action_auteur`, cette redirection conduit au script homonyme dans le répertoire `exec`. Ce deuxième script se réduit le plus souvent à charger le fichier homonyme dans le répertoire `inc`, d'appeler sa fonction principale dont le résultat est retourné au client

par l'intermédiaire de la fonction `ajax_retour`. Il est ainsi très facile d'étendre SPIP en mode AJAX en utilisant cette infrastructure.

Le répertoire `inc`, le plus volumineux, contient essentiellement les fonctions construisant les pages de l'espace privé renvoyées au client, ces fonctions devant être à terme les filtres utilisés par les fichiers de `exec` quand ils seront des squelettes. Si l'on veut voir SPIP sous l'archétype *Modèle-Vue-Controleur*, les fichiers de `inc` remplissent le rôle de *Vue*. Si l'on veut voir SPIP sous l'archétype *(Print(Eval(Read)))* de Lisp, c'est la partie *Print*. Toutefois ce répertoire contient également beaucoup de fonctions relevant plutot du *Controle* et devra donc être réorganisé. La plupart des fichiers de `inc` sont chargés par l'intermédiaire de `charger_fonction`, et ce sera le cas de tous à terme. Aucune des fonctions de ce répertoire n'est censée utiliser `echo`. Les contributions à SPIP sont appelées à respecter ces règles dès à présent.

Le répertoire `install` contient exclusivement les fonctions nécessaires à l'installation de SPIP. Chaque étape peut-être surchargée ou complétée par d'autres, la fonction principale de `exec/install.php` utilisant ce répertoire selon le même principe que `ecrire/index.php` avec le répertoire `exec`.

Le répertoire `urls` contient des fichiers définissant chacun le même jeu de *fonctions de réécriture d'URL*. Il s'agit des fonctions calculant, à partir d'un index numérique dans une table de la base de données, un signet plus facile à lire et écrire que l'appel du script PHP effectivement opéré par le serveur HTTP pour cet index et cette table. Là encore, il suffit de rajouter un fichier dans ce répertoire pour obtenir un nouveau jeu, dont le nom sera présenté dans le panneau de configuration de l'espace privé gérant les types d'`urls` (avant SPIP 2.0, ce panneau n'existaient pas et ces fichiers étaient utilisés par la globale `type_urls`).

Le répertoire `lang` contient exclusivement des fichiers de données, tableaux indiquant la traduction, pour toutes les langues connues de SPIP, de tous les arguments que la fonction `_T`, définie dans `inc/utils.php`, est susceptible de recevoir. Les fichiers sont chargés exclusivement par les fonctions de `inc/lang.php`. Traduire les fichiers de référence `*fr*` en donnant un nom conventionnel aux fichiers obtenus suffit à déclarer une nouvelle langue à SPIP.

Le répertoire `charset` contient lui aussi exclusivement des fichiers de données, tableaux permettant de passer d'un codage de caractères à un autre (utf, iso, ascii, entités html etc). Ils sont lus exclusivement par les fonctions de `inc/charsets.php`. Il suffit là encore de rajouter un fichier pour disposer d'un nouveau codage, mais SPIP propose tous ceux couramment utilisés, aussi un telle intervention est rarissime.

Le répertoire `base` contient les fonctions d'interfaces entre PHP et les serveurs SQL que SPIP peut appeler. En particulier, le fichier générique `abstract_sql.php` contient les fonctions qu'il faut utiliser pour dialoguer avec les serveurs SQL, les fonctions de base de PHP pour cela ne devant pas être utilisées directement par souci de portabilité. Ne doit évidemment figurer aucun code MIME dans ce répertoire.

Le répertoire `req` contient les portages effectifs du serveur SQL virtuel de SPIP vers les serveurs réels (MySQL, PG) et assimilés (SQLite).

