Documentation SPIP

Manuel de référence des boucles et balises

Manuel de référence des boucles et balises

Comment créer les pages d'un site géré sous SPIP, grâce au langage des boucles, balises et filtres de SPIP.

Des boucles et des balises

Juin 2001 — maj : Novembre 2003

Tout ce qui suit concerne désormais le langage de description de la mise en page des *squelettes* dans SPIP ; si vous avez bien compris l'explication précédente, vous savez que nous travaillons donc dans les fichiers « .html ».

La présente documentation est volontairement technique (il s'agit ici de références techniques) ; vous pouvez préférer commencer par notre guide **Pas à pas**, plus didactique, et revenir ensuite ici pour une documentation plus précise.

Des boucles

La notion de base du langage de SPIP est la *boucle*.

- La logique de la boucle

Une base de données, classiquement, c'est une liste d'éléments : ici, une liste des articles, une liste des rubriques, une liste des auteurs, etc. Pour « fabriquer » le site, on va donc extraire de cette liste certains de ses éléments :
- à la base, on veut extraire *un seul élément* d'une liste ; par exemple, afficher l'article désiré ;
- mais il est fréquent d'extraire *plusieurs éléments* d'une liste ; par exemple, dans la page d'une rubrique, on veut afficher *tous* les articles contenus dans cette rubrique, ainsi que *toutes* les sous-rubriques contenues dans cette rubrique ;
- plus subtil : il arrive fréquemment qu'il n'y ait pas d'éléments satisfaisants à tel ou tel endroit ; SPIP doit alors pouvoir gérer l'éventualité de l'absence de ces éléments ; par exemple, le squelette de l'affichage des rubriques demande l'affichage de toutes les sous-rubriques contenues dans une rubrique ; que faire, alors, s'il n'y a pas sous-rubriques dans cette rubrique spécifique ?

Ces trois situations sont traitées par la notion unique de *boucle*, qui permet à la fois de gérer l'affichage d'un seul élément, de plusieurs éléments successifs, ou l'absence d'éléments.

Le système de boucle permet, dans un code unique :
- d'indiquer à quel endroit du code HTML on a besoin de quel type d'élément (à tel endroit on veut récupérer la liste des articles, à tel endroit on veut inclure la liste des sous-rubriques...) ;
- de prévoir l'affichage d'un élément unique ;
- d'indiquer comment est affichée une liste de plusieurs éléments ;
- de déterminer ce qu'on affiche lorsqu'il n'y a aucun élément correspondant.

Analogie avec la programmation en PHP/mySQL

Ceux qui ont déjà programmé des requêtes mySQL en PHP savent que le traitement se déroule en deux temps :
- la construction de la syntaxe de la requête (qui consiste à dire « je veux récupérer la liste des articles contenus dans telle rubrique... ») ;
- l'analyse et l'affichage des résultats au travers d'une boucle.

Ce sont ces deux événements qui sont gérés, dans SPIP, au travers des boucles.

Les balises SPIP

Grâce aux boucles, on a donc récupéré des éléments uniques ou des listes d'éléments : par exemple une liste d'articles ou une liste de rubriques...

Cependant, chaque élément de telles listes est composé de plusieurs éléments précis : par exemple un article se compose d'un titre, d'un surtitre, d'un sous-titre, d'un texte d'introduction (chapeau), d'un texte principal, d'un post-scriptum, etc. Il existe ainsi des balises spécifiques à SPIP, permettant d'indiquer précisément à quel endroit on affiche des éléments : « placer le titre ici », « placer le texte ici »...

Les balises à l'intérieur des boucles

Voici, au travers d'un cas classique, le principe de fonctionnement général d'une boucle accompagnée de ses balises (attention, ça n'est pas du langage SPIP, c'est une description logique) :

BOUCLE : afficher la liste des articles de cette rubrique

- afficher ici le titre de l'article
- afficher le sous-titre
- afficher le texte

Fin de la BOUCLE

Cette boucle, analysée par SPIP, peut donner trois résultats différents.

- **Il n'y a aucun article dans cette rubrique.**

Dans ce cas, bien évidemment, aucun des éléments « afficher ici... (titre, sous-titre...) » n'est utilisé. En revanche, si on l'a prévu, on peut afficher un message du genre « Il n'y a pas d'article ».

- **Il y a un seul article dans cette rubrique.**

Dans ce cas, très simplement, la page HTML est construite sur le modèle de la boucle :

- Titre de l'article
- Sous-titre
- Texte de l'article

- **Il y a plusieurs articles dans cette rubrique.**

La description de la mise en page (« placer ici... ») va alors être calculée successivement pour chacun des articles. Ce qui donne simplement :

- Titre de l'article 1
- Sous-titre de l'article 1
- Texte de l'article 1

- Titre de l'article 2

- Sous-titre de l'article 2
- Texte de l'article 2

  ...

- Titre du dernier article
- Sous-titre du dernier article
- Texte du dernier article

La suite de ce guide de référence se construira donc de la manière suivante :
- syntaxe générale des boucles ;
- syntaxe générale des balises de SPIP ;
- et, ensuite, une page spécifique à chaque type de boucles, indiquant quelles balises on peut y utiliser.

La syntaxe des boucles

Mai 2001 — maj : Novembre 2009

Syntaxe de base

La syntaxe simplifiée d'une boucle est la suivante :

```
<BOUCLEn(TYPE){critère1}{critère2}...{critèrex}>
* Code HTML + balises SPIP
</BOUCLEn>
```

On a vu, dans l'explication sur les boucles et les balises, que le « `Code HTML + balises SPIP` » se répétait autant de fois que la boucle obtenait d'éléments tirés de la base de données (c'est-à-dire une fois, plusieurs fois, ou zéro fois).

La ligne importante, ici, est :

```
<BOUCLEn(TYPE){critère1}{critère2}...{critèrex}>
```

- **L'élément `BOUCLE`** est l'ordre indiquant qu'il s'agit d'une boucle SPIP ; on ne peut donc pas le modifier ; dit autrement, toutes les boucles de SPIP commencent par l'instruction `BOUCLE`.

- **L'élément *n*** est le nom ou le numéro de la boucle, librement choisi par le webmestre pour chaque boucle qu'il utilise (attention : on prendra soin de nommer ses boucles avec uniquement **des caractères alphanumériques non accentués** et le **tiret-bas** « *underscore* » ; c'est-à-dire des caractères de la *classe* **[a-zA-Z0-9_]**. On verra plus loin qu'il est possible (c'est même tout l'intérêt de la manœuvre) d'utiliser plusieurs boucles dans un même squelette : les nommer est donc indispensable pour les identifier.

Si vous décidez de numéroter vos boucles, la syntaxe devient par exemple (pour la boucle 5) :
```
<BOUCLE5...> ... </BOUCLE5>
```

Si vous décidez de donner un nom à vos boucles (c'est généralement plus pratique, votre code est plus lisible), il faut impérativement faire précéder ce nom par le symbole « _ » (que l'on appelle habituellement *underscore*). Par exemple :
```
<BOUCLE_sousrubriques...> ... </BOUCLE_sousrubriques>
```

- **L'élément `(TYPE)`** est primordial : il indique quel type d'éléments on veut récupérer. La syntaxe est importante : le TYPE est indiqué entre parenthèses (sans espaces), en majuscules, et ce TYPE doit correspondre obligatoirement à l'un des types prévus dans SPIP (qu'on trouvera dans la présente documentation) : ARTICLES, RUBRIQUES, AUTEURS, BREVES, etc.

Pour l'exemple précédent, on aurait donc :

```
<BOUCLE_sousrubriques(RUBRIQUES)...>
...
</BOUCLE_sousrubriques>
```

- **Les critères `{critère1}{critère2}...{critèrex}`** indiquent à la fois selon quels critères on veut sélectionner les éléments de la base de données (afficher les sous-rubriques incluses dans cette rubrique, afficher les autres rubriques installées au même niveau hiérarchique que la présente rubrique...), et la façon dont on va classer ou sélectionner les éléments (classer les articles selon leur date, selon leur titre... afficher uniquement les 3 premiers articles, afficher la moitié des articles...). Comme on peut combiner les critères, on peut très aisément fabriquer des requêtes très puissantes, du genre « afficher la liste des 5 articles les plus récents écrits par cet auteur ».

Les critères sont entre acolades ; ils peuvent être séparés les uns des autres par un espace. Exemple :

```
<BOUCLE_meme_auteur(ARTICLES) {id_auteur} {par date}{inverse} {0,5}>
...
</BOUCLE_meme_auteur>
```

Les différents critères et leur syntaxe seront explicités par la suite, pour chaque type de boucle (certains critères fonctionnent pour tous les types de boucles, d'autres sont spécifiques à certaines boucles).

Syntaxe complète

Le syntaxe indiquée précédemment peut être complétée par des éléments conditionnels.
En effet, la boucle précédente affiche successivement les éléments contenus *à l'intérieur* de la boucle. SPIP permet de plus d'indiquer ce qu'on affiche avant et après la boucle au cas où elle contient un ou plusieurs résultats, et ce qu'on affiche s'il n'y a aucun élément.

Cela donne :

```
<Bn>
* Code HTML optionnel avant
<BOUCLEn(TYPE){critère1}{critère2}...{critèrex}>
* Code HTML + balises SPIP
</BOUCLEn>
* Code HTML optionnel après
</Bn>
* Code HTML alternatif
<//Bn>
```

Le *code optionnel avant* (précédé de `<Bn>`) n'est affiché que si la boucle contient au moins une réponse. Il est affiché avant les résultats de la boucle.

Le *code optionnel après* (terminé par `</Bn>`) n'est affiché que si la boucle contient au moins une réponse. Il est affiché après les résultats de la boucle.

Le *code alternatif* (terminé par `<//Bn>`) est affiché à la place de la boucle (et donc également à la place des codes optionnels avant et après) si la boucle n'affiche rien (soit parce que la base de données n'a fourni aucune réponse, soit parce que le code utilisant ces réponses dans la boucle n'affiche rien).

Par exemple, le code :

```
<B1>
 Cette rubrique contient les éléments suivants:
 <ul>
   <BOUCLE1(ARTICLES){id_rubrique}>
   <li>#TITRE</li>
   </BOUCLE1>
 </ul>
</B1>
 Cette rubrique ne contient pas d'article.
<//B1>
```

donne les résultats suivants :

- **s'il y a un seul article :**

```
Cette rubrique contient les éléments suivants:
<ul>
  <li>Titre de l'article</li>
</ul>
```

- **s'il y a plusieurs articles :**

```
Cette rubrique contient les éléments suivants:
<ul>
  <li>Titre de l'article 1</li>
  <li>Titre de l'article 2</li>
  ...
  <li>Titre du dernier article</li>
</ul>
```

- **s'il n'y a aucun article :**

```
Cette rubrique ne contient pas d'article.
```

Des critères d'environnement en cascade

Chaque boucle effectue la sélection des éléments tirés de la base de données en fonction de critères. Certains de ces critères correspondent à l'environnement dans lequel se trouve la boucle.

Par exemple : si on prévoit une boucle du genre « Afficher les articles inclus dans cette rubrique », il faut savoir de quelle rubrique il s'agit. C'est ce que l'on nomme l'environnement.

- **L'environnement fourni par l'URL**

Lorsque l'on visite une page d'un site SPIP, son adresse contient généralement une variable. Par exemple : `spip.php?rubrique15`

Cette variable définit donc un premier environnement : la boucle « Afficher les articles inclus dans cette rubrique » doit alors être comprise comme « Afficher les articles de la rubrique 15 ».

Clairement, avec le même code de squelette, si on appelle l'adresse : `spip.php?rubrique7`, l'interprétation de cette boucle deviendra « Afficher les articles de la rubrique 7 ».

- **L'environnement fourni par les autres boucles**

À l'intérieur d'une boucle, l'environnement est modifié par chaque élément de la boucle. En plaçant des boucles les unes à l'intérieur des autres, on hérite ainsi d'environnements imbriqués les uns dans les autres.

Ainsi, dans la structure suivante :

```
<BOUCLE_articles: afficher les articles de cette rubrique>
 Afficher le titre de l'article
 <BOUCLE_auteurs: afficher les auteurs de cet article>
   Nom de l'auteur
 </BOUCLE_auteurs>
</BOUCLE_articles>
```

On doit comprendre que :
- la première boucle (`BOUCLE_articles`) affiche les articles en fonction de la rubrique, selon l'environnement fournit par l'URL (`id_rubrique=15` par exemple) ;
- dans cette boucle, on obtient un ou plusieurs articles ;
- « à l'intérieur » de chacun de ces articles, on a un environnement différent (celui de l'article, c'est-à-dire, par exemple, `id_article=199`) ;
- la seconde boucle (`BOUCLE_auteurs`), qui est installée *à l'intérieur* de la première boucle, dépend pour chacune de ses exécutions successives (elle est exécutée pour chaque article de la première boucle) : « afficher les auteurs de cet article » devient successivement « afficher les auteurs du premier article », « du deuxième article » et ainsi de suite.

On voit que, par l'imbrication de boucles successives, on obtient différentes boucles, incluses les unes dans les autres, qui dépendent du résultat des boucles dans lesquelles elles sont situées. Et finalement, la toute première boucle (celle qui contient toutes les autres) dépend d'un paramètre fixé dans l'adresse de la page.

Boucles incluses et boucles successives

Si l'on peut inclure des boucles les unes à l'intérieur des autres (chaque boucle incluse dépendant alors du résultat de la boucle à l'intérieur de laquelle elle est installée), on peut tout aussi bien installer des boucles les unes à la suite des autres ; des boucles successives n'influent pas les unes sur les autres.

Par exemple, la page d'une rubrique est typiquement constituée des éléments suivants :

```
<BOUCLE_rubrique(RUBRIQUES){id_rubrique}>
 <ul>Titre de la rubrique
 <BOUCLE_articles(ARTICLES){id_rubrique}>
   <li> Titre de l'article</li>
 </BOUCLE_articles>
 <BOUCLE_sous_rubriques(RUBRIQUES){id_rubrique}>
   <li> Titre de la sous-rubrique </li>
 </BOUCLE_sous_rubriques>
 </ul>
</BOUCLE_rubrique>
 <ul>Il n'y a pas de rubrique à cette adresse.</ul>
<//B_rubrique>
```

La première boucle (`BOUCLE_rubrique`) dépend de la variable passée dans l'URL de la page (`id_rubrique=15` par exemple).

Les boucles suivantes (`BOUCLE_articles` et `BOUCLE_sous_rubriques`) sont installées *à l'intérieur* de la première boucle. Ainsi, s'il n'existe pas de rubrique 15, la première boucle ne donne aucun résultat (le code alternatif « Il n'y a pas de rubrique... » est affiché), et donc les deux boucles incluses sont totalement ignorées. Mais s'il existe une rubrique 15, ces deux sous-boucles seront analysées.

On constate également que ces deux boucles se présentent *l'une après l'autre*. Ainsi, elles fonctionnent en fonction de la première boucle, mais indépendamment l'une de l'autre. S'il n'y a pas d'articles dans la rubrique 15 (`BOUCLE_articles`), on affichera tout de même la liste des sous-rubriques de la rubrique 15 (`BOUCLE_sous_rubriques`) ; et inversement.

Compteurs

Deux balises permettent de compter les résultats dans les boucles.

- **#TOTAL_BOUCLE** retourne le nombre total de résultats affichés par la boucle. On peut l'utiliser dans la boucle, dans ses parties *optionnelles* — avant et après — ou même dans la partie *alternative* après la boucle.
Par exemple, pour afficher le nombre de documents associés à un article :

```
<BOUCLE_art(ARTICLES){id_article}>
      <BOUCLE_doc(DOCUMENTS) {id_article}>
      </BOUCLE_doc>
      [il y a (#TOTAL_BOUCLE) document(s).]
      <//B_doc>
</BOUCLE_art>
```

Attention : si la partie centrale de la boucle ne retourne rien (c'est le cas avec la boucle `<BOUCLE_doc>` ci-dessus, qui ne sert qu'à compter le nombre de résultats), le `#TOTAL_BOUCLE` ne pourra être affiché que dans la partie alternative *après* de la boucle (`<//B_doc>`).

- **#COMPTEUR_BOUCLE** retourne le numéro de l'itération actuelle de la boucle. On peut par exemple l'utiliser pour numéroter des résultats :

```
<BOUCLE_art(ARTICLES) {par date} {inverse} {0,10}>
      #COMPTEUR_BOUCLE - #TITRE<br>
</BOUCLE_art>
```

La syntaxe des balises SPIP

Mai 2001 — maj : octobre 2010

Chaque type de boucle permet de sélectionner des éléments de la base de données de SPIP : des articles, des rubriques, des brèves, etc. Chacun de ces éléments est lui-même constitué d'éléments précis : un titre, une date, un texte, etc. À l'intérieur d'une boucle, il faut donc pouvoir indiquer à quel endroit du code HTML on place tel ou tel de ces éléments précis.

Pour cela, on va utiliser des balises SPIP.

Fonctionnement simplifié

Une balise SPIP se place à l'intérieur d'une boucle (puisqu'il faut savoir si l'on veut récupérer un élément d'un article, d'une rubrique, etc.). Le nom de ces balises est généralement simple, et nous fournirons, pour chaque type de boucle, la liste complète des balises que l'on peut utiliser.

Une balise est toujours précédée du signe dièse (**#**).

Par exemple, affichons une liste de noms d'articles :

```
<BOUCLE_articles(ARTICLES){id_rubrique}>
      #TITRE<br />
</BOUCLE_articles>
```

Lorsque la boucle sera exécutée, la balise SPIP **#TITRE** sera à chaque fois remplacée par le titre de l'article en question :

```
Titre de l'article 1<br />
Titre de l'article 2<br />
...
Titre du dernier article<br />
```

Rien de bien compliqué : on se contente d'indiquer à l'intérieur du code HTML le nom de l'élément désiré, et celui-ci est remplacé par le contenu tiré de la base de données.

Codes optionnels

Dans la pratique, un élément de contenu est souvent accompagné de code HTML *qui ne doit s'afficher que si cet élément existe,* faute de quoi la mise en page devient imprécise.

Par exemple : il existe une balise SPIP pour indiquer le surtitre d'un article. Or de nombreux articles n'ont pas de surtitre.

Complétons l'exemple précédent :

```
<BOUCLE_articles(ARTICLES){id_rubrique}>
      #SURTITRE<br />
      #TITRE<br />
</BOUCLE_articles>
```

qui, classiquement, nous donne une liste d'articles, avec désormais l'indication du titre et du surtitre de chaque article. Mais que se passe-t-il si l'article n'a pas de surtitre ? On obtient le code : « `<br />` », c'est-à-dire une ligne blanche (un retour chariot).

Ce que nous devons faire : n'afficher le code « `<br />` » que si un surtitre existe pour l'article.

La syntaxe de la balise SPIP devient alors :

```
[ texte optionnel avant (#BALISE) texte optionnel après ]
```

La balise qui détermine l'option est placée entre parenthèses, et l'ensemble du texte conditionnel entre crochets. Le *texte optionnel avant* et le *texte optionnel après* ne s'affichent que s'il existe, dans la base de données, un élément correspondant à cette balise.

Notre exemple devient :

```
<BOUCLE_articles(ARTICLES){id_rubrique}>
      [(#SURTITRE)<br />]
      #TITRE<br />
</BOUCLE_articles>
```

On obtient alors le résultat recherché : s'il existe un surtitre pour cet article, il est affiché et suivi du `<br />` ; s'il n'existe pas de surtitre, même le `<br />` est occulté.

- *attention : cet affichage conditionnel ne fonctionne pas avec les balises dynamiques (par exemple #URL_LOGOUT).*

Utilisations avancées

On peut imbriquer des balises étendues les unes dans les autres. Ainsi, si dans notre exemple on voulait n'afficher le logo de l'article que si le surtitre est défini, on pourrait écrire :

```
<BOUCLE_articles(ARTICLES){id_rubrique}>
      [[(#LOGO_ARTICLE)<br />](#SURTITRE)<br />]
</BOUCLE_articles>
```

Note : On ne peut jamais mettre une boucle dans le code optionnel d'une balise. Mais si on veut faire n'afficher une boucle qu'en fonction d'une certaine balise, on peut utiliser `<INCLURE{...}>` à l'intérieur d'un code optionnel.

Balises non ambiguës

Quand on imbrique des boucles les unes dans les autres, il peut arriver que deux boucles aient des balises homonymes.

Par exemple, dans le code suivant :

```
<BOUCLE_rubriques(RUBRIQUES){id_rubrique}>
     <BOUCLE_articles(ARTICLES){id_rubrique}>
         #TITRE
     </BOUCLE_articles>
</BOUCLE_rubriques>
```

la balise **#TITRE** désigne le titre d'un article. Ainsi, si on voulait afficher le titre de la rubrique à l'intérieur de la boucle **_articles**, on ne pourrait pas utiliser `#TITRE`.

On peut appeler une balise homonyme de l'une des boucles englobantes en explicitant le nom de la boucle à laquelle la balise appartient. Il faut alors spécifier le nom de la boucle entre le # et le nom de la balise.

On écrira alors la balise `#BALISE` de la boucle `_boucle`[1] de la façon suivante : **#*_boucle*:BALISE**. Par exemple :

```
<BOUCLE_rubriques(RUBRIQUES){id_rubrique}>
     <BOUCLE_articles(ARTICLES){id_rubrique}>
         #_rubriques:TITRE > #TITRE
     </BOUCLE_articles>
</BOUCLE_rubriques>
```

affichera le titre de la rubrique, puis le titre de l'article : la balise `#TITRE` pour la boucle *_rubriques* devient `#_rubriques:TITRE` pour ne pas être confondue avec la balise `#TITRE` de la boucle *_articles*.

Filtrer les résultats

Il est fréquent de vouloir modifier un élément tiré de la base de données, soit pour obtenir un affichage différent (par exemple, afficher le titre entièrement en majuscules), ou pour récupérer une valeur découlant de cet élément (par exemple, afficher le jour de la semaine correspondant à une date).

Dans SPIP, on peut directement appliquer des *filtres* aux éléments récupérés de la base de données, en les indiquant dans la syntaxe des balises SPIP, qui devient :

```
[ option avant (#BALISE|filtre1|filtre2|...|filtren) option après ]
```

La syntaxe est donc de faire suivre le nom de la balise, entre les parenthèses, par les filtres succesifs, séparés par une barre verticale (nommée habituellement *pipe*).

- *attention : l'application de filtres ne fonctionne pas sur les balises dynamiques.*

Voici quelques filtres fournis par SPIP :

- *majuscules*, passe le texte en majuscules (plus puissant que la fonction de PHP correspondante, qui ne fonctionne pas correctement avec les caractères accentués) ; par exemple :

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

- *justifier*, affiche le texte en justification totale (c'est-à-dire `<P align=justify>`) ; par exemple :

```
[(#TEXTE|justifier)]
```

La présente documentation consacre un article aux différents filtres livrés avec SPIP.

[1] *Précision* : n'oubliez pas le cas échéant, l'underscore "_" initial dans le nom de la boucle si celui ci ne commence pas par un numéro.

Court-circuiter le traitement par SPIP

SPIP applique un traitement typographique à tous les textes tirés de la base de données. En particulier, il place des espaces insécables avant certains symboles (point-virgule, point d'interrogation, etc.), et analyse des raccourcis de mise en page.

Dans certains cas, vous pouvez avoir besoin de court-circuiter ce traitement, afin de récupérer directement le texte brut tel qu'il est placé dans la base de données. Pour cela, il suffit d'ajouter une astérisque (*) à la suite de la balise SPIP. Ce qui donne :

```
[ option avant (#BALISE*|filtre1|filtre2|...|filtren) option après ]
```

(voir #BALISE* et #BALISE**)

Les paramètres des balises

Certaines balises[2] acceptent des paramètres. On passera alors une liste de paramètres entre accolades «{» et «}» avec des virgules « , » pour séparer chaque paramètre. Par exemple : `#ENV{lang,fr}`.

Un paramètre peut être une constante ou une autre balise. Seulement les balises de forme simple peuvent être passées en paramètres (i.e. pas de code optionnel ou de filtres). On peut mettre les paramètres entre guillemets simples « '...' » si l'on ne veut pas qu'ils soient interprétés par SPIP.