Le répertoire `balise` contient les fichiers PHP associés aux *balises dynamiques* de SPIP. Leur nom est homonyme du squelette de `squelettes-dist/formulaires`. Compléter l'espace public de SPIP par un formulaire *F* consiste à créer un fichier *F*.html dans son `SPIP_PATH` et un fichier *F*.php dans un sous-répertoire `balise` de son `SPIP_PATH`. Le rôle de ce fichier PHP est de réceptionner les saisies demandées par ce formulaire, et éventuellement de ré-afficher celui-ci pour complément en cas de saisies invalides. C'est sans doute le type de contribution à l'espace public la plus difficile à réaliser, car la mécanique sous-jacente exige deux passes d'exécution de PHP dont il faut bien comprendre les rôles respectifs. Avant que ce mécanisme n'existe, la stratégie de développement de formulaire consistait à écrire des squelettes comportant des instructions PHP. Il est toujours possible de le faire, mais le résultat sera peu efficace car jamais mis en cache ; et il ne dispense pas de comprendre les deux passes de PHP intrinsèques au phénomène.

Le répertoire `public` contient le compilateur des squelettes. C'est une partie du code assez compliquée, mais elle bénéficie d'une interface de programmation qui rend ce compilateur totalement extensible sans exiger d'en comprendre tous les détails. La description la moins incomplète est ici :

http://www.spip-contrib.net/spikini/PagePrincipale?wiki=NouveauCompilo.

Le répertoire `lib` contient des sous-répertoires de bibliothèques développées en dehors de SPIP mais dont il peut avoir besoin. Actuellement, seule est fournie systématiquement la bibliothèque `safehtml`, fournissant des utilitaires de sécurisation des pages scriptables. Pour contribuer à cette partie, remonter directement à son site (http://pixel-apes.com/safehtml).

Dernier point : la plupart des fichiers de SPIP sont utilisés via `charger_fonction` qui charge un fichier et appelle sa fonction homonyme censée y être définie. Il s'ensuit que le nom d'un fichier PHP doit être composé exclusivement de caractères acceptables pour un nom de fonction PHP : on évitera donc le signe moins, le point etc.

Règles de programmation

SPIP a démarré à l'époque où PHP transformait automatiquement en variables globales les paramètres HTTP. Ce style de programmation suicidaire a été abandonné par PHP4. SPIP a suivi une évolution parallèle, mais décalée dans le temps. Bien que le code actuel ne suive donc pas toujours les règles qui vont suivre, il est demandé aux contributions à venir de les respecter dès à présent, sans attendre que SPIP cesse d'y déroger ici ou là. On pourra lire attentivement le fichier `ecrire/articles.php`, le plus proche des spécifications qui suivent.

- Privilégier l'écriture par fonctions. La philosophie du logiciel libre est d'être utilisé dans un maximum de contextes différents, en conséquence le code doit être écrit dans une optique de réutilisation en dehors de son contexte initial de développement. L'écriture par fonctions ne référençant aucune variable globale et n'effectuant aucun appel à `echo` ou `print` garantit un chargement silencieux et un appel sans effets secondaires indésirables.
- Eviter au maximum l'utilisation de variables globales. Elles sont responsables de nombreuses failles de sécurité et d'impossibilités de réutilisation du code. Les alternatives à leur usage sont :
    - la constante, qui a l'avantage de signaler au lecteur que cette valeur ne changera pas pendant toute la durée du script ;