[2] **#ENV** et **#EXPOSER**

La boucle ARTICLES

Mai 2001 — maj : mars 2010

Une boucle d'articles se code en plaçant entre parenthèses `ARTICLES` (avec un « s ») :

```
<BOUCLEn(ARTICLES){critères...}>
```

Les éléments contenus dans une telle boucle sont des articles.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{id_article}`** sélectionne l'article dont l'identifiant est `id_article`. Comme l'identifiant de chaque article est unique, ce critère ne retourne qu'une ou zéro réponse.

- **`{id_rubrique}`** sélectionne les articles contenus dans la rubrique dont l'identifiant est `id_rubrique`.

- **`{id_secteur}`** sélectionne les articles dans ce secteur (un secteur est une rubrique qui ne dépend d'aucune autre rubrique, c'est-à-dire située à la racine du site).

- **`{branche}`** sélectionne l'ensemble des articles de la rubrique ET de ses sous-rubriques. (C'est une sorte d'extension du critère `{id_secteur}`. Toutefois, à l'inverse de `{id_secteur=2}`, il n'est pas possible d'appeler directement une *branche* en faisant par exemple `{branche=2}` : techniquement parlant, il faut que la rubrique en question figure dans le contexte courant. Ce critère est à utiliser avec parcimonie : si votre site est bien structuré, vous ne devriez pas en avoir besoin, sauf dans des cas très particuliers.)

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

- **`{id_mot}`** sélectionne les articles correspondant à cet identifiant de mot-clé (utile pour indiquer la liste des articles traitant d'un sujet donné).

- **`{titre_mot=xxxx}`**, ou **`{type_mot=yyyy}`** sélectionne respectivement les articles liés au mot-clé dont le nom est « xxxx », ou liés à des mots-clés du groupe de mots-clés « yyyy ». Si l'on donne plusieurs critères `{titre_mot=xxxx}` (ou plusieurs `{type_mot=yyyy}`), on sélectionnera ceux qui auront tous ces mots à la fois.

- **`{id_groupe=zzzz}`** permet de sélectionner les articles liés à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ». [Nota : Ce critère n'est pas (en l'état actuel du développement de SPIP) cumulable avec le précédent `{type_mot=yyyy}`]

- **`{lang}`** sélectionne les articles de la langue demandée dans l'adresse de la page.

- **{traduction}** sélectionne les traductions de l'article courant en différentes langues.

- **{origine_traduction}** sélectionne les articles qui servent de base à des versions traduites (les articles "originaux").

Voir sur **programmer.spip.org** pour des exemples

- Les critères **{date}** (ou {date=...} ou {date==...}) permettent de sélectionner un article en fonction de la date passée dans l'URL.

- **{recherche}** sélectionne les articles correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP).

Le critère **{tout}** permet de sélectionner les données d'une table comme si aucun autre critère restrictif n'était appliqué.
Ainsi son utilisation relève-t'elle plus de l'*aide-mémoire* pour le webmestre puisque l'on obtient le même résultat en n'indiquant aucun critère.

Le statut de l'article

Comme toutes les boucles de SPIP, une boucle ARTICLES ne retourne que des articles *publiés* ; dans le cas où le site est réglé de manière à ne pas publier les articles « post-datés », un autre test est fait sur la date de l'article. Il est possible de débrayer ce système et d'afficher les articles « en cours de rédaction », « proposés à la publication » ou « refusés » grâce au critère {statut} :

- **{statut IN prop,prepa,publie,refuse,poubelle}** sélectionne les articles en fonction de leur statut de publication :
- **{statut=prepa}** sélectionne les articles en cours de rédaction dans l'espace privé ;
- **{statut=prop}** sélectionne les articles proposés à la publication ;
- **{statut=publie}** sélectionne les articles publiés sur le site, y compris les articles « post-datés » ;
- **{statut=refuse}** sélectionne les articles qui ont été refusés à la publication ;
- **{statut=poubelle}** sélectionne les articles qui ont été mis à la poubelle.

Les critères d'affichage

Une fois fixé l'un des critères ci-dessus, on pourra ajouter les critères suivants pour restreindre le nombre d'éléments affichés.

Les critères communs à toutes les boucles s'appliquent évidemment.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (par exemple : {par date} ou {par titre}).

- **#ID_ARTICLE** affiche l'identifiant unique de l'article. Utile pour fabriquer des liens hypertextes non prévus (par exemple vers une page « Afficher au format impression »).

- **#SURTITRE** affiche le surtitre.

- **#TITRE** affiche le titre de l'article.

- **#SOUSTITRE** affiche le soustitre.

- **#DESCRIPTIF** affiche le descriptif.

- **#CHAPO** affiche le texte d'introduction (chapeau).

- **#TEXTE** affiche le texte principal de l'article.

- **#PS** affiche le post-scriptum.

- Les balises de dates : `#DATE, #DATE_REDAC, #DATE_MODIF` sont explicitées dans la documentation sur « La gestion des dates ».

- **#ID_RUBRIQUE** affiche l'identifiant de la rubrique dont dépend l'article.

- **#ID_SECTEUR** affiche l'identifiant du secteur dont dépend l'article (le secteur étant la rubrique parente située à la racine du site).

- **#NOM_SITE** et **#URL_SITE** affichent le nom et l'url du « lien hypertexte » de l'article (si vous avez activé cette option).

- **#VISITES** affiche le nombre total de visites sur cet article.

- **#POPULARITE** affiche le pourcentage de popularité de cet article ; voir la documentation « La « popularité » des articles ».

- **#LANG** affiche la langue de cet article.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP (Ils ne peuvent pas être utilisés comme critère de classement).

- **#URL_ARTICLE** affiche l'URL de la page de l'article.

- **#NOTES** affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- **#INTRODUCTION** affiche le descriptif de l'article, sinon affiche les 600 premiers caractères du début de l'article (chapeau puis texte).

- **#LESAUTEURS** affiche les auteurs de cet article, avec lien vers leur propre page publique (afin de pouvoir directement leur écrire ou de consulter la liste des articles qu'ils ont publié). Cela évite de créer une boucle AUTEURS pour obtenir le même résultat.

- **#PETITION** affiche le texte de la pétition si elle existe. Si elle existe mais que le texte est vide, retourne un espace (une chaîne non vide sans incidence dans une page html).

- **#FORMULAIRE_SIGNATURE** fabrique et affiche le formulaire permettant de signer la pétition associée à cet article.

- **#FORMULAIRE_FORUM** fabrique et affiche le formulaire permettant de poster un message répondant à cet article. Pour en savoir plus, voir aussi « Les formulaires ».

- **#PARAMETRES_FORUM** fabrique et affiche la liste des variables exploitées par le formulaire permettant de répondre à cet article. Par exemple :

```
[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">Répondre à cet article</a>]
```

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message.

Par exemple :

```
<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">
Répondre à cet article</a>
```

renverra le visiteur sur la page actuelle une fois que le message a été validé.

Les logos

- **#LOGO_ARTICLE** affiche le logo de l'article, éventuellement avec la gestion du survol.

- **#LOGO_RUBRIQUE** affiche le logo de la rubrique de l'article.

- **#LOGO_ARTICLE_RUBRIQUE** affiche le logo de l'article, éventuellement remplacé par le logo de la rubrique s'il n'existe pas de logo spécifique à l'article.

Les logos s'installent de la manière suivante : `[(#LOGO_ARTICLE|alignement|adresse)]`

L'alignement peut être `left` ou `right`. L'adresse est l'URL de destination du lien de ce logo (par exemple `#URL_ARTICLE`). Si l'on n'indique pas d'adresse, le bouton n'est pas cliquable.

Si l'on veut récupérer directement le nom du fichier du logo (alors que les balises précédentes fabriquent le code HTML complet pour insérer l'image dans la page), par exemple pour afficher une image en fond de tableau, on utilisera le filtre `|fichier` comme suit : `[(#LOGO_ARTICLE|fichier)]`

Par ailleurs deux balises permettent de récupérer un seul des deux logos :

- **#LOGO_ARTICLE_NORMAL** affiche le logo sans survol ;

- **#LOGO_ARTICLE_SURVOL** affiche le logo de survol.

La boucle RUBRIQUES

Mai 2001 — maj : août 2010

La boucle `RUBRIQUES` retourne une liste de... rubriques (étonnant, non ?)

`<BOUCLEn(RUBRIQUES){critères...}>`

Remarque. Une boucle `RUBRIQUES` n'affiche que des rubriques « actives », c'est-à-dire contenant des articles publiés, des documents joints, des sites publiés — ou des sous-rubriques elles-mêmes actives. De cette façon, on évite de se trouver dans des rubriques « culs de sac » n'offrant aucun élément de navigation. Il est possible de forcer l'affichage des rubriques vides (voir ci-dessous, le critère `{tout}`).

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{id_rubrique}`** sélectionne la rubrique dont l'identifiant est `id_rubrique`. Comme l'identifiant de chaque rubrique est unique, ce critère retourne une ou zéro réponse.

- **`{id_secteur}`** sélectionne les rubriques de ce secteur. (On peut également, par extension, utiliser le critère `{branche}` décrit dans la boucle ARTICLES).

- **`{id_parent}`** sélectionne la liste des rubriques contenues dans une rubrique.

- **`{racine}`** sélectionne la liste des secteurs (rigoureusement identique à `{id_parent=0}`).

- **`{id_enfant}`** sélectionne la rubrique qui contient la rubrique (une seule réponse ; ou zéro réponse si la présente rubrique est située à la racine du site).

- **`{meme_parent}`** sélectionne la liste des rubriques dépendant de la même rubrique que la rubrique en cours. Permet d'afficher les rubriques « sœurs » qui se trouvent au même niveau dans la hiérarchie.

- Les rubriques peuvent être liées à des mots-clés. Les critères de mots-clés peuvent donc être utilisés dans les boucles `(RUBRIQUES)` :

  - `{id_mot}`, `{titre_mot=xxx}` récupèrent les rubriques liées au mot dont le numéro est *id_mot* ou dont le titre est *titre_mot* ;
  - `{id_groupe}`, `{type_mot=yyyy}` récupèrent les rubriques liées à des mots du groupe *id_groupe*, ou du groupe dont le titre est *type_mot*.

- **`{recherche}`** sélectionne les rubriques correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP). Voir la page consacrée au moteur de recherche.

- **`{lang}`** sélectionne les rubriques de la langue demandée dans l'adresse de la page.

- **{tout}** sélectionne *toutes* les rubriques, c'est-à-dire : les rubriques vides *en plus* des rubriques contenant des éléments publiés.
On réservera ce choix à des besoins très spécifiques ; en effet, par défaut, SPIP n'affiche pas sur le site public les rubriques qui ne contiennent aucun élément actif, afin de garantir que le site ne propose pas de « culs de sac » (navigation vers des pages ne proposant aucun contenu).

Les critères d'affichage

Une fois fixé l'un des critères ci-dessus, on pourra ajouter les critères suivants pour restreindre le nombre d'éléments affichés.

- Les critères communs à toutes les boucles s'appliquent évidemment.

- **{exclus}** permet d'exclure du résultat la rubrique dans laquelle on se trouve déjà (utile avec `meme_parent`).

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **#ID_RUBRIQUE** affiche l'identifiant unique de la rubrique.

- **#TITRE** affiche le titre de la rubrique.

- **#DESCRIPTIF** affiche le descriptif.

- **#TEXTE** affiche le texte principal de la rubrique.

- **#ID_SECTEUR** affiche l'identifiant du secteur dont dépend la rubrique (le secteur étant la rubrique située à la racine du site).

- **#ID_PARENT** affiche l'identifiant de la rubrique qui contient la rubrique actuelle. Si la rubrique est située à la racine du site, retourne 0.

- **#LANG** affiche la langue de cette rubrique.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP. (Ils ne peuvent pas être utilisés comme critère de classement.)

- **#NOTES** affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- **#INTRODUCTION** affiche les 600 premiers caractères du texte, les enrichissements typographiques (gras, italique) sont supprimés.

- **#URL_RUBRIQUE** affiche l'URL de la page de la rubrique.

- **#DATE** affiche la date de la dernière publication effectuée dans la rubrique et/ou ses sous-rubriques (articles, brèves...).

- **#FORMULAIRE_FORUM** fabrique et affiche le formulaire permettant de poster un message répondant à cette rubrique. Pour en savoir plus, voir aussi « Les formulaires ».

- **#PARAMETRES_FORUM** fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à cette rubrique. Par exemple :

```
[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">
Répondre à cette rubrique</a>]
```

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple :

```
<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à
cette rubrique</a>
```

renverra le visiteur sur la page actuelle une fois que le message a été validé.

- **#FORMULAIRE_SITE** fabrique et affiche un formulaire permettant aux visiteurs du site de proposer des référencements de sites. Ces sites apparaîtront comme « proposés » dans l'espace privé, en attendant une validation par les administrateurs.

Ce formulaire ne s'affiche que si vous avez activé l'option « Gérer un annuaire de sites » dans la *Configuration sur site* dans l'espace privé, et si vous avez réglé « Qui peut proposer des sites référencés » sur « les visiteurs du site public ».

Le logo

- **#LOGO_RUBRIQUE** le logo de la rubrique, éventuellement avec la gestion du survol. S'il n'y a pas de logo pour cette rubrique, SPIP va automatiquement chercher s'il existe un logo pour la rubrique dont elle dépend, et ainsi de suite de manière récursive.

Le logo s'installe de la manière suivante :

```
[(#LOGO_RUBRIQUE|alignement|adresse)]
```

Où :

- *alignement* est l'alignement du logo (left ou right)
- *adresse* est une url si on veut ajouter un lien directement sur le logo (par exemple `#URL_RUBRIQUE`).

`#LOGO_RUBRIQUE_NORMAL` affiche le logo « sans survol » ; `#LOGO_RUBRIQUE_SURVOL` affiche le logo de survol : ces deux balises permettent par exemple, quand on est dans une rubrique, de gérer un logo « avec survol » pour les liens vers les autres rubriques, et de laisser le logo de survol seul dans la rubrique active.

Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

La boucle BREVES

Mai 2001 — maj : août 2010

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

```
<BOUCLEn(BREVES){critères...}>
```

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{tout}`** : les brèves sont sélectionnées dans l'intégralité du site.

- **`{id_breve}`** sélectionne la brève dont l'identifiant est `id_breve`. Comme l'identifiant de chaque brève est unique, ce critère retourne une ou zéro réponse.

- **`{id_rubrique}`** sélectionne toutes les brèves contenues dans la rubrique en cours.

- **`{id_mot}`** sélectionne toutes les brèves liées au mot-clé en cours (à l'intérieur d'une boucle de type MOTS).

- **`{titre_mot=xxxx}`**, ou **`{type_mot=yyyy}`** sélectionne les brèves liées au mot-clé dont le nom est « xxxx », ou liées à des mots-clés du groupe de mots-clés « yyyy ». Attention, on ne peut pas utiliser plusieurs critères `{titre_mot=xxxx}` ou `{type_mot=yyyy}` dans une même boucle.

- **`{id_groupe=zzzz}`** permet de sélectionner les brèves liées à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ».

- **`{lang}`** sélectionne les brèves de la langue demandée dans l'adresse de la page.

- **`{recherche}`** sélectionne les brèves correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP). Voir la page consacrée au moteur de recherche.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **`#ID_BREVE`** affiche l'identifiant unique de la brève.

- **`#TITRE`** affiche le titre de la brève.

- **`#DATE`** affiche la date de publication de la brève.

- **`#TEXTE`** affiche le texte de la brève.

- **`#NOM_SITE`** affiche le nom du site indiqué en références.

- **`#URL_SITE`** affiche l'adresse (URL) du site indiqué en références.

- **`#ID_RUBRIQUE`** affiche l'identifiant de la rubrique dont dépend cette brève.

- **`#LANG`** affiche la langue de cette brève. Par défaut, la langue d'une brève est la langue du secteur dans lequel elle se trouve.

Les balises calculées par SPIP

Les éléments suivants sont calculés par SPIP (Ils ne peuvent pas être utilisés comme critère de classement).

- **`#NOTES`** affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- **`#INTRODUCTION`** affiche les 600 premiers caractères du texte, les enrichissements typographiques (gras, italique) sont supprimés.

- **`#URL_BREVE`** affiche l'URL de la page de la brève.

- **`#FORMULAIRE_FORUM`** fabrique et affiche le formulaire permettant de poster un message répondant à cette brève. Pour en savoir plus, voir aussi « Les formulaires ».

- **`#PARAMETRES_FORUM`** fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à cette brève :

```
[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">
Répondre à cette brève</a>]
```

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à cette bréve</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

Le logo

- **`#LOGO_BREVE`** affiche le logo de la brève, éventuellement avec la gestion du survol.

Le logo s'installe de la manière suivante : `[(#LOGO_BREVE|alignement|adresse)]`

- **#LOGO_BREVE_RUBRIQUE** affiche, si il existe, le logo de la brève ; si ce logo n'a pas été attribué, SPIP affiche le logo de la rubrique.

Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

.

La boucle AUTEURS

Mai 2001 — maj : Octobre 2009

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

```
<BOUCLEn(AUTEURS){critères...}>
```

Si l'on ne précise pas de critère de sélection, la boucle retournera tous les auteurs *ayant un article publié.*

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{tout}`** : les auteurs sont sélectionnés, qu'ils aient écrit un article ou non.

- **`{id_auteur}`** retourne l'auteur dont l'identifiant est `id_auteur`. Comme l'identifiant de chaque auteur est unique, ce critère retourne une ou zéro réponse.

- **`{id_article}`** retourne tous les auteurs de cet article.

- **`{lang}`** sélectionne les auteurs qui ont choisi, dans l'espace privé, la langue demandée dans l'adresse de la page. Si un auteur ne s'est jamais connecté dans l'espace privé, il ne sera pas trouvé par ce critère.

- **`{lang_select}`** Par défaut, une boucle AUTEURS affiche les balises et les chaînes localisées dans la langue du contexte (de la boucle englobante ou de l'url). Si on utilise ce critère, ces informations seront localisées dans la langue choisie par l'auteur.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par nom}`).

- **`#ID_AUTEUR`** affiche l'identifiant unique de l'auteur.

- **`#NOM`** affiche le nom de l'auteur.

- **`#BIO`** affiche la biographie de l'auteur.

- **`#EMAIL`** affiche son adresse email.

- **#NOM_SITE** affiche le nom de son site Web.

- **#URL_SITE** affiche l'adresse (URL) de son site.

- **#PGP** affiche sa clé publique pour PGP.

- **#LANG** affiche la langue de l'auteur (c'est-à-dire celle qu'il a choisie dans l'espace privé).

Les balises calculées par SPIP

- **#FORMULAIRE_ECRIRE_AUTEUR** fabrique et affiche un formulaire permettant d'écrire à l'auteur. Il faut que le serveur hébergeant le site accepte d'envoyer des mails. Ce système permet de ne pas divulguer l'adresse email de l'auteur. NB : pour que le formulaire soit affiché il faut que l'auteur ait renseigné le champ **email** de sa fiche auteur.

- **#NOTES** affiche les notes de bas de page (calculées à partir de l'analyse du texte).

- **#URL_AUTEUR** affiche l'adresse de la page `spip.php?auteurxxx`.

Le logo

- **#LOGO_AUTEUR** le logo de l'auteur, éventuellement avec la gestion du survol.

Le logo s'installe de la manière suivante : `[(#LOGO_AUTEUR|alignement|adresse)]`

Les variantes `#LOGO_AUTEUR_NORMAL` et `#LOGO_AUTEUR_SURVOL` permettent un affichage plus fin de ces deux variantes du logo.

La boucle FORUMS

Mai 2001 — maj : Décembre 2007

La boucle `FORUMS` retourne une liste de messages de forums.

`<BOUCLEn(FORUMS){critères...}>`

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{id_forum}`** retourne le message dont l'identifiant est `id_forum`. Comme l'identifiant de chaque message est unique, ce critère retourne une ou zéro réponse.

- **`{id_article}`** retourne les messages correspondant à cet article.

- **`{id_rubrique}`** retourne les messages correspondant à cette rubrique. Attention, il ne s'agit pas de messages des articles de cette rubrique, mais bien des messages de cette rubrique. En effet, il est possible d'activer dans l'espace privé des forums pour chaque rubrique.

- **`{id_breve}`** retourne les messages correspondant à cette brève.

- **`{id_syndic}`** retourne les messages correspondant à ce site.

- **`{id_thread}`** retourne les messages appartenant à ce fil de discussion.
Note : `id_thread` n'est rien d'autre que l'identifiant `id_forum` du message qui démarre le fil de discussion (aussi appelé « pied » de la discussion).

- **`{id_parent}`** retourne les messages dépendant d'un autre message. Indispensable pour gérer des fils de discussion (« *threads* ») dans les forums.

- **`{id_enfant}`** retourne le message dont dépend le message actuel (permet de « remonter » dans la hiérachie des fils de discussion).

- **`{meme_parent}`** retourne les autres messages répondant à un même message.

- **`{plat}`** ou **`{tout}`** : affiche tous les messages de forum sans prendre en compte leur hiérarchie : avec ce critère, vous pouvez sélectionner tous les messages quelle que soit leur position dans un thread (dans la limite des autres critères, bien sûr). Cela permet par exemple d'afficher les messages par ordre strictement chronologique par exemple, ou de compter le nombre total de contributions dans un forum.

N.B. En l'absence de critère `{id_forum}` ou `{id_parent}`, lorsque `{plat}` n'est pas utilisé, seuls les messages n'ayant pas de parent (i.e. à la racine d'un thread) sont affichés.

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

- Les messages des forums peuvent être liées à des mots-clés. Les critères de mots-clés peuvent donc être désormais utilisés dans les boucles `(FORUMS)` :

- `{id_mot}`, `{titre_mot=xxx}` récupèrent les messages liés au mot dont le numéro est *id_mot* ou dont le titre est *titre_mot* ;
- `{id_groupe}`, `{type_mot=yyyy}` récupèrent les messages liés à des mots du groupe *id_groupe*, ou du groupe dont le titre est *type_mot*.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

- **Les balises tirées de la base de données**

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **`#ID_FORUM`** affiche l'identifiant unique du message.

- **`#ID_THREAD`** affiche l'identifiant du fil de discussion auquel appartient ce message. (Il s'agit de l'`id_forum` du *pied* de la discussion.)

- **`#URL_FORUM`** donne l'adresse canonique de la page qui affiche le message de forum (par exemple, avec les URLs normales de SPIP, `article.php?id_article=8#forum15` pour le message 15 associé à l'article 8).

- **`#ID_BREVE`** affiche l'identifiant de la brève à laquelle ce message est attaché. Attention, cela n'est pas récursif : un message qui répond à un message attaché à une brève ne contient pas lui-même le numéro de la brève.

- **`#ID_ARTICLE`** est l'identifiant de l'article auquel répond le message.

- **`#ID_RUBRIQUE`** est l'identifiant de la rubrique à laquelle le message répond.

- **`#ID_SYNDIC`** est l'identifiant du site auquel le message répond.

- **`#DATE`** est la date de publication.

- **`#TITRE`** est le titre.

- **`#TEXTE`** est le texte du message.

- **`#NOM_SITE`** le nom du site Web indiqué par l'auteur.

- **`#URL_SITE`** l'adresse (URL) de ce site Web.

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

- **#EMAIL** est l'adresse email de l'auteur, sous forme directe d'envoi (le mailto: est intégré). Elle s'utilise donc ainsi `[(#EMAIL)]` pour avoir à la fois l'adresse mail apparente et, directement, le lien à cliquer sur cette adresse.

- **#IP** est l'adresse IP de l'auteur du message au moment de l'envoi de sa contribution.

- **Les balises calculées par SPIP**

- **#FORMULAIRE_FORUM** fabrique l'interface permettant de poster un message de réponse. Pour en savoir plus, voir aussi « Les formulaires ».

- **#PARAMETRES_FORUM** fabrique la liste des variables exploitées par l'interface du formulaire permettant de répondre à ce message. Par exemple :

```
[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">
Répondre à ce message</a>]
```

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple : `<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à ce message</a>` renverra le visiteur sur la page actuelle une fois que le message a été validé.

La boucle MOTS

Mai 2001 — maj : août 2010

La boucle `MOTS` retourne une liste de mots-clés.

`<BOUCLEn(MOTS){critères...}>`

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{tout}`** les mots sont sélectionnés dans l'intégralité du site.

- **`{id_mot}`** retourne le mot-clé dont l'identifiant est *`id_mot`*.

- **`{id_groupe}`** retourne les mots-clés associés au groupe de mots dont le numéro est *id_groupe*.