    - la variable statique, qui a l'avantage de signaler au lecteur qu'il s'agit de valeur à longue durée de vie mais n'intéressant que la fonction qui la déclare.
- Ecrire du code ne produisant aucune erreur ni avertissement en mode `error_reporting(E_ALL)`. Cela facilite la mise au point en cas de variable involontairement indéfinie. Si l'on a vraiment besoin d'exécuter du code en dehors de ce mode, utiliser l'artifice @ en le limitant au maximum à la portion de code problématique, et prévoir un message d'erreur dans le journal, en utilisant la fonction `spip_log`.
- Commenter le *contexte*, pas le texte. Il ne sert à rien de paraphraser le nom de ses variables et fonctions, ni les fonctions PHP décrites dans son manuel : les commentaires comme *boucle sur le tableau de valeurs* devant un `foreach` ne font que grossir inutilement les fichiers. En revanche, il est souhaitable d'indiquer le type des arguments (PHP étant un langage dynamiquement typé, il se devine difficilement) et leur propriété supposée à l'entrée de la fonction (par exemple : non nul). Quand un bug difficile est corrigé ou anticipé, dire pourquoi le code initial était incorrect pour éviter qu'une réécriture ultérieure ne le réintroduise en croyant optimiser. Enfin, SPIP étant développé en français, éviter les termes absents des dictionnaires de cette langue afin de faciliter la compréhension à ceux dont ce n'est pas la langue maternelle.
- Nommer rationnellement les fonctions et variables. L'organisation du code de SPIP est plutôt fondée sur le découpage en répertoires dédiés que sur des règles de nommage strictes, mais on évitera néanmoins les incohérences comme le multi-linguisme à l'intérieur d'un nom. Les fonctions d'un même fichier tenderont à avoir un préfixe ou un suffixe commun, inspiré du nom du fichier.
- Tester sur un maximum de configurations. N'oubliez pas que SPIP doit fonctionner sur toute plate-forrme, tant côté client que côté serveur. Il y a nécessairement plusieurs navigateurs sur votre machine, tester votre code sur au moins deux d'entre eux. Dans la mesure du possible, essayez également plusieurs hébergeurs. Lorsque une plate-forme force une écriture étrange, la mentionner explicitement, en précisant sa version et la date de l'essai.

Règles de présentation et d'écriture

SPIP est soucieux de la qualité typographique des pages qu'il envoie aux clients HTTP, il est donc lui-même soucieux de celle de ses propres fichiers. Il est recommandé de respecter les règles qui suivent, assez standard en programmation.

Présentation

- Le texte des fonctions doit être court, idéalement inférieur à la taille d'un écran standard, afin que l'intégralité de leur structure soit visible d'un seul coup d'œil.
- Le texte des fonctions doit être indenté, chaque bloc contenu à l'intérieur d'une paire d'accolades devant être renfoncé d'une unité d'indentation supplémentaire, cela s'appliquant évidemment récursivement.
- Prendre le caractère de tabulation pour indenter, car il permet à chacune de choisir librement la profondeur d'indentation dans les options de son éditeur de textes.
- Lorsqu'une instruction PHP occupe plusieurs lignes, mettre une ligne blanche avant et après ;
- Les fragments HTML, qu'il s'agisse de transitions (`<?php` et `?>`) ou d'expressions PHP seront des blocs XHTML complets : éviter de mettre une balise ouvrante au

début d’une fonction et sa fermante à la fin, cela ne permet pas de déplacer facilement cette paire en cas de besoin.

Typographie

- Lors de l’utilisation de parenthèses ou de crochets, éviter les espaces après l’ouvrant et avant le fermant.
- Dans les expressions PHP laisser un espace de part et d’autre des opérateurs binaires (+, =, *, AND, ...).
- Les opérateurs unaires (!, ...) doivent être collés au paramètre auquel ils s’appliquent.
- Par convention, lors d’un appel de fonction, il n’y a pas d’espace devant la parenthèse ouvrante : « f($x) » et non « f ($x) ». A contrario, et pour bien distinguer, on laisse un espace devant la parenthèse quand il s’agit d’une structure de contrôle intégrée au langage : « if (!$x) » et non « if(!$x) ».
- Les virgules et points-virgules sont suivis mais non précédés d’un espace.

Pourquoi réaliser un plugin ?

Août 2006 — maj : Novembre 2009

Adapter SPIP à ses propres besoins

SPIP met en place tout un formalisme pour permettre une modification et une extension harmonieuse du code. La principale avancée se situe dans la possibilité de "greffer" (et de retirer la greffe quand on le souhaite sans avoir touché le code de notre SPIP tout neuf) des améliorations et des modifications aussi diverses que variées. Les possibilités sont tellement vastes que toute la question est maintenant de savoir comment faciliter l'installation, tant pour soi-même que pour une distribution publique, de toutes ces personnalisations.