- **`{id_article}`** retourne les mots-clés associés à cet article (c'est l'utilisation la plus courante de cette boucle).

- **`{id_rubrique}`** retourne les mots-clés associés à une rubrique.

- **`{id_breve}`** retourne les mots associés à une brève.

- **`{id_syndic}`** retourne les mots associés à un site référencé.

- **`{id_forum}`** retourne les mots associés à un message de forum .(attention, utilisation très spécifique).

- **`{titre=france}`** retourne le mot-clé intitulé `france` (par exemple).

- **`{type=pays}`** retourne les mots-clés du groupe de mots-clés intitulé `pays` (par exemple).

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **`#ID_MOT`** affiche l'identifiant unique du mot.

- **#TITRE** affiche le titre (le mot-clé lui-même).

- **#DESCRIPTIF** affiche le descriptif du mot.

- **#TEXTE** affiche le texte associé au mot.

- **#TYPE** affiche la catégorie dans laquelle est installé ce mot-clé (par exemple, le mot-clé « France » pourrait être associé à la catégorie « Pays »).

- **#LOGO_MOT** affiche le logo associé au mot-clé.

Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

- **#URL_MOT** affiche l'adresse de ce mot

La boucle (GROUPES_MOTS)

D'une utilisation marginale, la boucle **GROUPES_MOTS** mérite d'être citée ici : elle permet, si vous avez plusieurs groupes de mots-clés, de sélectionner ces groupes, et d'organiser par exemple une page récapitulative de tous les mots-clés classés par groupe, puis par ordre alphabétique à l'intérieur de chaque groupe, par exemple via le code suivant :

```
<BOUCLE_groupes(GROUPES_MOTS){par titre}>
<h1>#TITRE</H1>
<BOUCLE_mots(MOTS){id_groupe}{par titre}{" - "}>
#TITRE
</BOUCLE_mots>
</BOUCLE_groupes>
```

Les balises et critères associés à cette boucle sont :
- **#ID_GROUPE**, l'identifiant du groupe de mots [également disponible dans la boucle(MOTS)] ;
- **#TITRE**, le titre du groupe [à l'intérieur de la boucle(MOTS), vous pouvez utiliser `#TYPE` pour afficher cette valeur].

La boucle DOCUMENTS

Juin 2002 — maj : Mai 2009

La boucle DOCUMENTS retourne une liste de documents multimédia associés (à un article, à une rubrique, éventuellement les images incluses dans une brève).

`<BOUCLEn(DOCUMENTS){critères...}>`

Cette boucle gère non seulement les documents joints non installés dans le texte d'un article, mais peut aussi accéder aux *images* (depuis la version 1.4, les images sont gérées, au niveau du programme, comme un genre spécifique de documents), aux vignettes de prévisualisation et aux documents déjà insérés dans le corps de l'article.

Pour mémoire, on utilisera donc le plus fréquemment (utilisation courante) la boucle DOCUMENTS avec, au minimum, les critères suivants (explications ci-après) :

<BOUCLEn(DOCUMENTS){mode=document}{doublons}>

Les critères de sélection

Une boucle DOCUMENTS s'utilise en général à l'intérieur d'un article ou d'une rubrique (éventuellement dans une brève, mais ici l'utilisation sera réservée à la récupération d'images, ce qui sera très spécifique).

- `{id_document}` sélectionne le document dont l'identifiant est id_document. Comme l'identifiant de chaque document est unique, ce critère ne retourne qu'une ou zéro réponse.

- `{id_article}` retourne les documents de l'article dont l'identifiant est *id_article*.

- `{id_rubrique}` retourne les documents de la rubrique *id_rubrique*.

- `{id_breve}` retourne les documents de la brève *id_breve* (il n'est pas possible d'associer des documents multimédia à une brève, seulement des images ; l'utilisation d'une boucle DOCUMENTS dans ce cadre sera donc très spécifique).

Notez bien : il n'est pas possible d'utiliser ici le critère `{id_secteur}` ; les documents sont conçus pour être intimement liés aux articles et aux rubriques, et non à être appelés seuls sans ces éléments (on parle dans SPIP de « documents joints »).

Les critères d'affichage

- `{mode=document}` ou `{mode=image}` permet d'indiquer si l'on veut appeler les documents multimédia, ou les images (en effet, désormais les images associées à l'article et éventuellement insérées dans l'article sont traités comme des *documents* en *mode=image*).

- `{extension=...}` permet de sélectionner les documents selon leur terminaison (terminaison du fichier multimédia, par exemple « mov », « ra », « avi »...). Cela peut être utilisé par exemple pour réaliser un *porfolio*, c'est-à-dire une boucle n'affichant que les documents de type image, une seconde boucle ensuite, avec une présentation graphique différente, les autres types de documents :

```
<BOUCLE_portfolio(DOCUMENTS){id_article}{extension==jpg|png|gif}
{mode=document}{doublons}>
```

Cette BOUCLE_portfolio récupère les documents joints à un article, non déjà affichés dans le texte de l'article, et donc les extensions des fichiers peuvent être « jpg », « png » ou « gif ».

- `{distant}` permet de sélectionner les documents selon qu'ils soient distant ou non. C'est à dire stockés sur un autre site ou téléchargés dans l'espace web du site. On précisera `{distant=oui}` et `{distant=non}` respectivement.

- {doublons} prend ici une importance particulière : elle permet non seulement de ne pas réafficher des documents déjà affichés par une autre boucle, mais également de ne pas réafficher les documents déjà intégrés à l'intérieur d'un article. Si l'on oublie ce critère, on affichera *tous* les documents associés à un article, *y compris* ceux qui auraient déjà été affichés à l'intérieur du texte[3] [1].

Les balises

- `#LOGO_DOCUMENT` affiche le logo (vignette de prévisualisation) associé au document ; si une vignette personnalisée n'a pas été installée manuellement par l'auteur de l'article, SPIP utilise une vignette standard selon le type du fichier.

- `#URL_DOCUMENT` est l'URL du fichier multimédia. Pour afficher une vignette cliquable pointant vers le document multimédia, on utilisera donc le code suivant :

```
[(#LOGO_DOCUMENT|#URL_DOCUMENT)]
```

- `#TITRE` affiche le titre du document.

- `#DESCRIPTIF` affiche le descriptif du document.

- `#FICHIER` affiche le nom de fichier du document, plus précisément son URL relative. Pour obtenir le nom de fichier seul, on passera ce filtre : `[(#FICHIER|basename)]`.

Une utilisation intéressante de cette balise est combinée avec le filtre `image_reduire`, dans le cadre d'un portfolio pour afficher une réduction de l'image plutôt que de son logo ; par exemple en utilisant :

```
[<a href="#URL_DOCUMENT">(#FICHIER|image_reduire{500}) </a>]
```

- `#TYPE_DOCUMENT` affiche le type (fichier Quicktime, fichier Real...) du document multimédia.

- `#EXTENSION` comme son nom l'indique, affiche l'extension du format du fichier, par exemple : `pdf`, `jpeg`, `move`, `ra`.

- `#TAILLE` affiche la taille du fichier multimédia. Ce chiffre est fourni en octets. Pour de gros fichiers, cette valeur devient rapidement inutilisable ; on pourra donc lui appliquer le filtre

[3] Si on utilise un critère avec un nom (`{doublons unnom}`), celui ci n'exclura pas les documents intégrés dans le texte de l'article.

`taille_en_octets`, qui affichera successivement en octets, en kilooctets, ou même en mégaoctets :

```
[(#TAILLE|taille_en_octets)]
```

- `#LARGEUR` et `#HAUTEUR` fournissent les dimensions en pixels.

- `#MIME_TYPE` affiche le type MIME du fichier — par exemple : `image/jpeg` —, *cf.* Type de média internet.

- `#DATE` est la date de mise en ligne du document. (Modifiable après l'ajout). Voir « La gestion des dates » pour des détails sur l'utilisation du champ `#DATE`.

- `#ID_DOCUMENT` affiche le numéro du document.

- `#DISTANT` est une balise qui affiche « oui » ou « non » selon que le document est distant (référencé par une url) ou pas.
Exemple :
```
#URL_DOCUMENT[(#DISTANT|=={oui}|oui)|parametre_url{nom,valeur}]
```

- `#EMBED_DOCUMENT` est une balise permettant d'incruster le document dans la page produite plutôt que de le référencer. Cette balise est aujourd'hui dépréciée, étant un cas particulier des *modèles* (voir Utiliser les modèles) qui permettent de moduler cette incrustation de manière plus souple et plus efficace.

Cette balise peut être complétée de paramètres propres aux formats utilisés, par exemple :

```
[(#EMBED_DOCUMENT|autostart=true)]
```

mais on lui préférera ce en quoi elle se traduit automatiquement à présent :

```
#MODELE{emb, autostart=true}
```

La boucle SITES (ou SYNDICATION)

Mai 2001 — maj : Mars 2009

La boucle `SITES` retourne une liste de sites référencés.

`<BOUCLEn(SITES){critères...}>`

Si l'on a syndiqué des sites référencés, cette boucle s'utilise, naturellement, associée à une boucle `SYNDIC_ARTICLES` qui permet de récupérer la liste des articles de ces sites.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{tout}`** sélectionne *tous* les sites référencés.

- **`{id_syndic}`** sélectionne le site référencé dont l'identifiant est *`id_syndic`*.

- **`{id_rubrique}`** sélectionne les sites référencés dans cette rubrique.

- **`{id_secteur}`** sélectionne les sites référencés dans ce secteur.

- **`{id_mot}`** sélectionne toutes les sites liés au mot-clé indiqué par le contexte (boucle `(MOTS)` englobante, paramètre d'URL etc).

- **`{titre_mot=xxxx}`**, ou **`{type_mot=yyyy}`** sélectionne les sites liés au mot-clé dont le nom est « xxxx », ou liés à des mots-clés du groupe de mots-clés « yyyy ». Si l'on donne plusieurs plusieurs critères `{titre_mot=xxxx}` (ou plusieurs `{type_mot=yyyy}`), on sélectionnera ceux qui auront tous ces mots à la fois.

- **`{id_groupe=zzzz}`** permet de sélectionner les sites liés à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ».

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

- **`{syndication=oui}`**, ou **`{syndication=non}`** permet de n'afficher que les sites référencés faisant l'objet d'une syndication, ou les sites non syndiqués.

- **`{moderation=oui}`** affiche les sites syndiqués dont les liens sont bloqués a priori (« modérés ») ; l'inverse de ce critère est `{moderation!=oui}`.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par nom_site}`).

- **`#ID_SYNDIC`** affiche l'identifiant unique du site référencé. Par exemple pour renvoyer vers la page décrivant le site *(site.html sur la /dist)* avec le code suivant :

```
<BOUCLE_sites(SITES) {id_rubrique} {par nom_site}>
<li><a
href="[(#ID_SYNDIC|generer_url_entite{site})]">#NOM_SITE</a></li>
</BOUCLE_sites>
```

- **`#NOM_SITE`** affiche le nom du site référencé.

- **`#URL_SITE`** affiche l'adresse (URL) du site référencé.

- **`#DESCRIPTIF`** affiche le descriptif du site référencé.

- **`#ID_RUBRIQUE`** affiche le numéro de la rubrique contenant ce site.

- **`#ID_SECTEUR`** affiche le numéro de la rubrique-secteur (à la racine du site) contenant ce site.

Autres balises

- **`#LOGO_SITE`** affiche le logo attribué au site.

- **`#URL_SYNDIC`** affiche l'adresse (URL) du fichier de syndication de ce site.

- **`#FORMULAIRE_FORUM`** fabrique et affiche le formulaire permettant de poster un message de forum à propos de ce site. Pour en savoir plus, voir aussi « Les formulaires ».

- **`#PARAMETRES_FORUM`** fabrique la liste des variables exploitées par l'interface du formulaire permettant de poster un message de forum à propos de ce site. Par exemple : `[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">Répondre à ce forum</a>]`

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple :

```
<a href="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">Répondre à
ce message</a>
```

renverra le visiteur sur la page actuelle une fois que le message a été validé.

La boucle SYNDIC_ARTICLES

Mai 2001 — maj : Janvier 2007

La boucle `SYNDIC_ARTICLES` retourne une liste des articles des sites syndiqués.

`<BOUCLEn(SYNDIC_ARTICLES){critères...}>`

On peut soit l'utiliser à l'intérieur d'une boucle `SITES` (cette dernière récupère une liste de sites référencés, ensuite on récupère chaque article de ces sites), soit directement à l'intérieur d'une rubrique (on récupère directement tous les articles syndiqués dans une rubrique, en court-circuitant le passage par la liste des sites).

La boucle `SITES` (ou `SYNDICATION`) n'affiche pas uniquement des sites syndiqués, mais plus généralement des sites référencés (la syndication de certains sites référencés étant une option). On pourra donc, pour obtenir une présentation graphique plus précise, utiliser une boucle `SYNDIC_ARTICLES` uniquement à l'intérieur d'une boucle `SITES` utilisant le critère `{syndication=oui}`.

Les critères de sélection

On utilisera l'un ou autre des critères suivants pour indiquer comment on sélectionne les éléments.

- **`{tout}`**, tous les sites syndiqués.

- **`{id_syndic_article}`** retourne l'article syndiqué dont l'identifiant est `id_syndic_article`. (Dans la pratique, il y très peu d'intérêt à fabriquer une page pour un article syndiqué, puisqu'on préférera renvoyer directement vers l'article en question.)

- **`{id_syndic}`** retourne la liste des articles du site syndiqué dont l'identifiant est `id_syndic`.

- **`{id_rubrique}`** retourne la liste des articles syndiqués dans cette rubrique.

- **`{id_secteur}`** retourne la liste des articles syndiqués dans ce secteur.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par titre}`).

- **`#ID_SYNDIC_ARTICLE`** affiche l'identifiant unique de l'article syndiqué.

- **`#ID_SYNDIC`** affiche l'identifiant unique du site syndiqué contenant cet article.

- **#TITRE** affiche le titre de l'article.
Remarque : il est préférable d'utiliser ici le titre « brut » de l'article syndiqué - via le code `[(#TITRE*)]` -, pour éviter le moteur typographique. En effet les titres sont censés être déjà « typographiquement corrects » dans les backends, et on ne souhaite pas passer la correction typographique sur des titres en anglais ou sur des titres comprenant des expressions du genre « Les fichiers ~/.tcshrc ».

- **#URL_ARTICLE** affiche l'adresse (URL) de l'article syndiqué (sur son site original).

- **#DATE** affiche la date de publication de cet article.

- **#LESAUTEURS** affiche les auteurs de l'article syndiqué.

- **#DESCRIPTIF** affiche le descriptif de l'article syndiqué.

- **#NOM_SITE** affiche le nom du site syndiqué contenant cet article.

- **#URL_SITE** affiche l'adresse (URL) du site

La boucle SIGNATURES

Mai 2001 — maj : Octobre 2008

La boucle SIGNATURES retourne une liste de signatures de pétition.

`<BOUCLEn(SIGNATURES){critères...}>`

Les critères de sélection

Les critères disponibles sont les suivants :

- **`{id_article}`** retourne les signatures de la pétition de cet article.

- **`{id_signature}`**, la signature correspondant à l'identifiant indiqué.

- **`{id_trad}`**, retourne les signatures des pétitions associées à l'article indiqué ou à ses traductions.

- **`{tout}`** toutes les signatures sont sélectionnées dans l'intégralité du site.

Tous ces critères peuvent être conditionnels, et éventuellement combinés. Par exemple `{id_trad ?}{id_article ?}` permet de sélectionner les signatures pour un article ou pour tout un jeu de traduction d'un article, selon le contexte fourni.

Les critères d'affichage

Les critères communs à toutes les boucles s'appliquent.

Attention. Dans ce type de boucles, certains critères de classement ne sont pas identiques aux balises SPIP indiquées ci-dessous :
- `{par nom_email}` classe les résultats selon le `#NOM` du signataire ;
- `{par ad_email}` classe selon l'`#EMAIL` du signataire.

Les balises de cette boucle

Les balises tirées de la base de données

Les balises suivantes correspondent aux éléments directement tirés de la base de données. Vous pouvez les utiliser également en tant que critère de classement (généralement : `{par nom_email}`).

- **`#ID_SIGNATURE`** affiche l'identifiant unique du message.

- **`#ID_ARTICLE`** affiche l'identifiant de l'article pour cette pétition.

- **`#DATE`** affiche la date de publication.

- **`#MESSAGE`** affiche le texte du message.

- **`#NOM_EMAIL`** affiche le nom de l'auteur du message.

- **#EMAIL** affiche l’adresse email de l’auteur.
- **#NOM_SITE** affiche le nom du site Web indiqué par l’auteur.
- **#URL_SITE** affiche l’adresse (URL) de ce site Web.

La boucle HIERARCHIE

Mai 2001 — maj : Août 2008

La boucle `HIERARCHIE` retourne la liste des `RUBRIQUES` qui mènent de la racine du site à la rubrique ou à l'article en cours.

`<BOUCLEn(HIERARCHIE){critères...}>`

Les critères de sélection

On utilisera obligatoirement l'un des deux critères suivants pour indiquer comment on sélectionne les éléments :

- `{id_article}` retourne la liste des rubriques depuis la racine jusqu'à la rubrique contenant l'article correspondant à cet identifiant.

- `{id_rubrique}` retourne la liste des rubriques depuis la racine jusqu'à la rubrique correspondant à cet identifiant (exclue).

Les critères `{id_article}` ou `{id_rubrique}` ne peuvent pas être utilisés avec une comparaison. Par exemple, `<BOUCLE_hi(HIERARCHIE) {id_article=12}>` retournera une erreur.

Attention : cette boucle sera obligatoirement placée à l'intérieur d'une boucle `ARTICLES` ou `RUBRIQUES` — elle ne va pas par elle-même « chercher » l'id_article ou id_rubrique indiquée dans l'URL. (Le même principe vaut pour les boucles `HIERARCHIE` des squelettes inclus par la commande `<INCLURE{fond=xxx}>`)

Les critères d'affichage

Tous les critères de la boucle RUBRIQUES peuvent être utilisés avec cette boucle, y compris les critères de tri

(il devient possible par exemple de trier une `<BOUCLE_x(HIERARCHIE){id_article}{par hasard}>`).

Les balises de cette boucle

Les éléments obtenus avec une boucle `HIERARCHIE` sont des rubriques. On peut donc utiliser toutes les balises proposées pour les boucles `RUBRIQUES`.

Note : Il n'y a pas de critère id_breve dans HIERARCHIE mais, dans le cas d'une brève, l'usage d'id_article retournera quand même la bonne rubrique.

Les critères communs à toutes les boucles

Mai 2001 — maj : septembre 2010

Certains critères s'appliquent à (presque) tous les types de boucles. Ce sont des critères destinés à restreindre le nombre de résultats affichés ou à indiquer l'ordre d'affichage. On peut sans difficulté combiner plusieurs de ces critères de sélection.

Classer les résultats

{par *critère_de_classement***}** indique l'ordre de présentation des résultats. Ce critère de classement correspond à l'une des balises tirées de la base de données pour chaque type de boucle. Par exemple, on pourra classer les articles `{par date}`, `{par date_redac}` ou `{par titre}`. (Notez que, si les balises sont en majuscules, les critères de classement sont en minuscules.)

Cas particulier : **{par hasard}** permet d'obtenir une liste présentée dans un ordre aléatoire.

Inverser le classement. De plus, **{inverse}** provoque l'affichage du classement inversé. Par exemple `{par date}` commence par les articles les plus anciens ; avec `{par date}{inverse}` on commence la liste avec les articles les plus récents.

Le critère *inverse* peut prendre en paramètre n'importe quelle balise pour varier dynamiquement le sens du tri. Par exemple, il est possible d'écrire : `<BOUCLE_exemple(ARTICLES){par #ENV{tri}}{inverse #ENV{senstri}}>`, ce qui permet de choisir la colonne de tri et le sens du tri par l'url (&senstri=1 ou &senstri=0)

Classer par numéro. Lorsqu'on réalise le classement selon un élément de texte (par exemple le *titre*), le classement est réalisé par ordre *alphabétique*. Cependant, pour forcer un ordre d'affichage, on peut indiquer un numéro devant le titre, par exemple : « 1. Mon premier article », « 2. Deuxième article », « 3. Troisième... », etc ; avec un classement alphabétique, le classement de ces éléments donnerait la série « 1, 10, 11, 2, 3... ». Pour rétablir le classement selon les numéros, on peut utiliser le critère :

{par num critère**}**

Par exemple :

```
<BOUCLE_articles(ARTICLES){id_rubrique}{par date}{inverse}>
```

affiche les articles d'une rubrique classés selon l'ordre chronologique inversé (les plus récents au début, les plus anciens à la fin), et :

```
<BOUCLE_articles(ARTICLES){id_rubrique}{par titre}>
```

les affiche selon l'ordre alphabétique de leur titre ; enfin :

```
<BOUCLE_articles(ARTICLES){id_rubrique}{par num titre}>
```

les affiche selon l'ordre du numéro de leur titre (remarque : l'option `{par num titre}` ne fonctionne pas pour les versions de MySQL antérieures à la version 3.23).

```
<BOUCLE_articles(ARTICLES){id_rubrique}{par multi titre}>
```

Dans le cadre d'un site multilingue le critère **{par multi** `critère`**}** permet de trier par ordre alphabétique dans chaque langue. Sans l'ajout de "multi" la boucle renvoie le même classement pour chaque langue.

Classer selon plusieurs critères On peut classer selon plusieurs critères : **{par** `critère1, critère2`**}**. On indique ainsi des ordres de classement consécutifs. Les résultats seront d'abord triès selon le *critère1*, puis le *critère2* pour les résultats ayant le même *critère1*. On peut spécifier autant de critères que nécessaire.

Par exemple `{par date, titre}` triera les résultats par *date* puis les résultats ayant la même *date* seront triés par *titre*.

On peut spécifier plusieurs critères **{par** *...***}** pour une boucle pour arriver au même résultat. Par exemple : `{par date} {par titre}` est équivalent à l'exemple précédent.

Remarque : Quand on utilise plusieurs critères de tri, le critère **{inverse}** ne s'applique qu'au critère de tri placé juste avant.

La notation **{!par ...}** inverse un critère de tri en particulier. Par exemple : `{!par date} {par num titre}` tri par *date* décroissantes puis par numéros croissants dans le *titre* pour les résultats ayant la même *date*.

Comparaisons, égalités

{*critère* **<** *valeur***}** Comparaison avec une valeur fixée (on peut utiliser « > », « < », « = », « >= », « <= ». Tous les *critères de classement* (tels que tirés de la base de données) peuvent également être utilisés pour limiter le nombre de résultats.

La valeur à droite de l'opérateur peut être :

- Une valeur constante fixée dans le squelette. Par exemple :

```
<BOUCLE_art(ARTICLES){id_article=5}>
```

affiche l'article dont le numéro est 5 (utile pour mettre en vedette un article précis sur la page d'accueil).

```
<BOUCLE_art(ARTICLES){id_secteur=2}>
```

affiche les articles du secteur numéro 2.

- Une balise disponible dans le contexte de la boucle. Par exemple :

```
<BOUCLE_art(ARTICLES){id_article=5}>
<BOUCLE_titre(ARTICLES) {titre=#TITRE}>
...
</BOUCLE_titre>
</BOUCLE_art>
```

sert à trouver les articles qui ont le même titre que l'article 5.

Attention : On ne peut utiliser qu'une balise simple. Il n'est pas permis de la filtrer ou de mettre du code optionnel.

Spécialement, si on veut utiliser la balise **#ENV** — ou tout autre balise prenant des paramètres —, on doit utiliser la notation : `{titre = #ENV{titre}}` et **pas** : `{titre = [(#ENV{titre})]}`.

Expressions régulières :

Très puissant (mais nettement plus complexe à manipuler), le terme de comparaison « == » introduit une comparaison selon une expression régulière. Par exemple :

```
<BOUCLE_art(ARTICLES){titre==^[aA]}>
```

sélectionne les articles dont le titre commence par « a » ou « A ».

Négation :

On peut utiliser la notation `{xxx != yyy}` et `{xxx !== yyy}`, le ! correspondant à la négation (opérateur logique NOT).

```
<BOUCLE_art(ARTICLES){id_secteur != 2}>
```

sélectionne les articles qui n'appartiennent pas au secteur numéro 2.

```
<BOUCLE_art(ARTICLES){titre!==^[aA]}>
```

sélectionne les articles dont le titre ne commence pas par « a » ou « A ».