Techniquement, l'introduction des plugins a permis d'ouvrir les possibilités suivantes :
- tous les fichiers du noyau sont surchargeables [1] et toutes les fonctions appelables systématiquement [2],
- une interface d'application (API) est maintenue par la définition d'un certain nombre de points d'entrée dans le code.

On a besoin de réaliser un plugin dans quatre cas :

- **Fonctions et options** : réaliser son premier plugin, migrer et rendre portables ses fonctions et ses options pour soi ou pour les autres.

- **Les points d'entrée** : injecter, le temps d'activation d'un plugin, du code au coeur de SPIP et modifier profondément son fonctionnement.

- **Modifier les fichiers natifs** : en l'absence de point d'entrée, modifier des parties du code de SPIP sans intervenir concrètement sur le noyau.

- **Réecrire son propre code** : inventer son propre script que l'on greffera sur SPIP.

Si vous n'avez aucun de ces objectifs en tête, ne vous bilez pas avec les pages suivantes :) Cependant, vous serez probablement intéressé par ce que peuvent vous fournir des plugins distribuées librement.

Notes

[1] Surcharger un fichier consiste à multiplier la lecture d'un fichier afin de lui faire prendre la dernière valeur lue. De façon très schématique, si vous surcharger un fichier qui a pour valeur "Bonjour" avec un fichier qui a pour valeur "Bonsoir", l'écran affichera "Bonsoir" et non "Bonjour".

[2] Vous pouvez appeler à partir de n'importe quel fichier une fonction déjà existante, pour autant que vous ayez inclus en début de fichier le fichier qui contient la fonction que vous voulez utiliser. Ceci est particulièrement puissant, car les concepteurs de SPIP ont réalisé beaucoup de fonctions qui sont autant de routines que vous n'aurez pas à reprogrammer. On peut citer en exemple *lire_fichier* (qui lit un fichier), *ecrire_fichier* (qui écrit un fichier), *preg_files* (qui cherche un fichier), toutes les requetes sql et des dizaines de fonctions qui sont simplifiées au possible grâce à des routines contenues dans le code source de SPIP. Voir la doc du code pour en savoir plus.

On trouvera une liste conséquente de plugins disponibles (dont il sera possible d'étudier le code)

- soit sur **spip-contrib** : leur description (http://www.spip-contrib.net/Plugins,112) ; la zone de téléchargement (http://www.spip-contrib.net/spip.php?page=paquets),
- soit sur **plugins-spip** (http://plugins.spip.net/).

Réaliser un premier plugin

Août 2006 — maj : août 2010

On crée un plugin au même endroit qu'on installe ceux qu'on peut récupérer et déjà prêts à fonctionner (Voir « Installer un plugin »).

Le système de plugin s'appuie sur la recherche de chemins connus dans une liste bien précise. Les technophiles appelleront ça `SPIP_PATH` (cf. Où placer les fichiers de squelettes ?). L'activation, ou non, d'un plugin décide de l'ajout de son répertoire associé dans cette liste. On devra donc, pour réaliser son premier plugin, créer :

- Un répertoire `plugins/` à la racine du site. La conséquence directe de cette action est d'activer l'interface de gestion des plugins. Le premier symptôme étant l'apparition du bouton « Gestion des plugins » dans le menu « Configuration » de l'interface privée (en mode interface complète uniquement)

- Un sous-répertoire `mon_premier_plugin/` pour le plugin à réaliser.