Affichage en fonction de la date

Pour faciliter l'utilisation des comparaisons sur les dates, on a ajouté des critères :
- `age` et `age_redac` correspondent respectivement à l'ancienneté de la publication et de la première publication d'un article, en jours : `{age<30}` sélectionne les éléménts publiés depuis un mois ;
- les critères `mois`, `mois_redac`, `annee`, `annee_redac` permettent de comparer avec des valeurs fixes (`{annee<=2000}` pour les éléments publiés avant la fin de l'année 2000).

On peut combiner plusieurs de ces critères pour effectuer des sélections très précises. Par exemple :

```
<BOUCLE_art(ARTICLES){id_secteur=2}{id_rubrique!=3}{age<30}>
```

affiche les articles du secteur 2, à l'exclusion de ceux de la rubrique 3, et publiés depuis moins de 30 jours.

Astuce. Le critère `age` est très pratique pour afficher les articles ou les brèves dont la date est située « dans le futur », avec des valeurs négatives (à condition d'avoir sélectionné, dans la Configuration précise du site, l'option « Publier les articles post-datés »). Par exemple, ce critère permet de mettre en valeur des événements futurs. `{age<0}` sélectionne les articles ou les brèves dont la date est située dans le futur (« après » aujourd'hui)...

Âge par rapport à une date fixée. Le critère age est calculé par rapport à la date d'aujourd'hui (ainsi `{age<30}` correspond aux articles publiés depuis moins d'un mois par rapport à aujourd'hui). Le critère **`age_relatif`** compare la date d'un article ou d'une brève à une date « courante » ; par exemple, à l'intérieur d'une boucle ARTICLES, on connaît déjà une date pour chaque résultat de la boucle, on peut donc sélectionner par rapport à cette date (et non plus par rapport à aujourd'hui).

Par exemple :

```
<BOUCLE_article_principal(ARTICLES){id_article}>
      <h1>#TITRE</h1>
<BOUCLE_suivant(ARTICLES){id_rubrique}{age_relatif<=0}{exclus}{par
 date}{0,1}>
      Article suivant: #TITRE
      </BOUCLE_suivant>
</BOUCLE_article_principal>
```

la BOUCLE_suivant affiche un seul article de la même rubrique, classé par date, dont la date de publication est inférieure ou égale à la date de l'« article_principal » ; c'est-à-dire l'article de la même rubrique publié après l'article principal.

De plus amples informations sur l'utilisation des dates se trouvent dans l'article sur « La gestion des dates ».

Affichage d'une partie des résultats

- **`{branche}`** Limite les résultats — pour des boucles ayant un #ID_RUBRIQUE — à la branche actuelle (la rubrique actuelle et ses sous-rubriques). Par exemple :
`<BOUCLE_articles(ARTICLES) {branche}>` retournera tous les articles de la rubrique actuelle et de ces sous-rubriques,
`<BOUCLE_articles(ARTICLES) {!branche}>` retournera tous les articles qui ne sont pas dans la rubrique actuelle ou ses sous-rubriques,

On peut utiliser le critère **`{branche?}`** *optionnel* pour ne l'appliquer que si une rubrique est sélectionnée dans le contexte (une boucle englobante ou l'url fournie un id_rubrique). Par exemple :
`<BOUCLE_articles(ARTICLES) {branche?}>` retournera tous les articles de la rubrique actuelle et de ces sous-rubriques si il y a un id_rubrique dans le contexte, sinon, tous les articles du site.

- **`{doublons}`** ou **`{unique}`** (ces deux critères sont rigoureusement identiques) permettent d'interdire l'affichage des résultats déjà affichés dans d'autres boucles utilisant ce critère.

- **`{doublons`** *xxxx***`}`** on peut avoir plusieurs jeux de critères `{doublons}` indépendants. Les boucles ayant `{doublons rouge}` n'auront aucune incidence sur les boucles ayant `{doublons bleu}` comme critère.

- **`{exclus}`** permet d'exclure du résultat l'élément (article, brève, rubrique, etc.) dans lequel

on se trouve déjà. Par exemple, lorsque l'on affiche les articles contenus dans la même rubrique, on ne veut pas afficher un lien vers l'article dans lequel on se trouve déjà.

- **{***xxxx* **IN** *a,b,c,d***}** limite l'affichage aux résultats ayant le critère *xxxx* égal à *a*, *b*, *c* **ou** *d*. Les résultats sont triés dans l'ordre indiqué (sauf demande explicite d'un autre critère de tri). Il est aussi possible de sélectionner des chaînes de caractères, par exemple avec `{titre IN 'Chine', 'Japon'}`.

Les balises sont admises dans les arguments de IN, et notamment la balise `ENV`, à laquelle sont appliqués les filtres d'analyse pour assurer que la requête SQL sera bien écrite. De manière dérogatoire, SPIP testera si l'argument de `ENV` désigne un tableau (venant par exemple de saisies de formulaire dont l'attribut `name` se termine par `[]`). Si c'est le cas, et si les filtres d'analyse ont été désactivés en suffixant cette balise par une double étoile, alors chaque élément du tableau sera considéré comme argument de IN, SPIP appliquant les filtres de sécurité sur chacun d'eux.

Le squelette standard `formulaire_forum_previsu` fournit un exemple d'utilisation avec une boucle MOTS ayant le critère `{id_mot IN #ENV**{ajouter_mot}}` : cette boucle sélectionne seulement les mots-clés appartenant à un ensemble indiqué dynamiquement. Ici, cet ensemble aura été construit par le formulaire du squelette standard `choix_mots`, qui utilise des attributs `name=ajouter_mot[]`.

- `{a, b}` où `a` et `b` sont des nombres. Ce critère permet de limiter le nombre de résultats. *a* indique le résultat à partir duquel on commence l'affichage (attention, le premier résultat est numéroté 0 - zéro) ; *b* indique le nombre de résultats affichés.

Par exemple `{0, 10}` affiche les dix premiers résultats ; `{4, 2}` affiche les deux résultats à partir du cinquième (inclus).

- **{debut_***xxx***,***b***}** est une variante très élaborée de la précédente. Elle permet de faire commencer la limitation des résultats par une variable passée dans l'URL (cette variable remplace ainsi le *a* que l'on indiquait précédemment). C'est un fonctionnement un peu compliqué, que fort heureusement on n'a pas besoin d'utiliser trop souvent.

La variable passée dans l'URL commence forcément par `debut_`*xxx* (où *xxx* est un mot choisi par le webmestre) . Ainsi, pour une page dont l'URL est :

```
spip.php?page=petition&id_article=13&debut_signatures=200
```

avec un squelette (`petition.html`) contenant par exemple :

```
<BOUCLE_signatures(SIGNATURES){id_article}{debut_signatures,100}>
```

on obtiendra la liste des 100 signatures à partir de la 201-ième [rappel]. Avec l'URL :

```
spip.php?page=petition&id_article=13&debut_signatures=300
```

on obtient la liste des 100 signatures à partir de la 301-ième [rappel].

- `{a, n-b}` est une variante de `{a, b}` qui limite l'affichage en fonction du nombre de résultats dans la boucle. *a* est le résultat à partir duquel commencer à faire l'affichage ; *b* indique le nombre de résultats à **ne pas afficher** à la fin de la boucle.

`{0, n-10}` affichera tous les résultats de la boucle sauf les 10 derniers.

`{5, n}` affichera les résultats du 6ème au dernier.

Attention :
si « `a` » et « `b` » sont à remplacer par des nombres, c'est bien la **lettre** « `n` » qu'il faut utiliser quand nécessaire.

- `{n-a, b}` est le pendant de `{a, n-b}`. On limite à *b* résultats en commençant l'affichage au a^e résultat avant la fin de la boucle.

Par exemple : `{n-20,10}` affichera 10 résultats en partant du 20^e résultat avant la fin de la boucle.

- **{***a***/***b***}** où *a* et *b* sont des chiffres. Ce critère permet d'afficher une partie *a* (proportionnellement) des résultats en fonction d'un nombre de « tranches » *b*.

Par exemple : `{1/3}` affiche le premier tiers des résultats. Ce critère est surtout utile pour présenter des listes sur plusieurs colonnes. Pour obtenir un affichage sur deux colonnes, il suffit de créer une première boucle, affichée dans une case de tableau, avec le critère `{1/2}` (la première moitié des résultats), puis une seconde boucle dans une seconde case, avec le critère `{2/2}` (la seconde moitié des résultats).

Attention. L'utilisation du critère {doublons} avec ce critère est périlleuse. Par exemple :

```
<BOUCLE_prem(ARTICLES){id_rubrique}{1/2}{doublons}>
      <li> #TITRE
</BOUCLE_prem>
<BOUCLE_deux(ARTICLES){id_rubrique}{2/2}{doublons}>
      <li> #TITRE
</BOUCLE_deux>
```

n'affichera pas tous les articles de la rubrique ! Imaginons par exemple qu'il y ait au total 20 articles dans notre rubrique. La BOUCLE_prem va afficher la première moitié des articles, c'est-à-dire les 10 premiers, et interdire (à cause de `{doublons}`) de les réutiliser. La BOUCLE_deux, elle, va récupérer la deuxième moitié des articles de cette rubrique *qui n'ont pas encore été affichés* par la BOUCLE_prem ; donc, la moitié des 10 articles suivants, c'est-à-dire les 5 derniers articles de la rubrique. Vous avez donc « perdu » 5 articles dans l'opération...

Affichage *entre* les résultats

{"*inter***"}** permet d'indiquer un code HTML (ici, *inter*) inséré *entre* les résultats de la boucle. Par exemple, pour séparer une liste d'auteurs par une virgule, on indiquera :

```
<BOUCLE_auteurs(AUTEURS){id_article}{", "}>
```

Divers

{logo} permet de ne sélectionner que les articles (ou rubriques, etc) qui disposent d'un logo. Il fonctionne aussi dans la boucle (HIERARCHIE). Le critère inverse `{!logo}` liste les objets qui n'ont pas de logo.

Notes

[rappel] le premier résultat est numéroté 0, donc le 200e résultat représente réellement la 201 e signature

Les balises propres au site

Décembre 2001 — maj : Mars 2009

Les balises suivantes sont disponibles à n'importe quel endroit du squelette, même en dehors d'une boucle (hors « contexte »).

Balises définies à la configuration

Le contenu de ces balises est défini dans l'espace privé, lors de la configuration de votre site.

- **#NOM_SITE_SPIP** affiche le nom du site.

- **#URL_SITE_SPIP** affiche l'adresse du site. Elle ne comprend pas le / final, ainsi vous pouvez créer un lien du type `#URL_SITE_SPIP/sommaire.php`

- **#DESCRIPTIF_SITE_SPIP** affiche, comme son nom l'indique, le descriptif du site, que l'on renseigne dans la page de configuration générale du site.

- **#EMAIL_WEBMASTER** affiche l'adresse du webmestre. Par défaut, SPIP prend l'adresse de celui qui a installé le site (le premier administrateur).

- **#LOGO_SITE_SPIP** affiche le logo du site. Cette balise renvoie le logo du site 0. Il ne faut pas confondre avec le logo de la racine, aussi désigné sous le nom de logo standard des rubriques, c'est-à-dire celui de la rubrique 0.

- **#CHARSET** affiche le jeu de caractères utilisé par le site. Sa valeur par défaut est `iso-8859-1`, jeu de caractères dit « iso-latin »[4].

- **#LANG** : utilisée en dehors des boucles ARTICLES, RUBRIQUES, BREVES et AUTEURS, cette balise affiche la langue principale du site.

- **#LANG_DIR**, **#LANG_LEFT**, **#LANG_RIGHT** : ces balises définissent le sens d'écriture de la langue du contexte actuel (par exemple, de l'article qu'on est en train d'afficher). Voir l'article « Réaliser un site multilingue » pour plus d'information.

- **#MENU_LANG** (et **#MENU_LANG_ECRIRE**) : ces balises fabriquent et affichent un menu de langues permettant au visiteur d'obtenir la page en cours dans la langue choisie. 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).

Balises de mise en page

La balise **#CHEMIN{xxx}** donne le chemin complet vers le fichier `xxx`, qu'il se trouve à la racine, dans le dossier des squelettes, dans squelettes-dist/ etc.

On peut ainsi placer les fichiers « accessoires » (feuille de style, javascript, etc...) au squelette dans le répertoire **squelettes** et donc simplement distribuer ce dossier pour échanger ses

[4] Cf. www.uzine.net/article1785.html pour une introduction aux charsets, en attendant une documentation plus complète de cette fonctionnalité de SPIP.

squelettes. On écrira donc, par exemple, pour inclure une feuille de style du répertoire squelette :

```
<link rel="stylesheet" href="#CHEMIN{mon_style.css}" type="text/css"
/>
```

- **#PUCE** affiche devinez-quoi...

- **#FORMULAIRE_ADMIN** est une balise optionnelle qui permet de placer les boutons d'administration (« recalculer cette page », etc.) dans ses squelettes. Lorsqu'un administrateur parcours le site public, si cette balise est présente, elle sera remplacée par les boutons d'administration, sinon, les boutons seront placés à la fin de la page.

On peut aussi modifier la feuille de style *spip_admin.css* pour contrôler la position des boutons.

- **#DEBUT_SURLIGNE**, **#FIN_SURLIGNE** sont deux balises qui indiquent à SPIP dans quelle partie de la page colorer les mots clefs recherchés. Voir : « Les boucles et balises de recherche ».

- La balise **#INSERT_HEAD** doit se situer entre les balises `<head>` et `</head>` de vos squelettes. Elle permet à SPIP, ainsi qu'aux plugins éventuels, d'ajouter du contenu entre ces deux balises html.

Balises techniques

Attention, ces balises s'adressent à des utilisateurs avertis de SPIP.

- La balise **#REM** ne produit aucun affichage : elle permet de commenter le code des squelettes, de cette façon : `[(#REM) Ceci est un commentaire. ]`. Ces commentaires n'apparaissent pas dans le code généré pour le site public.

- **#SELF** retourne l'URL de la page appelée, nettoyée des variables propres à l'exécution de SPIP. Par exemple, pour une page avec l'url : `spip.php?article25&var_mode=recalcul` la balise #SELF retournera : `spip.php?article25`

Par exemple pour faire un formulaire :

```
<form action="#SELF" method="get">
```

Remarque : la balise `#SELF` représentant l'adresse de la page, elle n'est pas compatible avec les `<INCLURE()>` (sauf si le `$delais` de l'inclusion est mis à 0).

- **#URL_PAGE** retourne une url de type « page » (cf. les urls de spip), vers la page passée en paramètre et qui pourra être utilisée dans un lien. Par exemple, pour accéder à la page générée par le squelette `toto.html`, située dans votre dossier-squelette, `#URL_PAGE{toto}` généra automatiquement l'url `spip.php?page=toto`. Un second paramètre est autorisé pour ajouter des paramètres à l'url. Exemple `#URL_PAGE{toto,id_article=#ID_ARTICLE}` génèrera l'url `spip.php?page=toto&id_article=XXX`.

- **[(#ENV{*xxxx*,*zzzz*})]** permet d'accéder à la variable de nom *xxxx* passée par la requête HTTP. *zzzz* est une partie optionnelle qui permet de retourner une valeur même si la variable *xxxx* n'existe pas. On trouve une explication détaillée sur Spip-Contrib

Par défaut, la balise #ENV est filtrée par `htmlspecialchars`. Si on veut avoir le résultat brut, l'étoile « * » peut être utilisée comme pour les autres balises : `[(#ENV*{xxxx})]`.

Par exemple pour limiter la liste d'auteurs affichés :

```
<BOUCLE_auteurs(AUTEURS) {nom == #ENV{lettre,^A}}>
```

Retourne la liste d'auteur ayant le nom correspondant à l'expression régulière passé dans l'url par la variable lettre (`liste_auteur.php3?lettre=^Z`) ou les auteurs qui ont un nom commençant par un 'A' s'il n'y a pas de variable dans l'url.

- La balise **#SPIP_CRON** est liée à la gestion par SPIP des calculs qu'il doit faire périodiquement (statistiques, indexation pour le moteur de recherche, syndication de sites etc.).

Si cette balise n'est pas présente sur le site, le moteur de SPIP effectue ses calculs, en temps utile, après avoir envoyé une page à un visiteur ; malheureusement php ne permet pas de fermer la connexion à la fin de la page, et dans certains cas cela peut conduire certains visiteurs malchanceux (ceux dont le passage déclenche une procédure un peu longue, notamment la syndication) à constater une certaine lenteur dans l'affichage de la page demandée.

La balise `#SPIP_CRON` permet de contourner ce problème : son rôle est de générer un marqueur `<div>`

invisible dont la propriété « background » pointe sur le script `spip_background.php3` ; ce script à son tour effectue les calculs nécessaires « en tâche de fond », et renvoie une image transparente de 1×1 pixel. Cette astuce permet donc d'éviter tout sentiment de « ralentissement » en déportant les éventuelles lenteurs sur un script annexe.

A noter : cette balise n'est pas stratégique, et sa présence ou son absence ne modifient en rien la régularité du calcul des tâches périodiques du site.

- La balise **#SET{*variable*,*valeur*}** et son pendant **#GET{*variable*}** : La balise `#SET{xxx,yyy}` affecte une valeur yyy à une variable xxx **propre au squelette calculé**. Cette valeur peut être récupérée par la balise `#GET{xxx}`. Les variables créées ainsi ne sont pas transmises au squelette inclus.

Attention ! Si l'on affecte une valeur à une variable dans la partie facultative avant d'une boucle, il ne sera pas possible de récupérer cette valeur dans la boucle. Cela tient à la manière dont Spip calcule les squelettes.

- La balise **#HTTP_HEADER{*argument*}** permet de modifier l'entête HTTP de la page retournée par SPIP. Exemple : `#HTTP_HEADER{Content-Type: text/css}`. **Attention !** Le fait d'utiliser cette balise supprime les boutons d'administration. Cette balise ne peut pas être utilisée dans des squelettes inclus via la syntaxe `<INCLURE>`.

- La balise **#EVAL{*argument*}** évalue l'expression PHP mise en accolade. Par exemple `#EVAL{1+1}` affichera 2, `#EVAL{_DIR_IMG_PACK}` affichera ainsi le chemin vers le répertoire `ecrire/img_pack/`. Attention, il est fortement conseillé de s'en servir avec modération.

- La balise **#CACHE{*temps*}** permet de déterminer le délai au bout duquel le squelette est réinterprété. Le temps est exprimé en secondes. Il peut se mettre sous forme de calcul. Par exemple : `#CACHE{24*3600}`.

- La balise **#SQUELETTE** affiche le chemin du squelette courant.

- La balise **#VAL{*argument*}** retourne l'argument entre accolade. Par exemple `#VAL{toto}` retourne "toto".

Les formulaires

Août 2002 — maj : septembre 2010

SPIP permet une grande interaction du site avec les visiteurs ; pour cela, il propose de nombreux formulaires sur le site public, permettant tantôt de gérer les accès à l'espace privé, tantôt d'autoriser l'ajout de messages et signatures.

Les formulaires s'insèrent dans les squelettes par une simple balise ; SPIP se charge ensuite de gérer le comportement (souvent complexe) de ces formulaires en fonction de l'environnement et des configurations effectuées dans l'espace privé.

Fonctions interactives

- #FORMULAIRE_RECHERCHE

Il s'agit du formulaire du moteur de recherche intégré à SPIP. Il est présenté dans l'article sur les boucles de recherche.

- #FORMULAIRE_FORUM

Le `#FORMULAIRE_FORUM` gère l'interface permettant de poster des messages dans les forums publics. Il concerne donc en premier chef la boucle FORUMS mais peut être utilisé dans toutes les boucles acceptant un forum :

- La boucle ARTICLES,
- La boucle RUBRIQUES,
- La boucle BREVES,
- La boucle SITES (ou SYNDICATION).

Le formulaire dépend évidemment du choix des forums modérés a posteriori, a priori ou sur abonnement.

Par défaut, une fois le message posté, le visiteur est renvoyé vers la page de l'élément[5] [1] auquel il a répondu. On peut décider de renvoyer le visiteur vers une autre page en passant une url en paramètre à cette balise. Par exemple :

- `[(#FORMULAIRE_FORUM{'spip.php?page=merci'})]` renverra vers la page *spip ?page=merci*.

- `[(#FORMULAIRE_FORUM{#SELF})]` renverra vers la page où le formulaire de forum est placé (voir la balise #SELF).

Dans le cas (très spécifique) où l'on a autorisé la présence de mots-clés dans les forums publics, on peut affiner le comportement de ce formulaire avec des variables de personnalisation.

[5] article, rubrique, brève, site ou forum

- #FORMULAIRE_SIGNATURE

La balise `#FORMULAIRE_SIGNATURE` affiche un formulaire permettant aux visiteurs du site de signer les pétitions associées aux articles. Cette balise se place donc dans une boucle ARTICLES.

La signature des pétitions réclame obligatoirement une validation des signataires par email. Ce formulaire n'a donc d'intérêt que si votre hébergeur autorise l'envoi de mails par PHP.

- #FORMULAIRE_SITE

La balise `#FORMULAIRE_SITE` affiche un formulaire permettant aux visiteurs du site de proposer des référencements de sites. Ces sites apparaîtront comme « proposés » dans l'espace privé, en attendant une validation par les administrateurs.

Ce formulaire ne s'affiche que si vous avez activé l'option « Gérer un annuaire de sites » dans la *Configuration sur site* dans l'espace privé, et si vous avez réglé « Qui peut proposer des sites référencés » sur « les visiteurs du site public ».

Les sites référencés étant, dans SPIP, attachés aux rubriques, on ne peut placer ce `#FORMULAIRE_SITE` qu'à l'intérieur d'une boucle RUBRIQUES.

- #FORMULAIRE_ECRIRE_AUTEUR

Placée à l'intérieur d'une boucle AUTEURS, cette balise affiche le formulaire qui permet d'envoyer un mail à l'auteur. Cela permet d'écrire aux auteurs sans divulguer leur adresse email sur le site public.

Placé dans une boucle ARTICLES, ce formulaire permet d'envoyer un mail à tous les auteurs de cet article.

Placé dans une boucle FORUMS, ce formulaire permet d'envoyer un mail directement à l'auteur du message si l'auteur est enregistré sur le site.

Inscription, authentification...

- #FORMULAIRE_INSCRIPTION

Sans doute le plus importante, la balise `#FORMULAIRE_INSCRIPTION` affiche le formulaire permettant l'inscription de nouveaux rédacteurs. Celui-ci ne s'affiche que si vous avez autorisé l'inscription automatique depuis le site public (sinon, cette balise n'affiche rigoureusement rien).

L'inscription nécessite l'envoi des informations de connexion (login et mot de passe) par email ; ce formulaire ne fonctionne donc que si votre hébergeur autorise l'envoi de mails par PHP.

- [(#FORMULAIRE_INSCRIPTION{forum})]

Est l'équivalente de la précédente, pour l'inscription des visiteurs, appelés à écrire dans les forums (réservés aux visiteurs enregistrés), option qui se détermine dans la partie privée

configuration/interactivité/Mode de fonctionnement par défaut des forums publics. Le paramètre « forum » correspond à l'intitulé du statut d'auteur désiré lors de l'inscription.

Après la validation, un message avertit le visiteur : "Votre nouvel identifiant vient de vous être envoyé par email."

- #LOGIN_PRIVE

Tout aussi importante (sinon plus), cette balise affiche le formulaire d'accès à l'espace privé (la partie « /ecrire » du site).

Important : cette balise doit impérativement être présente dans le squelette appelé par la page `spip.php?page=login`, c'est-à-dire dans le squelette nommé `login.html`. En effet, lors des accès directs à l'adresse « /ecrire » de votre site, c'est vers `spip.php?page=login` que SPIP va vous rediriger.

- #LOGIN_PUBLIC

D'une utilisation beaucoup plus spécifique, la balise `#LOGIN_PUBLIC` affiche un formulaire permettant à vos utilisateurs de s'identifier tout en restant sur le site public (sans entrer dans l'espace privé). Cette balise sert notamment à authentifier les visiteurs pour les sites proposant des forums *modérés sur abonnement*. Elle peut aussi servir de brique de base pour restreindre l'accès à certains contenus sur le site public : mais cela reste d'un maniement complexe, et nécessitera encore des développements et la rédaction de tutoriels complets avant d'être facilement utilisable par tous ; néanmoins, un exemple d'utilisation avancée est donné plus bas.

Le `#LOGIN_PUBLIC`, par défaut, « boucle sur lui-même », c'est-à-dire que le formulaire revient sur la page où il se trouve. On peut cependant indiquer une page vers laquelle le formulaire mènera, sous la forme :

```
[(#LOGIN_PUBLIC{#URL_PAGE{mapage}})]
```

Si votre site offre une inscription automatique à l'espace privé, les données de connexion à l'espace public sont identiques à celles de l'espace privé ; c'est-à-dire que les données envoyées à l'utilisateur pour s'identifier à l'espace public lui permettent également d'accéder à l'espace privé. Si, au contraire, vous avez interdit l'inscription automatique à l'espace privé, *il faut impérativement avoir au moins un article dont les forums seront réglés en mode « sur abonnement »* pour activer cette balise ; dès lors, SPIP pourra fournir des informations de connexion pour le site public sans accès à l'espace privé.

- **#URL_LOGOUT** est le pendant de `#LOGIN_PUBLIC` ; il donne une URL permettant à un visiteur authentifié de se déconnecter.
Noter que `#URL_LOGOUT` étant une balise *dynamique*, elle renverra *toujours* quelque chose de sorte que les parties conditionnelles (La syntaxe des balises SPIP) de la balise seront toujours affichées. *Ainsi une notation du type `[<a href="(#URL_LOGOUT)">déconnexion</a>]` retournera le source html `<a href="">déconnexion</a>` lorsque le visiteur n'est pas connecté. Pour un affichage conditionnel du lien de déconnexion, voir plus bas.*

On peut passer un paramètre à cette balise pour spécifier l'adresse de retour après la déconnexion. Par exemple `[(#URL_LOGOUT{spip.php?page=sommaire})]` renverra vers la page de sommaire.

Voici un exemple simple, mais complet, d'utilisation de ces deux balises. Il faut passer par un peu de php pour tester la variable `$auteur_session`, qui indique qu'un auteur est identifié ou non. Si c'est le cas, on peut récupérer (voire tester) son statut, son login, etc., via `$auteur_session['statut']`....

Notez bien que le contenu n'est « sécurisé » que sur ce squelette. Si votre squelette « imprimer cet article », par exemple, ne vérifie pas `$auteur_session`, tout le monde (y compris les moteurs de recherche !) pourra avoir accès à ce fameux contenu que vous souhaitez protéger.

```
<?php if ($auteur_session) { ?>
Vous êtes authentifié,
<a href='#URL_LOGOUT'>cliquez ici pour vous  déconnecter</a>
... ici le contenu en accès restreint....
<?php } else { ?>
<h2>Cette partie est en accès restreint</h2>
#LOGIN_PUBLIC
<?php } ?>
```

Depuis SPIP 2.0, il n'est plus utile d'utiliser du code php dans votre squelette ; vous pouvez désormais écrire :

```
[(#LOGIN_PUBLIC|non)
  [<a href="(#SESSION|oui)
    #URL_LOGOUT">déconnexion</a>
  ]
]
```

pour afficher au choix le formulaire de login ou le lien de déconnexion en fonction de l'état (identifié/pas identifié) du visiteur.

Styles

On peut sensiblement modifier l'interface graphique des formulaires par l'intermédiaire des feuilles de style. Voir : « Ils sont beaux, mes formulaires ! ».

Les filtres de SPIP

Mai 2001 — maj : octobre 2010

Nous avons vu dans la syntaxe des balises SPIP qu'il était possible de modifier le comportement et l'affichage des balises en leur attribuant des filtres.

```
[ option avant (#BALISE|filtre1|filtre2|...|filtren) option après ]
```

Les filtres 1, 2, ..., n sont appliqués successivement à la `#BALISE`.

Les filtres de mise en page

Les filtres de mise en page suivants (majuscules, justifier...) ne sont plus conseillés. Il est recommandé de leur préférer, désormais, l'utilisation des styles CSS correspondants.

- **majuscules** fait passer le texte en majuscules. Par rapport à la fonction de PHP, *majuscules* s'applique également aux lettres accentuées.

- **justifier** fait passer le texte en justification totale (`<P align=justify>`).

- **aligner_droite** fait passer le texte en justification à droite (`<P align=right>`).

- **aligner_gauche** fait passer le texte en justification à gauche (`<P align=left>`).

- **centrer** centre le texte (`<P align=center>`).

Les filtres des dates

Les filtres suivants s'appliquent aux dates (`[(#DATE|affdate)]` par exemple).

- **affdate** affiche la date sous forme de texte, par exemple « 13 janvier 2001 ».

On peut lui passer un paramètre de formatage de la date, correspondant à un format spip (« `'saison'` », etc.) ou à un format de la commande php date (« `'Y-m-d'` »). Par exemple :

- `[(#DATE|affdate{'Y-m'})]` affichera numériquement l'année et le mois de la date filtrée séparés par un tiret,
- la notation `[(#DATE|affdate{'saison'})]` est totalement équivalente à : `[(#DATE|saison)]`.

- Il existe aussi des variantes de *affdate* qui fournissent des raccourcis :

affdate_jourcourt affiche le nom du mois et la valeur numérique du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors l'année est aussi affichée : « 1 Novembre 2004 ».

affdate_court affiche le nom du mois et le numéros du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors on affiche seulement le mois et l'année sans le numéros du jour : « Novembre 2004 ».

affdate_mois_annee affiche seulement le mois et l'année : « Avril 2005 », « Novembre 2003 ».

- **jour** affiche le jour (en nombre).

- **mois** affiche le mois (en nombre).

- **annee** affiche l'année.

- **heures** affiche les heures d'une date (les dates fournies par SPIP contiennent non seulement le jour, mais également les horaires).

- **minutes** affiche les minutes d'une date.

- **secondes** affiche les secondes.

- **nom_jour** affiche le nom du jour (lundi, mardi...).

- **nom_mois** affiche le nom du mois (janvier, février...).

- **saison** affiche la saison (hiver, été...).

- Le filtre **unique** retourne la valeur de l'élément filtré seulement si c'est la première fois qu'elle est rencontrée. Ce filtre n'est pas limité aux dates mais est intéressant pour, par exemple, afficher une liste d'articles par date :

```
<BOUCLE_blog(ARTICLES){par date}{inverse}{"<br>"}>
      [<hr /><h1>(#DATE|affdate_mois_annee|unique)</h1>]
      #TITRE ...
</BOUCLE_blog>
```

cette balise n'affichera la date qu'à chaque changement de mois.

Voici un autre exemple :

```
<BOUCLE_blog2(ARTICLES){par date}{inverse}>
    [<hr /><h1>(#DATE|annee|unique)</h1>]
        [<h2>(#DATE|affdate{'Y-m'}|unique|nom_mois)</h2>]
             <a href="#URL_ARTICLE">#TITRE</a><br />
</BOUCLE_blog2>
```

affichera une liste ressemblant à :

```
2005
     mars
             article de mars
             autre article de mars
     février
             article de février
2004
     décembre
             un article
```

On utilise la notation `affdate{'Y-m'}` pour afficher le nom du mois à chaque année. En effet :

- si l'on ne faisait que `#DATE|nom_mois|unique`, les noms de mois ne seraient affichés que la première année.
- si le filtrage était : `#DATE|unique|nom_mois`, on afficherait toutes les dates. En effet, `#DATE` retourne une date complète qui contient aussi l'heure. Il y a donc une grande chance que les dates complètes de deux articles publiés le même jours soient différentes.

C'est pourquoi on garde juste le mois et l'année de la date avant de la passer au filtre *unique*.

On peut passer un argument optionnel à ce filtre pour différencier deux utilisations indépendantes du filtre. Par exemple : `[(#DATE|affdate_mois_annee|unique{ici})]` n'aura pas d'incidence sur `[(#DATE|affdate_mois_annee|unique{la})]`.

Filtres de texte

- **liens_ouvrants** transforme les liens SPIP qui donnent vers des sites extérieurs en liens de type « popup », qui ouvrent dans une nouvelle fenetre ; c'est l'équivalent du *target=blank* du HTML.
N.B. : les développeurs de SPIP estiment qu'il s'agit en général d'une impolitesse, car les internautes savent très bien s'ils ont envie ou pas d'ouvrir une nouvelle fenêtre - or ce système le leur impose. Mais la demande était trop forte, et nous avons craqué ;-)

- **supprimer_numero** sert à éliminer le numéro d'un titre, si par exemple on veut faire des tris d'articles `{par num titre}` mais ne pas afficher les numéros (car ils ne servent qu'à ordonner les articles). Le format des préfixes numerotés est « XX. titre », XX étant un nombre à n chiffres (illimité).

- **PtoBR** transforme les sauts de paragraphe en simples passages a la ligne, ce qui permet de « resserrer » une mise en page, par exemple à l'intérieur d'un sommaire

- **taille_en_octets** permet de transformer un nombre d'octets (25678906) en une chaîne de caractères plus explicite (« 24.4 Mo »).

- **supprimer_tags** est une suppression basique et brutale de tous les `<...>`

- **textebrut** s'apparente au filtre `supprimer_tags`, mais il agit de manière un peu plus subtile, transformant notamment les paragraphes et `<br>` en sauts de ligne, et les espaces insécables en espaces simples. Il s'utilise, par exemple, pour faire un descriptif META :
`[<meta name="description" content="(#DESCRIPTIF|textebrut)">]`

- **texte_backend** peut être utilisé pour transformer un texte et le rendre compatible avec des flux XML. Ce filtre est utilisé, par exemple, dans le squelette *backend.html* qui génére le fil RSS du site.

- **couper** coupe un texte après un certain nombre de caractères. Il essaie de ne pas couper les mots et enlève le formatage du texte. Si le texte est trop long, alors « (...) » est ajouté à la fin. Ce filtre coupe par défaut à 50 caractères, mais on peut spécifier une autre longueur en passant un paramètre au filtre, par exemple : `[(#TEXTE|couper{80})]`.

- **lignes_longues** coupe les mots « trops longs » (utile si l'on a, par exemple, des urls à afficher dans une colone étroite). Ce filtre coupe par défaut à 70 caractères, mais on peut

spécifier une autre longueur en passant un paramètre au filtre, par exemple : `[(#URL_SITE|lignes_longues{40})]`.

- **match** utilise une expression régulière (cf. *preg_match()*) pour extraire un motif dans le texte si il est présent, ne retourne rien sinon. Par exemple pour récupérer le premier mot du titre `[(#TITRE|match{^\w+?})]`. Ce peut être un texte simple, afficher "toto" si dans le titre : `[(#TITRE|match{toto})]`

- **replace** utilise aussi une expression régulière (cf. *preg_replace()*) pour supprimer ou remplacer toutes les occurences d'un motif dans le texte. Avec un seul parametre, une expression régulière, le motif sera remplacé par une chaîne, c'est à dire supprimé. Par exemple pour supprimer tous les "notaXX" du texte `[(#TEXTE|replace{nota\d*})]`. Lorsqu'un deuxiéme paramètre est fourni, les occurences du motif seront remplacées par cette valeur. Par exemple pour remplacer tous les 2005 ou 2006 du texte en 2007 `[(#TEXTE|replace{200[56],2007})]`. Ce peut être des textes simples, remplacer tous les "au temps" par "autant" : `[(#TEXTE|replace{au temps,autant})]`

Filtres de test

- Le filtre `|sinon`, qui indique ce qu'il faut afficher si l'élément « filtré » est vide : ainsi `[(#TEXTE|sinon{"pas de texte"})]` affiche le texte ; si celui-ci est vide, affiche *« pas de texte »*.

- Le filtre **`|?{`***`sioui`***`,`***`sinon`***`}`** est une version évoluée de `|sinon`. Il prend un ou deux paramètres :

  - *sioui* est la valeur à afficher à la place de l'élément filtré si celui-ci est non vide.
  - *sinon* est optionnel. C'est la valeur à afficher si l'élément filtré est vide. `[(#TEXTE|?{#TEXTE,"pas de texte"})]` est équivalent à l'exemple donné pour le filtre `|sinon`.

- Jeu de filtres pour faire des comparaisons avec des valeurs :

  - **`|=={`***`valeur`***`}`** et **`|!={`***`valeur`***`}`** permettent de vérifier, respectivement, l'égalité ou l'inégalité entre l'élément filtré et *valeur*. Par exemple : `<li [(#TITRE|=={édito}|?{'id="edito"',''})]>#TITRE</li>`

  - **`|>{`***`valeur`***`}`**, **`|>={`***`valeur`***`}`**, **`|<{`***`valeur`***`}`** et **`|<={`***`valeur`***`}`** comparent l'élément filtré (qui doit être numérique) avec une valeur numérique.
    Par exemple :

    [(#TOTAL_BOUCLE)
    [(#TOTAL_BOUCLE|>{1}|?{'articles','article'})] dans cette rubrique.]

Remarque : De manière générale, tous les opérateurs de comparaison de php peuvent être utilisés comme filtres.

Filtres de logos

- **fichier** Affecté à un logo, ce filtre permet de récupérer directement le nom de fichier correspondant au logo.

- ||**autres filtres** (jusqu'à SPIP 2) permet de passer des filtres « maison » sur les logos, la logique est un peu tordue. L'analyse se déroule comme suit :

- si le premier « filtre » n'est pas un alignement, SPIP considère qu'il s'agit d'un URL et fait un lien du logo vers cette adresse ;
- si le premier filtre est un alignement, SPIP considère que le deuxième « filtre » est un URL ;
- les filtres suivants sont de vrais filtres au sens habituel (y compris des filtres « maison » déclarés dans mes_fonctions.php ;
- pour appliquer un filtre quelconque sans mettre d'URL, il faut mettre deux barres. Par exemple : `<?php $logo = '[(#LOGO_RUBRIQUE||texte_script)]'; ?>` permet de récupérer le logo dans la variable php `$logo`, pour traitement ultérieur (voir ci-dessous pour la signification de `|texte_script`).

Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

- Les filtres **`hauteur`** et **`largeur`** retournent les informations sur la taille (i.e. Hauteur et Largeur) de l'élément filtré si c'est une image.

Ces filtres n'ont qu'un intérêt moyen à être appliqués directement à un logo de document puisqu'il y a déjà #HAUTEUR et #LARGEUR à disposition pour les documents. Par contre, on peut les appliquer après le filtre `image_reduire` pour connaître la taille exacte de l'image réduite.

Plus généralement, on peut les appliquer sur n'importe quelles balises (ou filtre) retournant un balise HTML `<img ...>`.

- **`|image_reduire{largeur,hauteur}`** permet de forcer une taille maximale d'affichage des images et des logos.

Ce filtre s'utilise par exemple sur un logo d'article de la façon suivante :

```
[(#LOGO_ARTICLE|right||image_reduire{130})]
```

Dans cet exemple, le logo de l'article apparaît aligné à droite, à une taille maximale de 130 pixels.

Ce filtre peut prendre deux arguments : *largeur* et *hauteur*. Si l'un de ces deux arguments est égal à 0, SPIP ne tient compte que de l'autre et calcule cette dimension en conservant les proportions de l'image. De plus, ce filtre s'applique aussi à la balise **`#TEXTE`** et ce sont alors toutes les images que le rédacteur introduit dans le texte grâce aux raccourcis SPIP qui sont alors réduites.

Ainsi, par exemple :

```
[(#TEXTE|image_reduire{600,0})]
```

affiche toutes les images insérées dans le fil du texte à une largeur maximale de 600 pixels. Cela permet de préserver la mise en page même sans que le rédacteur ait à se soucier de la taille des images qu'il télécharge sur le site.

NB. : Si l'option « création de vignettes » est activée dans la configuration du site, ces logos réduits seront des fichiers d'images spécifiques calculés automatiquement par le serveur (idéalement, avec l'extension GD2 installée sur le serveur), pour les formats acceptés par le serveur (avec GD2, habituellement, les formats JPG et PNG). Sinon, c'est une version complète de l'image qui est affichée, mais avec une taille d'affichage fixée directement en HTML.

Les filtres mathématiques

SPIP 1.9 introduit une série de filtres d'opérations mathématiques.

- **`|plus{xx}`**, **`|moins{xx}`** et **`|mult{xx}`** correspondent respectivement à l'addition, la soustraction et la multiplication.

- **`|div{xx}`** correspond à la division *non-euclidienne* ("après la virgule").

- **`|modulo{xx}`** correspond au reste de la division euclidienne par xx d'un nombre.

Par exemple `[(#COMPTEUR_BOUCLE|modulo{5})]` compte de 0 à 4 puis revient à 0 etc

De plus l'ensemble des fonctions mathématiques PHP peuvent être utilisées comme filtres.

Autres Filtres

- **traduire_nom_langue** s'applique à la balise #LANG et retourne un traduction du code de langue qu'elle retourne (fr, en, it, etc.) dans cette langue.

Remarque : Les traductions des codes sont faites dans la langue que représente ce code et suivent les conventions d'écriture de cette langue.

Ainsi « fr » sera traduit « français » en minuscule, alors que « es » sera traduit « Español » avec une majuscule.

- **alterner{*a,b,c,...*}** s'applique à une balise numérique (en général #COMPTEUR_BOUCLE ou #TOTAL_BOUCLE) et affiche l'argument correspondant à la valeur de cette balise . On peut ainsi alterner un affichage dans une boucle. Par exemple, `[(#COMPTEUR_BOUCLE|alterner{'white','yellow'})]` affichera « white » à la première itération de la boucle, « yellow » à la deuxième, « white » à la troisième, « yellow » à la quatrième, etc. Ainsi, on peut faire une liste d'article qui utilise une couleur différente pour les lignes paires et impaires :

```
<B_lesarticles>
      <ul>
<BOUCLE_lesarticles(ARTICLES) {par titre}>
      <li style="background:
      [(#COMPTEUR_BOUCLE|alterner{'white','yellow'})]">#TITRE</li>
</BOUCLE_lesarticles>
      </ul>
</B_lesarticles>
```

- **inserer_attribut{*attribut,valeur*}** permet d'ajouter un attribut html dans une balise html générée par SPIP. Par exemple :
`[(#LOGO_DOCUMENT||inserer_attribut{'alt',#TITRE})]` va ajouter un attribut « alt » avec le titre du document dans la balise « img » du logo.

- **extraire_attribut{*attribut*}** est l'inverse du filtre précédent. Il permet de récupérer un attribut d'une balise html générée par SPIP. Par exemple, on peut trouver le chemin de la vignette générée par le filtre

- **vider_attribut{*attribut*}** est une variante de inserer_attribut. Il permet de supprimer les attribut html. Exemple `[(#LOGO||vider_attribut{width})]` enlever l'attribut 'width' sur l'image d'un logo (quelle idée !)

image_reduire : `<div style="background: url([(#LOGO_ARTICLE||image_reduire{90}|extraire_attribut{src})]) left;"]>#TEXTE</div>`

- **parametre_url{*parametre,valeur*}** est un filtre qui permet d'ajouter des paramètres dans une url générée par une balise SPIP. Si *valeur* vaut `''` le filtre supprime un paramètre actuellement dans l'url. Par exemple, la balise `#SELF` retourne l'url de la page actuelle, donc :

- `[(#SELF|parametre_url{'id_article','12'})]` placera une variable id_article égale à 12 dans l'url,
- `[(#SELF|parametre_url{'id_article',''})]` effacera l'id_article actuellement dans l'url.

Employé avec un seul paramètre le filtre agit différemment : `#URL...|parametre_url{x}` extrait le parametre 'x' de l'url et en retourne la valeur.

On peut par exemple l'utiliser pour faire des boutons pour naviguer parmi les documents sur une page :

```
<BOUCLE_actuel(DOCUMENTS) {id_document}>
  #LOGO_DOCUMENT
  <ul>
  <BOUCLE_precede(DOCUMENTS) {par date} {age_relatif <= 0} {0,1} {exclus}>
  <li>
    <a href="[(#SELF|parametre_url{'id_document',#ID_DOCUMENT})]"
           title="précédent">
      [(#LOGO_DOCUMENT||image_reduire{70})]
    </a>
  </li>
  </BOUCLE_precede>
  <BOUCLE_suivant(DOCUMENTS) {par date} {age_relatif > 0}{0,1}>
  <li>
    <a href="[(#SELF|parametre_url{'id_document',#ID_DOCUMENT})]"
           title="suivant">
      [(#LOGO_DOCUMENT||image_reduire{70})]
    </a>
  </li>
  </BOUCLE_suivant>
  </ul>
</BOUCLE_actuel>
```

ancre_url{*ancre*} ajoute un modifie une ancre à une url. Par exemple `[(#URL_ARTICLE|ancre_url{ancre})]`.

Filtres techniques

- **entites_html** transforme un texte en entités HTML, que l'on peut donc implanter dans un formulaire, exemple : `[<textarea>(#DESCRIPTIF|entites_html)</textarea>]`

- **texte_script** transforme n'importe quel champ en une chaîne utilisable en PHP ou Javascript en toute securité, exemple : `<?php $x = '[(#TEXTE|texte_script)]'; ?>`. Attention : utilisez bien le caractère `'` et non `"` : en effet, dans le second cas, si votre texte

contient le symbole $, le résultat peut être catastrophique (affichage partiel, affichage d'autre chose, plantage php, etc.).

- **attribut_html** rend une chaîne utilisable sans dommage comme attribut HTML ; par exemple, si l'on veut ajouter un texte de survol au lien normal vers un article, on utilisera `<a href="#URL_ARTICLE"[ title="(#DESCRIPTIF|attribut_html|couper{80})"]>#TITRE</a>`.

- **liens_absolus** s'applique sur une balise de texte et transforme tous les liens que celui-ci contient en liens absolus, en préfixant le protocole (http ou https) et le nom d'hôte courants. Ce filtre est particulièrement utile dans des squelettes de fil rss par exemple. Pour forcer une certaine URL de base, il est possible de la passer en argument, par exemple `#TEXTE|liens_absolus{#URL_SITE_SPIP}`

- **url_absolue** marche de la même façon que le filtre précédent, mais s'applique à une balise qui retourne une url (par exemple #URL_ARTICLE). Tout comme le filtre `liens_absolus`, ce filtre accepte une URL de base comme paramètre optionnel.

- **abs_url** combine les deux balises précédentes et peut donc s'appliquer à un texte ou à une balise d'url. Il accepte le même paramètre optionnel.

- **form_hidden** Si on fait un formulaire qui utilise comme action un lien comprenant des arguments (par exemple, quand on utilise la balise `#SELF` avec le type d'url par defaut), il faut remettre ces valeurs dans des champs *hidden* ; cette fonction calcule les champs en question. A utiliser par exemple :

```
<form action="#SELF">
[(#SELF|form_hidden)]
...
</form>
```

- Le filtre **|compacte** permet de réduire la taille d'un CSS ou d'un javascript en supprimant tout les commentaires. Le filtre prend en entrée le nom du fichier, et produit un nouveau fichier dont il renvoie le nom `<link rel="stylesheet" href="[(#CHEMIN{spip_style.css}|compacte)]" type="text/css" media="all" />`.

- Un nombre d'autres filtres techniques sont référencés dans le chapitre "filtres".

Ajouter ses propres fonctions

Les filtres de SPIP sont des fonctions PHP qui reçoivent la balise sur laquelle ils sont appliqués en premier paramètre et retournent le texte à afficher. Vous pouvez utiliser directement les fonctions habituelles de PHP, mais également créer les vôtres, sur le modèle :

```
<?php
function mon_filtre($texte){
   $texte = (bidouillages en PHP) ...;
   return $texte;
}
?>
```

Afin de ne pas avoir à modifier des fichiers de SPIP (qui risqueraient d'être écrasés lors d'une prochaine mise à jour), vous pouvez installer vos fonctions personnelles dans un fichier `mes_fonctions.php` : si SPIP repère un fichier ayant ce nom, il l'inclut automatiquement.

Il peut se situer :

- dans le dossier où sont stockés vos squelettes
- à la racine du site Mais **jamais** dans les deux à la fois.

Il est possible de définir des filtres *applicables uniquement à un squelette particulier*.
Par exemple, on créera, dans un fichier `article_fonctions.php`, des filtres qui ne seront effectifs que dans le squelette `article.html`.
attention : ce fichier `xxxx_fonctions.php` devra se trouver au même niveau (dans le même répertoire) que le squelette `xxxx.html`.