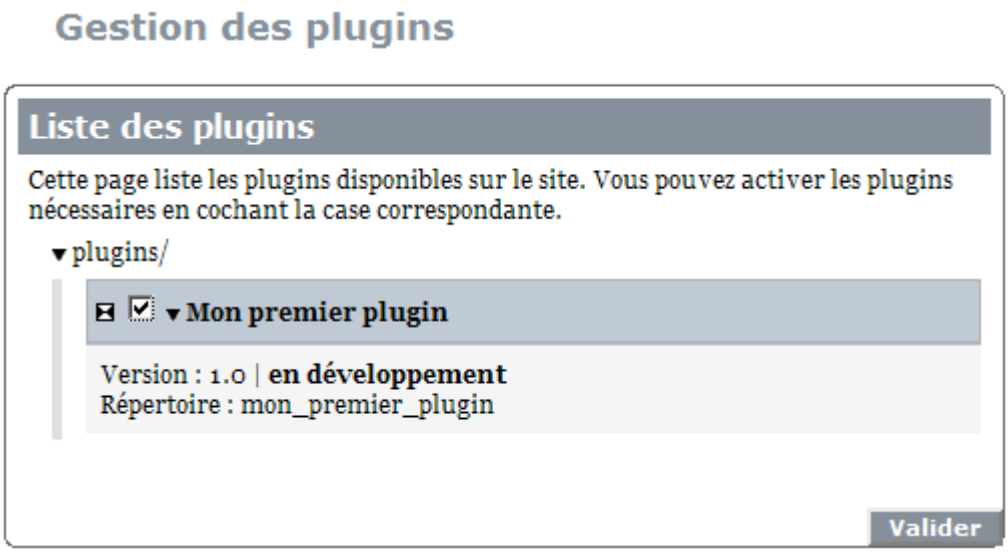
Pour indiquer à SPIP ce qu'est censé faire le plugin, celui-ci est décrit par un fichier, qui porte le nom `plugin.xml`, qu'on créé dans le répertoire même du plugin. Il s'agit d'un fichier à la syntaxe stricte mais relativement simple que voici :

```
<plugin>
 <nom>Mon premier plugin</nom>
 <version>1.0</version>
 <prefix>demo</prefix>
</plugin>
```

Il s'agit d'un fichier contenant les seules balises obligatoires.

- La balise `nom` peut être internationalisée grâce à la balise multi (exemple : `<nom><multi>My first plugin[fr]Mon premier plugin</multi></nom>`). C'est le titre du plugin.
- La balise `version` est purement informative. Il n'y a pas de règle précise pour gérer les numéros de version. Il appartient au développeur de les gérer lui-même. Dans l'avenir, il est toutefois possible que cette valeur serve à gérer d'éventuelles dépendances et un minimum de cohérence entre SPIP et les plugins...
- La balise `prefix` est cruciale. Il est tout d'abord important de s'assurer de l'unicité de cette valeur parmi tous les plugins que vous installez. Il définit ensuite le préfixe des fonctions que vous allez programmer en php et qui seront les éléments moteurs du plugin. Ainsi, si plusieurs plugins cohabitent avec des fonctions aux traitements similaires, il sera impossible qu'elles provoquent des erreurs à cause de leurs noms. C'est ce qu'on appelle aussi un « espace de noms ».

Bien que ne produisant aucun effet pour le moment, ce minimum permet déjà d'activer le plugin dans l'interface :



Pour ce faire, cochez la case en face du nom du plugin et cliquez sur le bouton « valider » en bas. La zone du titre est grisée.

Amusez-vous à retirer l'une ou l'autre de ses balises pour constater que SPIP peut vous indiquer ce qui manque ou tout simplement ne pas fonctionner.

Il est possible de placer d'autres éléments dans un fichier plugin.xml :

- La balise `<etat>` : 4 valeurs possibles. En l'absence de cette balise, ou si elle est présente mais vide, le plugin est considéré comme "en développement". Le fonctionnement du plugin n'est aucunement influencé par la valeur de cette balise. À noter qu'une puce colorée permet de reconnaître l'état d'un coup d'œil.

Etat	Puce	Signification
dev	noire	En cours de développement. Nombreux bugs possibles. Ne fonctionne peut-être pas.
test	orange	La fonction principale est opérationnelle et est à essayer. Les développeurs comptent sur nous pour remonter des commentaires sur d'éventuelles erreurs ou des remarques sur l'ergonomie d'une interface graphique, par exemple.
stable	verte	Complètement opérationnel. Sans bug, en théorie.
experimental	rouge	C'est l'aventure !

- Les balises `<auteur>` et `<description>` sont facultatives et fonctionnent comme les textes d'un article (tous raccourcis autorisés) et servent à fournir des informations succintes sur le plugin. Ainsi le code :

```
<plugin>

 <nom>Mon premier plugin</nom>
 <version>1.0</version>
 <prefix>demo</prefix>
 <etat>dev</etat>
 <auteur>Jean Dupont [contact->mailto:jdupont@monsite.net]
 _ [mon site->http://www.monsite.net]</auteur>
 <description>Ce plugin est une démonstration. Il est
distribué sous licence GNU/GPL</description>
</plugin>
```

aura pour conséquence d'afficher :

Notez qu'il est nécessaire de coder les caractères accentués avec des entités HTML (`&eacute;` pour é, par exemple).

Si vous placez au même niveau que `plugin.xml` un fichier `squelette.html`, votre plugin une fois activé, permet l'utilisation du dit squelette qui sera utilisable en l'appelant dans votre navigateur avec la notation « usuelle » : `votresite.net/spip.php?page=squelette`. Il s'agit là d'un simpliste mais premier exemple de surcharge de fichier. Ce principe sera approfondi dans un autre tutoriel.

Pour illustrer cette démonstration, nous allons réaliser un plugin dont le but est de colorer d'une manière particulière, chaque occurrence du mot « spip » dans un texte. Nous allons pour cela, placer dans le répertoire du plugin les 2 fichiers utiles et informer SPIP de leurs noms via quelques balises supplémentaires dans le fichier XML :

```
<plugin>

 <nom>Mon dernier plugin</nom>
 <version>1.0</version>
 <prefix>demo</prefix>
 <etat>experimental</etat>
 <auteur>Jean Dupont [contact->mailto:jdupont@monsite.net]
 _ [mon site->http://www.monsite.net]</auteur>
 <description>Ce plugin est une démonstration. Il est
distribué sous licence GNU/GPL</description>
 <fonctions>exemple_fonctions.php</fonctions>
 <options>exemple_options.php</options>
</plugin>
```

Voici la description des 2 balises ajoutées :

- `<fonctions>` : contient le nom d'un fichier qui sera chargé à chaque recalcul. C'est l'équivalent pour chaque plugin, du fichier `mes_fonctions.php`. Il n'est donc utilisé que pour le site public, puisque qu'il n'est appelé qu'en cas de calcul du cache. On y mettra typiquement des filtres ou des définitions de balises, de critères.
- `<options>` : contient le nom d'un fichier qui sera chargé à chaque appel de page . C'est l'équivalent pour chaque plugin, du fichier `mes_options.php`. Il est aussi utilisé dans l'interface privée à chaque appel de page.

Créez les 2 fichiers avec le contenu ci-dessous :

`exemple_fonctions.php` :

```
<?php

function colore_spip($texte) {
 global $couleur;
 return preg_replace('/([^(class=")])(spip)/i',
  '$1<span style="color: '.$couleur.';">$2</span>',
  $texte);
}

?>
```

`exemple_options.php` :

```
<?php

$couleur = '#ff017d';

?>
```

Attention ! Il y a des noms de fichier qu'il est préférable ne pas utiliser dans le cas des plugins : nous sommes dans le cadre d'une liste de répertoires connus et SPIP s'arrête toujours au premier fichier trouvé. Si le fichier de la balise `<fonctions>` s'appelle

`mes_fonctions.php`, il y aura confusion avec le fichier de votre dossier squelettes, s'il existe et surtout, si plusieurs plugins sont programmés avec ce nom de fichier, le système risque de se perdre... idem pour `mes_options.php`, qu'il faut réserver au noyau de SPIP ainsi que pour les fichiers de langues `local_xx.php` auxquels on préfèrera une autre méthode décrite dans un tutoriel consacré à la surcharge de code.

Donc : **évitez les noms « communs » tels que `mes_fonctions.php` et `mes_options.php`**.

Nous verrons dans un autre article comment surcharger les répertoires de SPIP. Sachez déjà que des noms de répertoires tels que « modeles », « formulaires », « inc », « action » etc... sont à utiliser à bon escient.