Filtres avec des paramètres

Il est possible de passer des paramètres dans les filtres. La syntaxe est :

```
[(#BALISE|filtre{arg1, arg2}|...)]
```

Le filtre doit être défini de la manière suivante dans mes_fonctions.php :

```
function filtre($texte, $arg1='valeur par defaut1',

$arg2='valeur par défaut 2')
{
      ....calculs....
      return (une chaine de caractères);
}
```

On peut ainsi appeler n'importe quelle fonction php, ou s'appuyer sur des fonctions définies dans SPIP ou dans `mes_fonctions.php`, pour peu qu'elles respectent l'ordre des arguments (le texte à traiter doit être impérativement le premier argument). Par exemple, pour enlever les points à la fin d'un texte, on pourra faire : `[(#TEXTE|rtrim{'.?!'})]`.

Les arguments des filtres peuvent être des balises (sans codes optionnels ni filtres). Par exemple :

```
[(#TOTAL_BOUCLE|=={#COMPTEUR_BOUCLE}|?{'Fin.',''})]
```

On peut mettre un balise avec notation étendue en paramètre. Par exemple :

```
[(#DESCRIPTIF|sinon{[(#CHAPO|sinon{#TEXTE}|couper{300})]})]
```

Les boucles récursives

Mai 2001 — maj : Février 2009

Les boucles récursives offrent une fonctionnalité très puissante de mise en page de structure hiérarchique. Leur écriture est concise, mais leur utilisation demande une bonne maîtrise logique de l'enchaînement des boucles.

Pour construire une boucle récursive, il suffit d'indiquer dans son `TYPE` le nom d'une boucle contenant celle qu'on écrit :

```
<BOUCLEx ....>
....
<BOUCLEn(BOUCLEx)></BOUCLEn>
....
</BOUCLEx>
```

La boucle *n* fonctionne comme si l'on avait recopié l'intégralité de la boucle *x* (toutes les balises et le code HTML, ainsi que les textes conditionnels avant, après et alternatif) à l'endroit où l'on insère la boucle *n*. La boucle *n* étant à l'intérieur de la boucle *x*, on obtient un comportement récursif : la boucle *x* contient une boucle *n*, qui elle-même reproduit la boucle *x* qui contient la boucle *n*, et ainsi de suite, jusqu'à ce que la boucle *x* ne donne plus aucun résultat. Aucun critère ne figure dans la boucle *n*, le changement de contexte à chaque appel de la boucle *x* devant conduire les critères de celle-ci à ne plus trouver aucun élément.

Cette technique permet de créer notamment l'affichage des threads des forums. Une première boucle « fabrique » l'entrée des threads (les messages qui répondent directement à un article), une seconde boucle affiche les réponses à ces messages, et une boucle récursive provoque la récursivité sur cette seconde boucle :

```
<BOUCLE_forum(FORUMS){id_article}>
   #TITRE
   <B_reponses>
   <UL>
   <BOUCLE_reponses(FORUMS){id_parent}>
      <LI>#TITRE
      <BOUCLE_recursive(BOUCLE_reponses)>
      </BOUCLE_recursive>
      </LI>
   </BOUCLE_reponses>
   </UL>
   </B_reponses>
</BOUCLE_forum>
```

Plus généralement, cette fonctionnalité provoque un affichage graphiquement très clair de structures arborescentes, en particulier la hiérarchie des rubriques de votre site.

Remarque 1 : Le compilateur considère que la notation `<BOUCLEn(BOUCLEx)>` à l'extérieur de la boucle *x* est vide de sens, ce qui lui permet d'atteindre l'optimalité du nombre de champs des requêtes SQL qu'il produit, pour tout type de boucles.
Cet emploi n'est pas une récursion mais une inclusion, fonctionnalité à laquelle répond la balise INCLURE, nettement préférable.

Remarque 2 : Dans l'état actuel du compilateur de SPIP, la séquence `<BOUCLEn(BOUCLEx)></BOUCLEn>` doit figurer au premier niveau de la boucle *x*, autrement dit la boucle *n* doit être immédiatement englobée par la boucle *x*, non par une autre boucle elle-même à l'intérieur de la boucle *x*.
La levée de cette restriction est à l'étude.

Remarque 3 : On peut appeler une balise homonyme de l'une des boucles englobantes en explicitant le nom de la boucle à laquelle la balise appartient. Il faut alors spécifier le nom de la boucle entre le # et le nom de la balise. Plus de détail dans l'article la syntaxe des balises SPIP.

On écrira alors la balise `#BALISE` de la boucle `_boucle` [1] de la façon suivante : `#_boucle:BALISE`.

La gestion des dates

Mai 2003 — maj : Novembre 2008

Voici une liste de critères et de balises pour mieux gérer les dates des articles.

Afficher les dates

- **#DATE** est la date de mise en ligne. (Modifiable après la mise en ligne de l'article, de la brève, etc. La date d'une rubrique est celle de son élément le plus récent.)

- **#DATE_REDAC** est la date de première publication. (Modifiable à volonté, disponible sur les articles seulement.)

- **#DATE_MODIF** cette balise désigne la date de dernière modification de l'article.

- **#DATE_NOUVEAUTES** permet d'afficher la date du dernier envoi du mail présentant les nouveautés.

Formater les dates

Si les balises `#DATE...` sont utilisées sans filtre, alors toutes les informations de date sont affichées dans un format numérique (au format MySQL) : « `2001-12-01 03:25:02` ».

Les filtres `|annee`, `|mois`, `|jour`, `|heures`, `|minutes`, `|secondes`, mais aussi `|affdate`, `|date_relative`, `|nom_mois`, `|nom_jour`, `|saison`, etc. s'appliquent pour permettre tous les affichages habituels sous divers formats. Une liste complète des filtres pouvant être appliqués aux dates est fournie dans l'article les filtres de SPIP.

Contexte de date

SPIP fournit à toutes les boucles un contexte de date. Si l'on se trouve à l'intérieur d'une boucle `(ARTICLES)`, `(BREVES)` ou `(RUBRIQUES)`, la date en question est la date de publication de l'article, de la brève ou la date de dernière modification de la rubrique.

Si en revanche on se trouve au premier niveau du squelette (c'est-à-dire en-dehors de toute boucle), la date considérée est la date du jour - à moins qu'on ait passé une date dans l'URL de la page (voir l'exemple plus bas).

Dans ce dernier cas, et pour les versions de php supérieures à 3.0.12, la date passée dans l'URL est analysée avec la fonction `strtotime` : ainsi `?date=2003`, `?date=2003/01` fonctionneront, mais aussi `date=-1year` (il y a un an), `?date=1march1970` (articles publiés le 1er mars 1970), etc.

Critère de date, d'âge, et d'âge relatif

Le critère `{age}` permet de sélectionner les articles en fonction de la durée qui sépare leur date de publication en ligne avec la date courante. Ainsi `{age<30}` permettra de ne pas afficher les articles âgés de plus de 30 jours.

L'`{age_relatif}` permet de comparer les dates de publication de deux articles : si l'on vient de sélectionner un article dans une boucle, une seconde boucle placée à l'intérieur de la première pourra demander les articles publiés dans la semaine qui précède celui-ci, via `{age_relatif<=7}{age_relatif>=0}`, etc.

Les critères `{age}` et `{age_relatif}` permettent de distinguer deux articles publiés le même jour. On peut donc programmer des boucles pour obtenir l'article « précédent » ou le « suivant » :

```
<BOUCLE_art(ARTICLES){id_article}>
<BOUCLE_precedent(ARTICLES){age_relatif>=0}{par date}{inverse}{1,1}>
      précédent : <a href='#URL_ARTICLE'>#TITRE</a> #DATE
</BOUCLE_precedent>
<br />
<b>#TITRE</b> - #DATE
<br />
<BOUCLE_suivant(ARTICLES){age_relatif<0}{par date}{0,1}>
      suivant : <a href='#URL_ARTICLE'>#TITRE</a> #DATE
</BOUCLE_suivant>
</BOUCLE_art>
```

Attention ! Malgré les apparences les comparaisons de date sont d'un maniement délicat : en effet, à cause des « dates floues » (un article publié un mois donné, sans que le jour soit précisé), le calcul de l'age_relatif peut donner la valeur zéro dans un sens, et pas dans l'autre ! D'où la dissymétrie des boucles présentées ci-dessus : dans un sens on cherche le « second plus récent » des articles `{age_relatif>=0}` (car le plus récent, avec la comparaison non-stricte, ne peut être que l'article lui-même) ; dans l'autre le plus âgé des articles publiés strictement plus tard.

Les critères `{jour_relatif}`, `{mois_relatif}` et `{annee_relatif}` fonctionnent comme l'*age_relatif*, mais prennent en compte des dates arrondies au jour, au mois et à l'année respectivement ; par exemple, si l'URL comporte la variable `?date=2003-01-01`, la boucle suivante donnera « tous les les articles du mois de mars 2003 »

```
<h3>Articles de [(#DATE|nom_mois)] [(#DATE|annee)]:</h3>
<BOUCLE_blog(ARTICLES){mois_relatif=0}{par date}{"<br />"}>
      <a href='#URL_ARTICLE'>#TITRE</a>
      ([(#DATE|jour)]/[(#DATE|nom_mois)]
</BOUCLE_blog>
```

La date de rédaction antérieure

Si vous avez activé l'utilisation des dates de publication antérieure, la plupart des critères présentés ci-dessus fonctionnent : il suffit d'ajouter `_redac` au critère. Ainsi `{age_redac>365}` affichera les articles dont la date de publication antérieure remonte à plus d'un an.

Si une boucle sélectionne un article dont la date_redac est définie, une boucle interne comportant le critère `{annee_relatif_redac=0}` ira chercher les articles dont la date de publication antérieure appartient à la même année.

Un exemple de sommaire de site trié par date

A titre d’exemple, voici comment on peut afficher tous les articles d’un site, triés par mois de publication :

```
<BOUCLE_articlem(ARTICLES){par date}{inverse}>
<BOUCLE_premierdumois(ARTICLES){id_article}{doublons}>
      <b> [(#DATE|nom_mois|majuscules)] [(#DATE|annee)] </b>
<ul>
      <li><a href="#URL_ARTICLE">[(#TITRE|couper{50})]</a> -
      [(#DATE|jour)]/[(#DATE|mois)]</li>
</BOUCLE_premierdumois>
   <BOUCLE_MOIS(ARTICLES) {mois_relatif=0}{doublons}{par date}{inverse} >
       <li><a href="#URL_ARTICLE">[(#TITRE|couper{50})]</a> -
      [(#DATE|jour)]/[(#DATE|mois)]</li>
   </BOUCLE_MOIS>
</ul>
</BOUCLE_articlem>
```

Balises

#AIDER

juillet 2010

La balise `#AIDER{}` permet d'afficher l'icone de l'aide au sein de vos squelettes.

Cette balise prend comme argument la clef du titre du sous-menu affiché dans l'iframe de gauche de l'aide en ligne.
Elle crée ainsi un lien (ouvrant un « *popup* ») vers ce paragraphe précis de l'aide.

Exemple :
`#AIDER{ins_upload}` fabriquera le lien cliquable (popup) vers le paragraphe « *Installer des fichiers par FTP ou SCP* » de l'aide en ligne.

Pour connaître tous les « *points d'entrée* » (tous les arguments) utilisables, se reporter à l'article Étendre l'aide en ligne.

Attention : la balise `#AIDER` doit **obligatoirement** recevoir un argument. `#AIDER` seule dans un squelette provoquera une erreur ; si vous désirez n'appeler que le *menu* de l'aide en ligne en gardant vide l'iframe de droite il faut écrire : `#AIDER{' '}` (donner comme argument un espace entre deux apostrophes).

#ANCRE_PAGINATION

Mars 2010

La balise *#PAGINATION* crée (par défaut) une ancre html qui permet au navigateur d'afficher directement la partie de la page qui est paginée ; toutefois on peut vouloir par exemple placer les liens de pagination en dessous d'une liste d'articles, mais, dans un même temps, vouloir placer l'ancre *au-dessus* de cette liste.

C'est à cela que sert la balise **`#ANCRE_PAGINATION`**, qui crée l'ancre en question, et interdit à la balise `#PAGINATION` suivante de placer la sienne.

```
<B_page>
#ANCRE_PAGINATION
 <ul>
<BOUCLE_page(ARTICLES) {par date} {pagination}>
  <li>#TITRE</li>
</BOUCLE_page>
  </ul>
#PAGINATION
</B_page>
```

#ARRAY

Juin 2009 — maj : août 2010

La balise `#ARRAY` peut contenir un *tableau PHP*, c'est-à-dire un ensemble de paires clé/valeur, que l'on veut stocker pour les réutiliser dans la suite du squelette.

Le tableau doit être déclaré par la balise `#SET` et peut ensuite être récupéré par la balise `#GET` [1].

Il peut être utilisé ensuite, entre autres, associé au critère `IN` d'une boucle.

Déclarer un tableau et récupérer des valeurs

- **`#SET{mon_tableau, #ARRAY{cle1,valeur1,cle2,valeur2}}`**
crée la variable *mon_tableau* et lui affecte un tableau PHP comme valeur, qui pourrait être représenté comme suit :

Clés	Valeurs
cle1	valeur1
cle2	valeur2

Il n'est pas nécessaire d'entourer les chaines de caractères avec des guillemets simple ou double, sauf pour spécifier une chaine vide qui s'écrira ''

- À des fins de test, on peut afficher le tableau grâce au **filtre `|foreach`**.

`[(#GET{mon_tableau}|foreach)]` affichera :

- *cle1* => *valeur1*
- *cle2* => *valeur2*

- La valeur associée à une clé donnée peut être récupérée en utilisant le **filtre `|table_valeur`** :

`[(#GET{mon_tableau}|table_valeur{cle1})]` retourne *valeur1*.

- Vérifier la présence d'une valeur dans le tableau avec le **filtre `|find`** [2] :

```
[(#GET{mon_tableau}|find{valeur2}|oui)
  Ceci s'affiche si la valeur est dans le tableau.
]
```

Remplir un tableau dynamiquement

Un intérêt des tableaux est de les remplir dynamiquement, par les résultats d'une ou plusieurs boucle(s).

Le tableau doit alors être déclaré par un `#SET` **avant la boucle** : `#SET{mon_tableau,#ARRAY}`

Dans la boucle, un nouveau `#SET` redéfinit *mon_tableau* à chaque itération : on le récupère par `#GET` et on y ajoute une nouvelle valeur grâce aux filtres `|push` ou `|array_merge`.

```
#SET{mon_tableau,#ARRAY}
<BOUCLE(...)>
    #SET{mon_tableau, #GET{mon_tableau}|push{#COMPTEUR_BOUCLE}}
</BOUCLE>
```

L'ordre des valeurs du tableau dépend des critères de tri de la boucle.

- **`|push`** ajoute simplement une valeur à la fin du tableau. La clé est indexée automatiquement : elle est incrémentée de 1 à chaque itération de la boucle et ce, à partir de 0 (la première clé est 0, puis 1, 2, 3,...).

Exemple : créer un tableau *mots_choisis* contenant les *#ID_MOT* de tous les mots-clés liés à un article.

```
#SET{mots_choisis, #ARRAY}
<BOUCLE_themes(MOTS){id_article}>
  #SET{mots_choisis, #GET{mots_choisis}|push{#ID_MOT}}
</BOUCLE_themes>
```

Si les mots liés à cet article portent les numéros 4, 9 et 18,
`[(#GET{mots_choisis}|foreach)]` retourne :

- 0=>4
- 1=>9
- 2=>18

Notez bien que `[(#GET{mots_choisis}|table_valeur{2})]` retourne *18*, la valeur associée à la clé *2*, donc la *troisième* valeur du tableau.

- **`|array_merge`** ajoute une paire clé/valeur à la fin du tableau. Cela permet donc de forcer le "nom" des clés.

Attention (1) : si une clé apparaît plusieurs fois, seule la dernière valeur pour cette clé sera retenue.

Attention (2) : la clé que l'on veut forcer ne peut pas être de type numérique. En effet, la documentation de `array_merge` précise :

Si vous passez un seul tableau à cette fonction et qu'il a des index numériques, les clés seront réindexées normalement.

Il existe deux solutions à cette limitation :

- **préfixer** cette clé avec une valeur alphabétique :

```
#SET{mots_choisis, #ARRAY}
<BOUCLE_themes(MOTS) {id_article}>

[(#SET{mots_choisis,#GET{mots_choisis}|array_merge{#ARRAY{mot#ID_MOT,#TITRE
}}})]
</BOUCLE_themes>
```

`[(#GET{mots_choisis}|foreach)]` retourne :

- mot4=>Pomme
- mot9=>Banane
- mot18=>Carotte

- **inverser** clé et valeur, à condition que *valeur* ne soit pas elle aussi numérique :

```
#SET{mots_choisis, #ARRAY}
<BOUCLE_themes(MOTS) {id_article}>

[(#SET{mots_choisis,#GET{mots_choisis}|array_merge{#ARRAY{#TITRE,#ID_MOT}}}
)]
</BOUCLE_themes>
```

`[(#GET{mots_choisis}|foreach)]` retourne :

- Pomme=>4
- Banane=>9
- Carotte=>18

Dans ce cas, on pourra appliquer la fonction PHP `|array_flip` comme filtre sur le tableau final (après la boucle) :

```
[(#SET{mots_choisis, #GET{mots_choisis}|array_flip})]
```

`[(#GET{mots_choisis}|foreach)]` retourne alors :

- 4=>Pomme
- 9=>Banane
- 18=>Carotte

Utiliser le tableau dans une boucle avec l'opérateur *IN*

Reprenons le tableau *mots_choisis* contenant des #ID_MOT, on peut sélectionner ensuite les articles liés aux mêmes mots-clés que notre article initial grâce au **critère `IN`**.

```
#SET{mots_choisis, #ARRAY}
<BOUCLE_themes(MOTS){id_article}>
  #SET{mots_choisis, #GET{mots_choisis}|push{#ID_MOT}}
</BOUCLE_themes>

<BOUCLE_memes_themes(ARTICLES) {id_mot IN #GET{mots_choisis}}>
  #TITRE <br />
</BOUCLE_memes_themes>
```

Cette utilisation connaît de multiples usages, notamment lorsque l'on veut sélectionner des objets à l'aide de plusieurs critères qui ne peuvent faire partie de la même boucle, puis les mélanger entre eux avec un critère de tri unique.

Exemples d'utilisation des tableaux

- Affichage conditionnel en fonction de la page (valeur de `#ENV{page}`) :

```
[(#ENV{page}|in_array{#ARRAY{0,article,1,rubrique,2,site}}|oui)
Affichage conditionnel: la page est celle d'un article, d'une rubrique ou
d'un site. ]
```

- Affichage conditionnel en fonction d'une variable passée dans l'URL [3] :

```
<BOUCLE_tous_les_mots(MOTS){par titre}{", "}>
  <a href="[(#SELF|parametre_url{lolo,#ID_MOT})]">#TITRE</a>
</BOUCLE_tous_les_mots>

#SET{les_mots, #ARRAY}
<BOUCLE_certains_mots(MOTS){id_article}>
  #SET{les_mots, #GET{les_mots}|push{#ID_MOT}}
</BOUCLE_certains_mots>

[<br />Ceci s'affichera si la valeur de la variable 'lolo' passée dans
l'url est présente dans un tableau 'les_mots' déclaré et rempli
précédemment.(#ENV{lolo}|in_any{#GET{les_mots}}|oui)]
```

- Sélectionner les articles d'une rubrique et ceux associés par ailleurs à un mot-clé puis lister tous ces articles par date.

```
#SET{les_articles,#ARRAY}
<BOUCLE_articles_rubrique(ARTICLES){id_rubrique}>
  #SET{les_articles,#GET{les_articles}|push{#ID_ARTICLE}}
</BOUCLE_articles_rubrique>
<BOUCLE_articles_mot(ARTICLES){id_mot}>
  #SET{les_articles,#GET{les_articles}|push{#ID_ARTICLE}}
</BOUCLE_articles_mot>

<BOUCLE_affiche(ARTICLES){id_article IN #GET{les_articles}}{par
date}>
  <br />#TITRE
</BOUCLE_affiche>
```

Notes

[1] Voir #SET et #GET pour plus de détails.

[2] `|find` correspond à la fonction PHP `in_array`, à la différence que ce filtre ne produit pas le message d'erreur "*Warning : in_array() [function.in-array] : Wrong datatype for second argument.*" si la variable n'est finalement pas un tableau. En effet, lorsqu'une variable est générée dynamiquement, on ne sait pas toujours si elle existera et s'il s'agira bien d'un tableau.

[3] Voir `#SELF` et |parametre_url.

#AUTORISER

Janvier 2009 — maj : Mai 2009

Cette balise permet, côté squelette, d'effectuer les mêmes contrôles que ceux qu'on peut réaliser en PHP avec la fonction `autoriser()`

Elle renvoie un espace si l'autorisation est donnée, et une chaîne vide dans le cas contraire. On met autant d'arguments (et les mêmes) qu'on en mettrait dans la fonction `autoriser()`.

Par exemple, pour savoir si le visiteur courant peut accéder aux statistiques de l'article :

```
[(#AUTORISER{voirstats,article,#ID_ARTICLE})
   <a href='ecrire/?exec=....'>voir les stats</a>
]
```

Comme pour la fonction `autoriser()`, on peut passer un `#ID_AUTEUR` en argument pour demander si l'auteur en question est autorisé à .... .

Par exemple pour signaler d'une étoile les administrateurs et rédacteurs dans une liste d'auteurs :

```
<BOUCLE_a(AUTEURS){tous}>
   #NOM [(#AUTORISER{ecrire,'','',#ID_AUTEUR}) *]
</BOUCLE_a>
```

Un autre exemple, ci dessous, si le visiteur a des droits de modifications sur l'article, afficher un formulaire pour l'éditer, qui, une fois validé, retourne sur la page de l'article en question :

```
[(#AUTORISER{modifier, article,#ID_ARTICLE})
   #FORMULAIRE_EDITER_ARTICLE{#ID_ARTICLE, #ID_RUBRIQUE, #URL_ARTICLE}
]
```

Rappel : les arguments de autoriser sont, dans l'ordre, (faire, quoi, id, qui, options).

Pour plus de détails nous vous invitons à consulter la documentation des fonctions de inc/autoriser.php.

#BIO

Octobre 2009

Utilisée dans une boucle (AUTEURS), la balise `#BIO` affiche la biographie de l'auteur.

#BOUTON_ACTION (depuis SPIP 2.1)

Avril 2010

La balise `#BOUTON_ACTION{libellé, url, class, confirm_message}` produit le source html complet d'un formulaire (methode POST) comportant uniquement un bouton de type `submit`.

L'intérêt de cette balise est qu'elle permet de transmettre des arguments à une url grâce à un pseudo-lien *non suivable par un robot* (les robots ne « cliquent » pas sur un bouton submit). C'est particulièrement utile dans le cadre d'une url qui renvoit vers un script d'intervention sur les données en base. Il faut alors protéger le lien vers cette url de tout risque d'être suivi par un robot.
`#BOUTON_ACTION` est là pour ça.

La balise est à utiliser ainsi :

```
#BOUTON_ACTION{libellé, url, class ,message de confirmation}
```

Le *3e argument*, optionnel, est un nom de classe css ; lui donner la valeur « *ajax* » permet au bouton de se comporter comme un lien ajax ;
Le *4e argument*, optionnel, permet de préciser le message de confirmation affiché lors de la validation du bouton.

- Attention :

  - si vous désirez préciser un message de confirmation, vous *devez* aussi renseigner le 3e argument : `#BOUTON_ACTION{libellé, url ,message de confirmation}` ne fonctionnera pas comme vous l'entendez (« *message de confirmation* » sera passé comme nom de classe css). Pour préciser un message de confirmation *sans* préciser de classe css, écrire :

    ```
    #BOUTON_ACTION{libellé, url , '', message de confirmation}
    ```

  - Si vous utilisez des caractères accentués dans votre message de confirmation, vous devez les traduire en entités numériques (`é` => `&#233;`).

Exemple :

```
#BOUTON_ACTION{valider, #URL_PAGE{mon_script}|parametre_url{var1,
val1}|parametre_url{var2, val2}|parametre_url{var3, val3}, ajax, &#234;tes-
vous s&#251;r ?}
```

produira le source html suivant :

```
<form class='bouton_action_post ajax' method='post'
action='./?page=mon_script&amp;var1=val1&amp;var2=val2&amp;var3=val3'>
<div>
<input name="page" value="mon_script" type="hidden" />
<input name="var1" value="val1" type="hidden" />
<input name="var2" value="val2" type="hidden" />
<input name="var3" value="val3" type="hidden" />
<button type='submit' class='submit' onclick='return confirm("&#234;tes-
vous s&#251;r ?");'>valider</button>
```

```
</div>
</form>
```

#CACHE

Mars 2010

La balise **#CACHE{temps en secondes}** permet de déterminer le délai au bout duquel le squelette est à nouveau calculé (`var_mode=calcul` dans l'URL actualise le cache de la page). Le temps est exprimé en secondes. Il peut être indiqué sous forme de calcul.

Cette balise est généralement placée au tout début des squelettes. En son absence, par défaut, la durée est de 24h (défini par la constante _DUREE_CACHE_DEFAUT).

Par exemple : `#CACHE{24*3600*30}` signifie que tous les mois (30 jours) votre squelette sera calculé à nouveau.

#CHAMP_SQL

Mars 2009

La balise `#CHAMP_SQL` peut être utilisée dans toutes les boucles et permet d'extraire un champ d'une table SQL. `#CHAMP_SQL` se révèle très utile quand une balise du même nom que le champ existe déjà (avec le multibases par exemple). Prenons l'exemple de `#POINTS` qui est inscrit dans le core de SPIP ; Pour extraire un champ #POINTS, nous devons obligatoirement utiliser `#CHAMP_SQL{points}` sinon le compilateur de SPIP croira que nous faisons appel à `#POINTS` qui ne peut se trouver que dans une boucle avec le critère `{recherche}`.

Remarque : La balise `#CHAMP_SQL` ne peut être définie dynamiquement, c'est obligatoirement une constante qui doit servir de paramètre. `#CHAMP_SQL{points}` fonctionne mais pas `#CHAMP_SQL{#GET{champ}}`.

#CHAPO

Décembre 2008

- **#CHAPO** affiche le texte d'introduction (chapeau).

#CHARSET

Mars 2010

`#CHARSET` affiche le jeu de caractères utilisé par le site. Sa valeur par défaut `utf-8` (jeu de caractères dit « universel »).

Ce charset est configurable depuis la page de configuration du site : `ecrire/?exec=config_lang` (« *Jeu de caractères du site* »).

#CHEMIN

Octobre 2009 — maj : Décembre 2009

La balise `#CHEMIN{fichier.ext}` retourne le chemin complet vers *fichier.ext*, qu'il se trouve dans le dossier squelettes/, dans le dossier d'un plugin, à la racine, dans squelettes-dist/ etc.

`#CHEMIN` parcourt l'ensemble du « *spip_path* » **dans un ordre précis** (voir ci-après) jusqu'à trouver le fichier recherché :

- d'abord :
  - ./squelettes/
- puis, *dans l'ordre alphabétique du nom de leur dossier* et indépendamment de leur place dans auto/ ou non, les plugins *nécessitant* d'autres plugins :
  - ./plugins/auto/plugin_A_nécessitant_plugin_X/
  - ./plugins/plugin_B_nécessitant_plugin_Y/
  - ./plugins/plugin_C_nécessitant_plugin_X/
- puis, *sans ordre particulier*, les autres plugins :
  - ./plugins/plugin_Y/
  - ./plugins/plugin_G
  - ./plugins/plugin_X/
  - ./plugins/plugin_E
- puis *la racine* du site :
  - ./
- enfin, *et dans cet ordre*, les 3 répertoires :
  - ./squelettes-dist/
  - ./prive/
  - ./ecrire/

Exemples :

```
<link rel="stylesheet" href="#CHEMIN{le_style.css}" type="text/css" />
<img src="#CHEMIN{images/mon_logo.png}" alt="" />
```

#COMPTEUR_BOUCLE

Septembre 2009 — maj : mars 2010

`#COMPTEUR_BOUCLE` retourne le numéro de l'itération actuelle de la boucle. On peut par exemple l'utiliser pour numéroter des résultats :

```
<BOUCLE_art(ARTICLES) {par date} {inverse} {0,10}>
      #COMPTEUR_BOUCLE - #TITRE<br>
</BOUCLE_art>
```

#CONFIG

Mai 2010

La balise technique `#CONFIG` permet d'afficher la valeur d'un réglage stocké dans la table *spip_meta*.

`#CONFIG` retourne le contenu brut d'une valeur contenu dans la table *spip_meta*
Exemple : Connaître si les visiteurs sont acceptés sur le site :
`[Visiteurs: (#CONFIG{accepter_visiteurs})]` affichera "oui" ou "non".

Il est possible de fournir une valeur par défaut si le contenu de la meta est vide.
`#CONFIG{nom_meta,valeur_defaut}`
Exemple : Connaître l'adresse du site
`[Adresse du site : (#CONFIG{adresse_site, pas d'url pour le site})]`

Attention : la balise retournant la valeur *brute* depuis la table spip_meta, les valeurs stockées sous forme de tableau sérializé sont retournées telles qu'elles.

#DATE

Novembre 2009 — maj : août 2010

`#DATE` est la date de mise en ligne. (Modifiable après la mise en ligne de l'article, de la brève, etc. La date d'une rubrique est celle de son élément le plus récent.)

Si une date est rentrée de façon incomplète, vous devez utiliser un "*" après la balise `#DATE` pour un affichage compréhensible dans la partie public. Par exemple, avec `[(#DATE*|affdate)]` :
- votre date a été rentrée sous la forme "*nc-juillet-2009*", vous aurez à l'affichage "*juillet 2009*"
- votre date a été rentrée sous la forme "*nc-non connu-2009*", vous aurez à l'affichage "*2009*"

#DATE_MODIF, #DATE_REDAC

Décembre 2008

- Les balises de dates : `#DATE, #DATE_REDAC, #DATE_MODIF` sont explicitées dans la documentation sur « La gestion des dates ».

#DATE_NOUVEAUTES

Avril 2010

La balise `#DATE_NOUVEAUTES` affiche la date du dernier envoi du mail présentant les nouveautés si cette fonction a été sélectionnée dans la boite *« Annonce des nouveautés »* de la page configuration du site `ecrire/?exec=config_contenu` (si cette fonction est désactivée la balise retourne 1970-01-01 01:00:00).

#DESCRIPTIF

Décembre 2008

- **`#DESCRIPTIF`** affiche le descriptif.

#DESCRIPTIF_SITE_SPIP

Septembre 2009— maj : mars 2010

`#DESCRIPTIF_SITE_SPIP` affiche, comme son nom l'indique, le descriptif du site, que l'on renseigne dans la page de configuration générale du site.

#DISTANT

Mars 2010

La balise `#DISTANT` affiche « oui » ou « non » selon que le document est distant (référencé par une url) ou pas.

Exemple (afficher une imagette pour indiquer que le document est hébergé sur un site externe) :

```
[(#DISTANT|=={oui}|oui) <img src="ext.png" alt="">]
```

#DOSSIER_SQUELETTE

mars 2010

dépréciée par #CHEMIN qui la remplace et l'améliore.

#DOUBLONS

Novembre 2009 — maj : juillet 2010

`#DOUBLONS{mots}` ou `#DOUBLONS{mots,famille}` donne l'etat des doublons `(MOTS)`à cet endroit sous forme de tableau d'id_mot `array(1,2,3,...)`

`#DOUBLONS` tout seul donne la liste brute de tous les doublons

`#DOUBLONS*{mots}` donne la chaine brute ",1,2,3,..." (changera si la gestion des doublons evolue)

pour des doublons « *nommés* » (exemple : `<BOUCLE_b(ARTICLES){id_article>100}{doublons A}>`) on utilisera : `#DOUBLONS{articles, A}` pour avoir le tableau des id_articles.

pour passer le ou les tableaux de doublons à un INCLURE, on notera : `#INCLURE{fond=noisette, env, doublons=#DOUBLONS{articles, A}}` pour un tableau nommé `{doublons A}` d'une BOUCLE(ARTICLES)
et `#INCLURE{fond=noisette, env, doublons}` pour tous les tableaux de doublons déclarés dans le squelette appelant (avant l'appel de l'INCLURE).

pour récupérer un tableau de doublons **depuis** un INCLURE, il conviendra de ne faire retourner par cet INCLURE que `#DOUBLONS*{articles, B}` et, dans le squelette appelant, d'utiliser `#SET{ret, #INCLURE{fond=noisette, env, doublons}|explode{','}}` : le `#GET{ret}` résultant pouvant alors être utilisé dans un critère de boucle (`{id_article IN #GET{ret}}`).

#EDIT

avril 2010

`#EDIT{xxx}` dans la class d'un élément entourant la balise `#XXX` permet de la rendre éditable à l'aide du plugin crayons (voir la documentation sur spip-contrib)

#EDITABLE

5 juillet 2010

Utilisée à l'intérieur d'un squelette de formulaire CVT, la balise #EDITABLE indique si l'internaute doit saisir les données d'un formulaire.

En la testant, le squelette peut masquer la partie saisie du formulaire lorsqu'il est appelé après la validation.

L'exemple de Formulaires CVT par l'exemple est :

```
[<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>
]
```

Si on vient d'accepter les données, on en informe l'internaute.
Si il y a une erreur, on l'affiche
si on en en phase de saisie, on affiche le formulaire html

#EMAIL

mars 2010

La balise `#EMAIL` affiche l'adresse e-mail d'un auteur.
Elle est utilisable dans une boucle AUTEURS, une boucle FORUMS ou une boucle SIGNATURES.

#EMAIL_WEBMASTER

Mars 2010

La balise `#EMAIL_WEBMASTER` affiche l'adresse mail du webmestre telle qu'elle a été renseignée dans la page de configuration du site.

#EMBED_DOCUMENT

Mars 2010
Cette balise est dépréciée, au profit de **`#MODELE`**

#ENV

Février 2009 — maj : mai 2010

Balise permettant d'obtenir la valeur d'une variable fournie par un #INCLURE ou depuis une url

```
[(#ENV{variable, valeur_par_defaut})]
```

`#ENV{variable}` permet d'accéder aux variables d'environnement d'un squelette (son contexte) [1].

Le squelette peut recevoir cette variable soit de l'URL (la requête HTTP), soit du fichier qui l'inclut, soit encore d'un retour de formulaire.

Exemple de variable passée dans l'url : `spip.php?rubrique24&id_mot=5`
Dans *rubrique.html*, `#ENV{id_mot}` vaut **5**.

Exemple de variable passée dans un INCLURE :`<INCLURE{fond=mon_squelette}{id_article=136}>`
Dans *mon_squelette.html*, `#ENV{id_article}` vaut **136**.

Valeur par défaut

`[(#ENV{variable, valeur_par_defaut})]` : le paramètre optionnel *valeur_par_defaut* contient la valeur qui doit être retournée si la variable n'existe pas (équivalent à `[(#ENV{variable}|sinon{valeur_par_defaut})]`).

Utile quand un squelette est inclu à partir de différents squelettes, et que l'on en attend un comportement différent selon le contexte.

Court-circuiter les traitements de sécurité

Par défaut, SPIP applique les fonctions `interdire_scripts` et `entites_html` à la balise **`#ENV`**. Comme pour toute balise, on peut supprimer les traitements automatiques de SPIP avec **`#ENV*`** et **`#ENV**`** (voir, pour l'usage de ***** et ******, l'article *#BALISE* et #BALISE***).

`#ENV*` retourne la variable sans appliquer le filtre `entites_html`, donc sans transformer tous les caractères spéciaux en entités HTML.
`#ENV**` retourne la variable sans appliquer la fonction `interdire_scripts`. Elle renvoie donc le PHP exécuté. [2]
Ceci peut poser un problème de sécurité si cette variable reçoit une injection de code. Voilà pourquoi `interdire_scripts` est là par défaut pour invalider les < et autres `<script language=php>` contenus dans la variable.

Néanmoins **#ENV**** trouve une application lorsque l'on construit un formulaire. L'utilisation de la double étoile est ainsi nécessaire dans l'affichage des formulaires CVT. Par exemple, pour récupérer le tableau des erreurs dans le squelette du formulaire :

```
#ENV**{erreurs}|table_valeur{clef}...
```

Notes

[1] Par sécurité, `#ENV` ne récupère (ni n'affiche) jamais les variables nommées `PHPSESSID` ou dont le nom commence par `var_` (comme par exemple var_mode et var_profile).

[2] imaginons un formulaire tout simple :

```
<form method="get">
<input type="text" name="test" value="#ENV{test}"> <input type="submit">
</form>
#ENV{test}<br />
#ENV*{test}<br />
#ENV**{test}<br />
```

dans le formulaire affiché, entrons : `<?php echo date('Y-m-d'); ?>` dans le champ « *test* », puis validons.

- `#ENV{test}` renverra (source html) : `<?php echo date('Y-m-d'); ?>`
- `#ENV*{test}` renverra (source html) : `<?php echo date('Y-m-d'); ?>`
- `#ENV**{test}` renverra (source html) : `2009-02-12` *c'est à dire le php* ***exécuté***.

#EVAL

Mars 2010

La balise technique **#EVAL{}** retourne l'évaluation de l'expression PHP passée en argument :
`#EVAL{2*7}` affiche donc **14**,
`#EVAL{_DIR_IMG_PACK}` affiche le chemin vers le répertoire `ecrire/img_pack/`,
`#EVAL{phpversion()}` affiche la version php du serveur
`#EVAL{$_SERVER['REQUEST_URI']}` ...

Cette balise interprétant du code PHP, elle est à utiliser avec précaution.

#EXTENSION

Mars 2010

`#EXTENSION` affiche l'extension de format du fichier, par exemple : `pdf, jpeg, mov, ra`...

#FICHIER

Mars 2010

La balise `#FICHIER` affiche l'URL *relative* du document [1] ; par exemple : *IMG/pdf/mon_document.pdf*.
Pour extraire le nom seul du fichier, on pourra utiliser le filtre : `|basename` : `[(#FICHIER|basename)]` (affichera `mon_document.pdf`).

Une utilisation intéressante de cette balise est combinée avec le filtre `|image_reduire`, dans le cadre d'un portfolio pour afficher une réduction de l'image plutôt que de son logo ; par exemple en utilisant

```
[<a href="#URL_DOCUMENT">(#FICHIER|image_reduire{500}) </a>]
```

Notes

[1] dans le cas d'un document distant, `#FICHIER` affiche l'URL complète vers le document (http://autre_site.com/rep/document.ext)

#FILTRE

juillet 2010

`#FILTRE{f}` applique le filtre `f` à l'ensemble du squelette une fois celui-ci calculé.

Cette balise prend en paramètre un filtre ou une combinaison de filtres.
Ex : `#FILTRE{supprimer_tags|filtrer_entites|trim}`

Placée à la fin d'un squelette elle applique son paramètre en tant que filtre à l'ensemble du squelette généré, une fois celui-ci calculé.

Précision : Ce filtrage s'applique aux `#INCLURE` du squelette recevant cette balise , mais ne s'applique pas aux `<INCLURE>` contenu dans ce squelette.

Exemple : dans le plugin notification, un squelette `inscription.html` génère le contenu d'un mail. Pour rendre ce contenu adapté à un mail, le résultat du squelette est traité par une série de filtres destinés à enlever les tags HTML, les espaces redondants et les entités HTML : `#FILTRE{supprimer_tags|filtrer_entites|trim}` (voir : http://zone.spip.org/trac/spip-zone/browser/_plugins_/notifications/notifications/inscription.html?rev=38419)

#FOREACH

Décembre 2009 — maj : mars 2010

La balise `#FOREACH` permet de « boucler » sur un contenu équivalent à un tableau calculé dans le squelette en appliquant pour chacun de ses éléments un modèle SPIP.

`#FOREACH{balise, modele}`

Exemples

- Dans un squelette quelconque, l'utilisation de `#FOREACH{env}` affichera toutes les variables de la balise #ENV de la manière suivante :

- page=> test_boucle
- date=> 2009-12-22 22:44:27
- date_default=> 1
- date_redac=> 2009-12-22 22:44:27
- date_redac_default=> 1

Autrement dit, pour chaque élément du tableau associatif php que représente la balise #ENV, #FOREACH appliquera le modèle foreach.html (cf. plus bas).

- On peut vouloir afficher le contenu d'une autre balise. #CONFIG par exemple. Pour cela, on passera le nom de la balise comme premier paramètre :

`#FOREACH{config}` affichera donc à peu près :

- charset_sql_base=> utf8
- charset_collation_sql_base=> utf8_general_ci
- charset_sql_connexion=> utf8
- version_installee=> 14558
- nouvelle_install=> 1
- plugin=> `a:3:{s:3:"CFG";a:4:{s:3:"nom";s:90:"<multi> [fr]cfg: moteur de configuration [de]cfg: Konfigurationsmotor </multi>";s:4:"etat";s:6:"stable";s:3:"dir";s:8:"auto/cfg";s:7:"version";s:6:"1.14.1";}}`
- derniere_modif_rubrique=> 1260746188
- ...=> ...

Et pour toute balise SPIP représentant un tableau php on aura un résultat similaire (voir #ARRAY).

Fabriquer un modèle dédié

Le modèle `foreach.html` est celui qui est appliqué par défaut. On le trouve dans `squelettes-dist/modeles/`. Il est possible de le modifier, mais on peut aussi fabriquer un modèle dédié à une balise et le placer dans son propre dossier `squelettes/modeles/`.

Le modèle `foreach_balise.html` correspond à `#FOREACH{balise}`.

On peut aussi souhaiter créer des modèles généraux et les utiliser pour telle ou telle balise. Par exemple, pour des listes non énumérées un modèle nommé `squelettes/modeles/foreach_ul_li.html`, qui contiendrait : `<li><a href="#ENV{valeur}">#ENV{cle}</a></li>` serait appelé par `[<ul>(#FOREACH{ENV, foreach_ul_li})</ul>]` et afficherait (source html) :

```
<ul>
<li><a href="test_boucle">page</a></li>
<li><a href="2009-12-22 22:54:40">date</a></li>
<li><a href="1">date_default</a></li>
<li><a href="2009-12-22 22:54:40">date_redac</a></li>
<li><a href="1">date_redac_default</a></li>
</ul>
```

Plus de valeurs

Enfin, dans les cas plus complexes, on peut passer plus de valeurs au modèle en respectant la structure de tableau suivante pour la balise que vous aurez conçue :

```
array (
    'cle' => array (
        'val1' => 'valeur1'
        'val2' => 'valeur2'
        'val3' => 'valeur3'
        'val4' => 'valeur4'
    )
    'cle2' => array (
        'val1' => 'valeur1bis'
        'val2' => 'valeur2bis'
        'val3' => 'valeur3bis'
        'val4' => 'valeur4bis'
    )
)
```

et en fournissant un modèle adapté :

```
#ENV{val1}, #ENV{val2}, #ENV{val3} et #ENV{val4}
```

#FORMULAIRE_ADMIN

13 août 2010

La balise `#FORMULAIRE_ADMIN` est une balise optionnelle qui permet de placer les boutons d'administration (« recalculer cette page », etc.) dans ses squelettes. Lorsqu'un administrateur parcours le site public, si cette balise est présente, elle sera remplacée par les boutons d'administration, sinon, les boutons seront placés à la fin de la page.

#FORMULAIRE_ECRIRE_AUTEUR

13 août 2010

La balise `#FORMULAIRE_ECRIRE_AUTEUR` fabrique et affiche un formulaire permettant d'écrire à l'auteur. Il faut que le serveur hébergeant le site accepte d'envoyer des mails. Ce système permet de ne pas divulguer l'adresse email de l'auteur.
NB : pour que le formulaire soit affiché il faut que l'auteur ait renseigné le champ **email** de sa fiche auteur.

#FORMULAIRE_FORUM

Janvier 2009

- **`#FORMULAIRE_FORUM`** fabrique et affiche le formulaire permettant de poster un message répondant à cet article. Pour en savoir plus, voir aussi « Les formulaires ».

#FORMULAIRE_INSCRIPTION

7 septembre 2010

La balise `#FORMULAIRE_INSCRIPTION` affiche le formulaire permettant l'inscription de nouveaux rédacteurs. Celui-ci ne s'affiche que si vous avez autorisé l'inscription automatique depuis le site public (sinon, cette balise n'affiche rigoureusement rien).

L'inscription nécessite l'envoi des informations de connexion (login et mot de passe) par email ; ce formulaire ne fonctionne donc que si votre hébergeur autorise l'envoi de mails par PHP.

`[(#FORMULAIRE_INSCRIPTION{6forum})]` est l'équivalente de la précédente, pour l'inscription des visiteurs, appelés à écrire dans les forums (réservés aux visiteurs enregistrés), option qui se détermine dans la partie privée configuration/interactivité/Mode de fonctionnement par défaut des forums publics. Le paramètre « 6forum » correspond à l'intitulé du statut d'auteur désiré lors de l'inscription.

Après la validation, un message avertit le visiteur : "Votre nouvel identifiant vient de vous être envoyé par email."

#FORMULAIRE_SIGNATURE

Janvier 2009

- **`#FORMULAIRE_SIGNATURE`** fabrique et affiche le formulaire permettant de signer la pétition associée à cet article.

#GET

Décembre 2009 — maj : mars 2010

Voir **#SET et #GET**.

#HAUTEUR

Mars 2010

`#HAUTEUR` retourne la hauteur d'un document de type image (png, gif ou jpg) exprimée en pixels (retourne 0 pour les autres types de document).

#HTTP_HEADER

24 mars 2010

La balise `#HTTP_HEADER{argument}` permet de modifier l'entête HTTP de la page retournée par SPIP.

Remarques
- `#HTTP_HEADER` ne peut être précédée d'aucun caractère ni d'espace.
- `#HTTP_HEADER` ne doit pas être précédé de code html. Exemple :

```
<BOUCLE_truc(ARTICLES)>
<!-- debut de page -->
#HTTP_HEADER{...}
...
```

pas bon !

```
<BOUCLE_truc(ARTICLES)>
#HTTP_HEADER{...}
<!-- debut de page -->
...
```

bon !

- L'utilisation de cette balise supprime les boutons d'administration.
- Cette balise ne peut pas être utilisée dans des squelettes inclus via la syntaxe `<INCLURE>`.

Exemples d'utilisation
- `#HTTP_HEADER{Content-Type: text/css}`, permet de préciser que le squelette (dont l'extension reste .html) est en fait une feuille de style. Voir pour plus d'informations l'article sur SPIP-Contrib : http://www.spip-contrib.net/Feuille-de-style-dynamique-des.
- `#HTTP_HEADER{Content-Type: text/plain; charset=#CHARSET}` pour la version texte d'un article.

- `#HTTP_HEADER{Content-Type: text/csv; charset=#CHARSET}` `#HTTP_HEADER{Content-Disposition: attachment; filename=rapport.csv}` pour un squelette qui produit un fichier CSV.

#ID_ARTICLE

Décembre 2008

- **`#ID_ARTICLE`** affiche l'identifiant unique de l'article. Utile pour fabriquer des liens hypertextes non prévus (par exemple vers une page « Afficher au format impression »).

#ID_DOCUMENT

Janvier 2010

- `#ID_DOCUMENT` affiche l'identifiant unique du document.

#ID_RUBRIQUE

Décembre 2008

- **`#ID_RUBRIQUE`** affiche l'identifiant de la rubrique dont dépend l'article.

#ID_SECTEUR

Décembre 2008

- **`#ID_SECTEUR`** affiche l'identifiant du secteur dont dépend l'article (le secteur étant la rubrique parente située à la racine du site).

#INSERT_HEAD

Juin2010

`#INSERT_HEAD` insère automatiquement les scripts et feuilles de style fournis par SPIP et ses plugins.

La balise `#INSERT_HEAD` doit se situer entre les balises `<head>` et `</head>` de vos squelettes. Elle permet à SPIP, à ses extensions, ainsi qu'aux plugins éventuels, d'ajouter du contenu entre ces deux balises html.

Une fonction de compression existe. Cette fonctionnalité optimise la consommation de la bande passante pour l'ensemble des éléments transmis via cette balise en enlevant tous les espaces et commentaires "superflues" des fichiers .js et .css de tous les squelettes. On peut activer ou désactiver cette fonction dans le panneau de configuration de la partie privée.

Depuis SPIP 2.1, il est fortement recommandé d'utiliser systématiquement cette balise dans ses squelettes. Cette recommandation est due au passage en extension de certaines fonctionnalités du core qui se comportent alors comme des plugins.