Enfin, prenez garde à être très exact dans la saisie des noms de fichier dans `plugin.xml`, l'interface de gestion est très capricieuse pour l'ajout des dernières balises (messages d'erreur qui plantent le serveur...)

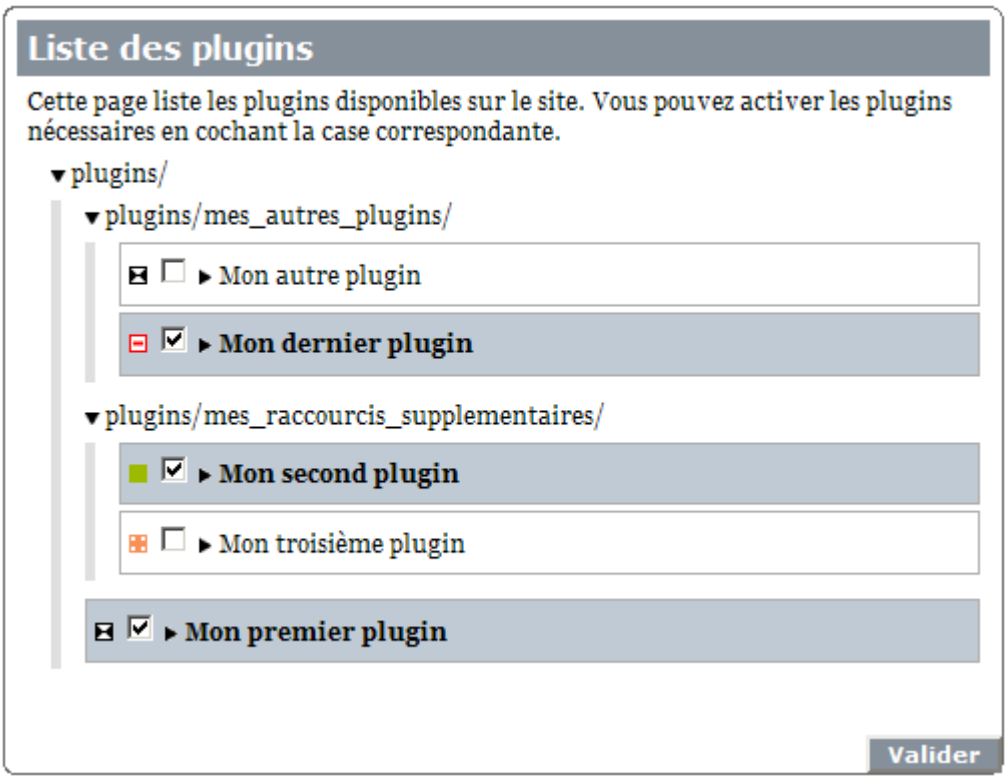
Bref, voilà, ce plugin apporte un nouveau filtre pour vos squelettes. Essayez `[(#TEXTE|colore_spip)]` dans article.html par exemple.

On installe tous les plugins dans le répertoire `plugins/`, mais on peut créer des sous-répertoires pour catégoriser les plugins qu'on installe ou développe.

Ainsi, pour l'arborescence suivante :

```
plugins/
  |
  |-- mon_premier_plugin/
  |-- mes_raccourcis_supplementaires/
         |
         |-- raccourcis_2/
         |-- raccourcis_3/
  |-- mes_autres_plugins/
         |
         |-- raccourcis_4/
         |-- raccourcis_5/
```

l’interface de gestion aura l’allure suivante :



Aller plus loin

Une fois lancé sur cette voie, des interrogations techniques, liées à des problématiques de programmation en PHP, par exemple, se posent, c’est légitime.

Le site de documentation technique est conçu pour répondre à ces questions. Concernant les plugins, vous pouvez poursuivre à cette adresse (http://doc.spip.org/@Tuto-Se-servir-des-points-d-entree) pour une première approche sur les points d’entrée de SPIP (ou « pipeline »).