#INSERT_HEAD_CSS

Juillet 2010

`#INSERT_HEAD_CSS` indique aux plugins où insérer leurs feuilles de style.

Les squelettes peuvent surcharger ces feuilles de style en fonction de leur ordre d'appel dans les squelettes.

A noter que, pour conserver la compatibilité avec les squelettes existants, si la balise `#INSERT_HEAD_CSS` n'est pas présente dans le squelette, #INSERT_HEAD insérera son contenu (il est à noter alors que l'ordre d'insertion ne sera pas optimale)

#INTRODUCTION

Janvier 2009 — maj : Septembre 2009

- **#INTRODUCTION** affiche le descriptif de l'article, sinon affiche les 600 premiers caractères de l'article (chapo puis texte).

Pour préciser spécifiquement l'introduction, il est possible d'encadrer la portion de texte voulue de son texte (en mode édition) par les tags `<intro>` et `</intro>` :

```
La foule des bourgeois et des bourgeoises s'acheminait donc de toutes parts
dès le matin, maisons et boutiques fermées, vers l'un des trois endroits
désignés. <intro>Chacun avait pris parti, qui pour le feu de joie, qui pour
le mai, qui pour le mystère.</intro> Il faut dire, à l'éloge de l'antique
bon sens des badauds de Paris, que la plus grande partie de cette foule se
dirigeait vers le feu de joie, lequel était tout à fait de saison, ou vers
le mystère, qui devait être représenté dans la grand'salle du Palais bien
couverte et bien close, et que les curieux s'accordaient à laisser le
pauvre mai mal fleuri grelotter tout seul sous le ciel de janvier dans le
cimetière de la chapelle de Braque.
_ {(v.hugo)}
```

`#INTRODUCTION` affichera « *Chacun avait pris parti, qui pour le feu de joie, qui pour le mai, qui pour le mystère.* »

Lorsqu'elle coupe un texte de plus de 600 caractères, la balise `#INTRODUCTION` ajoute à l'extrait des *points de suite* (`(...)`)
on peut vouloir modifier ces *points de suite* et pour cela, il faut ajouter dans mes_options.php :

```
define('_INTRODUCTION_SUITE', 'ce que je veux ici');
```

#LANG

Décembre 2008

- **#LANG** affiche la langue de cet article.

#LANG_DIR, #LANG_LEFT, #LANG_RIGHT

avril 2010

`#LANG_DIR`, `#LANG_LEFT`, `#LANG_RIGHT` : ces balises définissent le sens d'écriture de la langue du contexte actuel (par exemple, de l'article qu'on est en train d'afficher). Voir l'article « Réaliser un site multilingue » pour plus d'information.

#LARGEUR

Mars 2010

`#LARGEUR` retourne la largeur d'un document de type image (png, gif ou jpg) exprimée en pixels (retourne 0 pour les autres types de document).

#LESAUTEURS

Janvier 2009

- **`#LESAUTEURS`** affiche les auteurs de cet article, avec lien vers leur propre page publique (afin de pouvoir directement leur écrire ou de consulter la liste des articles qu'ils ont publié). Cela évite de créer une boucle AUTEURS pour obtenir le même résultat.

#LOGO_ARTICLE

Août 2010

Cette balise affiche le logo de l'article, éventuellement avec la gestion du survol.
Deux balises permettent de récupérer une seule des deux variantes du logos :
#LOGO_ARTICLE_NORMAL affiche le logo sans survol ;
#LOGO_ARTICLE_SURVOL affiche le logo de survol.

- **Dans un squelette**, le logo de l'article s'installe de la manière suivante :
[(#LOGO_ARTICLE|alignement|adresse)]
L'alignement ne peut être que left, right, center, top ou bottom. Cet alignement sera traduit dans le code html par `class=" ... "` (penser à définir la classe correspondante dans ses feuilles de style).
L'adresse est l'URL de destination du lien de ce logo (par exemple #URL_ARTICLE). Si l'on n'indique pas d'adresse, l'image n'est pas cliquable.
Exemple :

```
[(#LOGO_ARTICLE|right|#URL_ARTICLE)]
```

affiche le logo à droite avec une lien vers la page de l'article.

- **#LOGO_ARTICLE_RUBRIQUE** affiche le logo de l'article, éventuellement remplacé par le logo de la rubrique s'il n'existe pas de logo spécifique à l'article.

- Si l'on veut récupérer directement le nom du fichier du logo (alors que les balises précédentes fabriquent le code HTML complet pour insérer l'image dans la page), par exemple pour afficher une image en fond de tableau, on utilisera le filtre |fichier comme suit :

```
[(#LOGO_ARTICLE|fichier)]
```

- **Les filtres d'image** (recadrage, réduction, masque...) peuvent s'appliquer sur le logo de l'article.
Exemple :

```
[(#LOGO_ARTICLE||image_reduire{200,200})]
```

affiche le logo de l'article, dont la plus grande des 2 tailles (si elle est supérieure à 200px) est réduite à 200px et l'autre en proportion (voir le filtre image_reduire).

- Il est possible d'associer un **attribut alt** au logo de l'article en utilisant le filtre inserer_attribut :

```
[(#LOGO_ARTICLE|#URL_ARTICLE||inserer_attribut{alt,#TITRE})]
```

utilise le titre de l'article comme attribut alt du logo.

- Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

#LOGO_ARTICLE_RUBRIQUE

juillet 2010

Cette balise affiche le logo de l'article, éventuellement remplacé par le logo de la rubrique s'il n'existe pas de logo spécifique à l'article.

- **Dans un squelette**, ce logo s'installe de la manière suivante :
[(#LOGO_ARTICLE_RUBRIQUE|alignement|adresse)]
L'alignement ne peut être que left, right, center, top ou bottom. Cet alignement sera traduit dans le code html par `class=" ... "` (penser à définir la classe correspondante dans ses feuilles de style).
L'adresse est l'URL de destination du lien de ce logo (par exemple #URL_ARTICLE). Si l'on n'indique pas d'adresse, l'image n'est pas cliquable.
Exemple :

```
[(#LOGO_ARTICLE_RUBRIQUE|center|#URL_ARTICLE)]
```

affiche le logo au centre avec une lien vers la page de l'article.

- **Les filtres d'image** (recadrage, réduction, masque...) peuvent s'appliquer sur ce logo.
Exemple :

```
[(#LOGO_ARTICLE_RUBRIQUE||image_reduire{200,200})]
```

affiche le logo de l’article ou à défaut celui de sa rubrique, dont la plus grande des 2 tailles (si elle est supérieure à 200px) est réduite à 200px et l’autre en proportion (voir le filtre image_reduire).

#LOGO_AUTEUR

Août 2010

Cette balise affiche le logo de l'auteur, éventuellement avec la gestion du survol.

Cette balise affiche le logo de l'auteur, éventuellement avec la gestion du survol.

Les variantes #LOGO_AUTEUR_NORMAL et #LOGO_AUTEUR_SURVOL permettent un affichage plus fin de ces deux variantes du logo.

Depuis SPIP 2.1, l'appel des diverses balises `#LOGO_xxx` ainsi que leur *argumentation* ont été modifiés.

- **Dans un squelette**, le logo s'installe de la manière suivante :

```
[(#LOGO_AUTEUR|alignement|adresse)]
```

L'alignement ne peut être que left, right, center, top ou bottom. Cet alignement sera traduit dans le code html par `class=" ... "` (penser à définir la classe correspondante dans ses feuilles de style).
L'adresse est l'URL de destination du lien de ce logo (par exemple #URL_AUTEUR). Si l'on n'indique pas d'adresse, l'image n'est pas cliquable.
Exemple :

```
[(#LOGO_AUTEUR|left|#URL_AUTEUR)]
```

affiche le logo de l'auteur à gauche avec un lien sur le logo vers l'adresse de la fiche auteur.

- **Les filtres d'image** peuvent s'appliquer sur le logo de l'auteur.
Exemple :

```
[(#LOGO_AUTEUR||image_reduire{200,200})]
```

affiche le logo de l'auteur, dont la plus grande des 2 tailles (si elle est supérieure à 200px) est réduite à 200px et l'autre en proportion (voir le filtre image_reduire).

- Le squelette auteur.html de la dist propose d'afficher un favicon personnalisé sur la page auteur à partir du logo :

```
[(#MODELE{favicon}{favicon=#LOGO_AUTEUR})]
```

Depuis SPIP 2.1 les filtres de logos ont la même syntaxe que tous les autres. Un seul pipe suffit :

```
[(#LOGO_XXX|filtre)]
```

Mais :
- `#LOGO_XXX**` renvoie le fichier
- `#LOGO_XXX{top/left/right/center/bottom}` génère un alignement
- `#LOGO_XXX{url}` génère un logo qui pointe vers l'url.

#LOGO_AUTEUR_NORMAL

avril 2010

Cette balise affiche le logo de l'auteur, sans survol.

Dans un squelette, il s'utilise comme la balise #LOGO_AUTEUR :

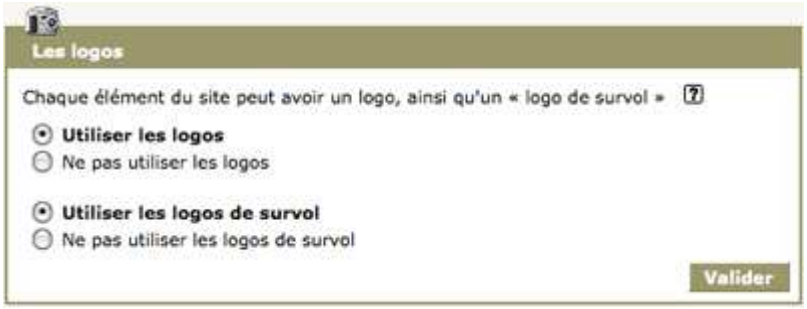
```
[(#LOGO_AUTEUR_NORMAL|alignement|adresse)]
```

#LOGO_AUTEUR_SURVOL

juillet 2010

Cette balise affiche l'état de survol du logo de l'auteur. Autrement dit le passage de la souris sur le logo de l'auteur déclenche l'affichage du logo de survol.

- Par défaut, les logos de survol ne sont pas activés dans SPIP ; on associe alors un seul logo à un objet (article, auteur, rubrique...)
Pour pouvoir gérer l'affichage d'un logo de survol, il faut tout d'abord activer l'option "Utiliser les logos de survol " dans l'interface d'administration de SPIP, sur la page "Configuration > Contenu du site" (`ecrire/?exec=configuration`).



- Dans un squelette, dans une boucle AUTEURS :

```
[(#LOGO_AUTEUR|alignement|adresse)]
```

affiche le logo de l'auteur et au passage de la souris le logo de survol s'il existe. Le code html est généré automatiquement par SPIP.

- Si l'on souhaite afficher directement le logo de survol, il faudra indiquer, dans la boucle AUTEURS du squelette :

```
[(#LOGO_AUTEUR_SURVOL|alignement|adresse)]
```

- Si l'on souhaite afficher le logo de survol s'il existe, et sinon le logo de l'article il faudra écrire, dans la boucle AUTEURS du squelette :

```
[(#LOGO_AUTEUR_SURVOL||sinon{#LOGO_AUTEUR})]
```

#LOGO_DOCUMENT

Avril 2010

Affiche l'image-logo associée à un document.

La balise `#LOGO_DOCUMENT` utilisée dans une boucle DOCUMENTS affiche l'image-logo associée au document :

- Soit une icone standard déterminée par le type du fichier [1] ;
- Soit, seulement pour les documents-image et seulement si la « *génération automatique des miniatures des images.* » a été sélectionnée en configuration sur la page `ecrire/?exec=config_fonctions`, une réduction du document image ;
- Soit la vignette personnalisée qui a été associée manuellement au document lui-même.

Notes

[1] par exemple :    ...

#LOGO_SITE_SPIP

Avril 2010

`#LOGO_SITE_SPIP` affiche le logo du site. Cette image peut être changée depuis l'espace privé en cliquant sur "Configuration".

Cette balise renvoie le logo du site (cf. `IMG/siteon0.png`) qu'il ne faut pas le confondre avec le logo par défaut des rubriques, aussi désigné "*logo standard des rubriques*" (cf. `IMG/rubon0.png`) qui, lui, peut être changé en cliquant sur "Édition" depuis l'espace privé.

#MENU_LANG, #MENU_LANG_ECRIRE

Avril 2010

`#MENU_LANG` et `#MENU_LANG_ECRIRE` : ces balises fabriquent et affichent un menu de langues permettant au visiteur d'obtenir la page en cours dans la langue choisie. 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).

#MIME_TYPE

Septembre 2009 — maj : Mars 2010

`#MIME_TYPE` affiche le type MIME du fichier — par exemple : `image/jpeg` —, *cf.* Type de média internet.

#NOM_SITE

Décembre 2008

- **`#NOM_SITE`** et **`#URL_SITE`** affichent le nom et l'url du « lien hypertexte » de l'article (si vous avez activé cette option).

#NOM_SITE_SPIP

mars 2010

`#NOM_SITE_SPIP` affiche le nom du site. À l'installation d'un SPIP vierge, le nom du site par défaut est "Mon site SPIP".
Pour le modifier, vous devez cliquer sur "Configuration" et changer le texte dans le champ "Nom de votre site".

#NOTES

Janvier 2009

- **`#NOTES`** affiche les notes de bas de page (calculées à partir de l'analyse du texte).

#PARAMETRES_FORUM

Janvier 2009

- **#PARAMETRES_FORUM** fabrique et affiche la liste des variables exploitées par le formulaire permettant de répondre à cet article. Par exemple :

```
[<a href="spip.php?page=forum&(#PARAMETRES_FORUM)">Répondre à cet
article</a>]
```

On peut lui passer un paramètre spécifiant l'adresse de retour après avoir posté le message. Par exemple :

```
<a ref="spip.php?page=forum&(#PARAMETRES_FORUM{#SELF})">
Répondre à cet article</a>
```

renverra le visiteur sur la page actuelle une fois que le message a été validé.

#PETITION

Janvier 2009 — maj : février 2010

- **#PETITION** affiche le texte de la pétition si elle existe.

Si elle existe mais que le texte est vide, retourne un espace (une chaîne non vide sans incidence dans une page html). Ceci permet de tester l'existence d'une pétition dans une boucle `(ARTICLES)`.

Exemple tiré de la dist, le squelette `inc-petition.html` n'est inclus que si la pétition de l'article est activée :

```
[(#PETITION|?{' '})<INCLURE{fond=inc-petition}{id_article}>]
```

Exemple plus élaboré : on inclut le squelette `inc-petition.html`, soit s'il y a des signatures, soit s'il y a une pétition. En effet, lorsque l'on supprime une pétition via l'espace privé, afin de clore la possibilité de la signer, le code précédent supprime également la liste des signatures.

```
<BOUCLE_signatures(SIGNATURES){id_article}{0,1}>
    <INCLURE{fond=inc-petition}{id_article}>
</BOUCLE_signatures>
    [(#PETITION|?{' '})
         <INCLURE{fond=inc-petition}{id_article}>
    ]
<//B_signatures>
```

#PLUGIN

Août 2010

Cette balise permet, soit de tester la présence d'un plugin actif sur le site, soit d'accéder aux informations contenues dans le fichier plugin.xml si le plugin est actif (aucune information n'est fournie si le plugin est inactif).

- `#PLUGIN{prefixe_du_plugin}` renvoie `true` si le plugin est actif.
- `#PLUGIN{prefixe_du_plugin, information_demandee}` renvoie l'information demandée sur le plugin comme sa version, son nom, sa description...
- `#PLUGIN{prefixe_du_plugin, tout}` renvoie toutes les informations du plugin sous forme d'un tableau associatif qui pourra être affecté à une variable pour être affiché en utilisant le filtre |table_valeur.

Le préfixe du plugin est le champ écrit dans la balise XML `<prefix>` du fichier `plugin.xml`, fichier qui se trouve toujours à la racine du répertoire contenant les scripts du plugin.

L'utilisation la plus fréquente de la balise `#PLUGIN` consiste à tester l'état d'activité du plugin pour afficher un code HTML :

- `[(#PLUGIN{prefixe_du_plugin}|oui) blablabla...]` affichera "blablabla..." si le plugin `prefixe_du_plugin` est activé.

#POPULARITE

Décembre 2008

- **`#POPULARITE`** affiche le pourcentage de popularité de cet article ; voir la documentation « La « popularité » des articles ».

#PUCE

avril 2010

`#PUCE` affiche une puce (pas complexe à comprendre) image présente à la racine de la dist. Vous pouvez personnaliser cette puce-image en créant un fichier "puce.gif" et le placer dans un dossier "squelettes".

#PS

Décembre 2008

- **#PS** affiche le post-scriptum.

#REM

Mars 2010

La balise #REM ne produit aucun affichage : elle permet de commenter le code des squelettes.

La balise #REM permet de commenter le code des squelettes :

```
[(#REM) Ceci est une remarque ]
```

La balise et sa partie optionnelle sont purement et simplement supprimées lors du calcul du squelette. [(#REM)] n'apparaît pas dans le code HTML du site public et ne produit donc aucun affichage.

Attention toutefois aux **effets pervers** de #REM : la partie optionnelle d'une balise ne devant pas contenir de boucle, l'écriture ci-dessous

```
[(#REM) essai de boucle: ne pas afficher !
<BOUCLE_a(ARTICLES){...}{0,1}> #ID_ARTICLE - #TITRE </BOUCLE_a>
]
```

affichera (*exemple*) :
[() essai de boucle : ne pas afficher ! 18 - Mon titre d'article]

#SELF

Décembre 2009 — maj mars 2010

#SELF retourne l'URL de la page appelée, nettoyée des variables propres à l'exécution de SPIP.

#SELF retourne l'URL de la page courante. Les variables propres à l'exécution de SPIP (par exemple, var_mode) sont supprimées.

Exemple : pour une page avec l'url spip.php?article25&var_mode=recalcul, la balise #SELF placée dans article.html retournera spip.php?article25.

Usages

- Certaines balises de formulaires acceptent un paramètre pour préciser sur quelle page le visiteur doit être redirigé après avoir validé le formulaire. #SELF peut alors être utilisé pour qu'il revienne sur la page courante. Exemple : [(#FORMULAIRE_FORUM{#SELF})]

- Avec le filtre |parametre_url, pour ajouter des variables dans l'URL de la page courante. Exemple : [(#SELF|parametre_url{'id_mot','3'})] ajoute id_mot=3 à l'URL courante. Voir |parametre_url pour plus de détails.

- #SELF est utile dans les formulaires, pour que l'utilisateur revienne à la page courante après validation :

```
<form action="#SELF">
[(#SELF|form_hidden)]
...
</form>
```

Le filtre |form_hidden calcule les champs cachés (*hidden*) du formulaire à partir des arguments de #SELF. Voir |form_hidden.

#SELF dans les squelettes inclus

Lorsque l'un des paramètres de l'URL de la page doit être récupéré dans un squelette inclus, #SELF doit être ajouté comme paramètre de l'INCLURE. De cette façon, un cache différent du squelette inclus sera créé pour chaque URL (Ceci constitue une sécurité contre une potentielle attaque *XSS)*.

- <INCLURE{fond=mon_squelette}{self=#SELF}>
- <INCLURE{fond=mon_squelette}{self}>
- <INCLURE{fond=mon_squelette}{env}>

Ce paramètre {self=#SELF} doit également être passé à un INCLURE lorsque l'on veut utiliser la balise #PAGINATION dans un squelette inclus, étant donné que celle-ci repose sur la fonction self pour retrouver la variable debut_....

#SESSION

Janvier 2009 — maj : mars 2010

`#SESSION`, permet d'accéder aux informations liées au visiteur authentifié et de différencier automatiquement le cache en fonction du visiteur.

`#SESSION` permet d'accéder à certaines informations propres au visiteur.
Les informations pour un visiteur authentifié sont :

- **id_auteur** : numéro interne unique de l'auteur
- **nom** : nom de signature de l'auteur
- **bio** : courte biographie sur l'auteur
- **email** : l'adresse mail de l'auteur
- **nom_site** : nom/titre du site internet de l'auteur
- **url_site** : adresse http du site internet de l'auteur
- **login** : identitifiant de connexion
- **statut** : 0minirezo (administrateur ou administrateur restreint), 1comite (rédacteur), 6forum (visiteur)
- **maj** : date et heure de la dernière modification des données de l'auteur (toute modification sur les informations de l'auteur contenues dans spip_auteurs telles que la dernière connexion, dernier message interne envoyé, dernière modification de fiche perso...)
- **pgp** : clé pgp publique
- **en_ligne** : date et heure de la dernière connexion (pas celle en cours)
- **imessage** : 'oui' si l'auteur a envoyé un message (privé)
- **messagerie** :
- **prefs** : liste des préférences de l'auteur (essentiellement affichage de l'espace privé, souvenir du cookie...)
  - couleur => 1 à 6, code de la couleur de l'espace privé
  - display => 1 (textes seuls) ; 2 (icones et textes) ; 3 (icones seuls)
  - cnx => vide ou 'perma' (si « *connecté plusieurs jours* » a été coché)
- **cookie_oubli** : hash du cookie de remplacement envoyé si oubli, puis vide
- **source** : 'spip' ou 'ldap'
- **lang** : langue utilisée dans l'espace privé
- **extra** : liste des champs extra déclarés pour l'auteur
- **auth** : type d'authentification utilisée (spip, ldap)
- **cookie** : 'oui' si connecté via cookie - inexistant si connecté autrement (dont PHP_AUTH_USER)
- **hash_env** : code interne (hash) identifiant de façon unique la session du visiteur
- **ip_change** : FALSE tant que l'IP du connecté ne change pas. TRUE si l'IP du connecté change. SPIP recrée alors une nouvelle session pour le primo connecté (avec ip_change à FALSE) ; cette nouvelle session déconnectant l'éventuel voleur de cookie.

Pour afficher une information si et seulement si le visiteur est administrateur :

- `[(#SESSION{statut}|=={0minirezo}|oui)  Vous êtes administrateur, restreint ou non]`

De la même manière, vous pouvez réserver l'affichage d'un champ selon le statut :

- `[(#SESSION{statut}|=={0minirezo}|oui)  #DESCRIPTIF]`

De même on peut complexifier le cas, en chargeant un squelette nommé `reservee.html` :

- `[(#SESSION{statut}|=={0minirezo}|oui)  <INCLURE{fond=reservee, env}>]`

Cette balise se complète assez bien avec #AUTORISER.

#SESSION_SET

Mai 2009— maj : mai 2010

Permet d'insérer dans la balise #SESSION des données supplémentaires

`#SESSION_SET` s'utilise ainsi on a `#SESSION_SET{variable, valeur}`, la valeur sauvegardée dans le tableau php sous la forme
`$GLOBALS['visiteur_session']['variable'] = 'valeur';`

L'information peut être à nouveau lue via `#SESSION{variable}`

Cette balise permet donc d'ajouter une donnée quelconque à la session ouverte lors de l'accès d'un internaute à Spip. Elle peut être utilisée sans modération pour conserver des éléments dont on souhaite disposer tout au long de la session ouverte, sans avoir besoin de refaire une boucle ou une autre manipulation.

La syntaxe

La syntaxe de base est très simple : `#SESSION_SET{variable,valeur}` L'élément 'variable' est un nom quelconque qui permettra de faire référence à ce que l'on veut utiliser, et 'valeur' est son contenu. Mais un exemple vaut un long discours.

Par exemple, je suis passionné par les éléphants, et je voudrais l'afficher à tout moment ... Donc : `#SESSION_SET{mapassion,éléphants}` mettra 'éléphants' dans 'ma passion'.

Pour l'utiliser, c'est très simple : `#SESSION{mapassion}` affichera 'éléphants'.

Utilisation pratique

Vous n'avez vu ci-dessus qu'un exemple pas forcément très concret. L'utilisation de la balise `#SESSION_SET{variable,valeur}` est particulièrement pertinent dès lors qu'un accès à un site se fait avec identification (du type 'login public') afin d'ajouter certaines données liées d'une façon ou d'une autre, dans la session de la personne loguée.

Si vous affichez la balise `#SESSION` sur une page, vous verrez qu'elle contient un grand nombre de données, notamment concernant l'auteur ou visiteur identifié. L'affichage d'une de ces données se fait directement via #SESSION, sauf si vous avez ajouté des champs extra.

Dans `#SESSION{extra}`, vous verrez s'afficher, en format **texte** le contenu de ce champ, sous la forme (par ex.)

```
a:2{s:10:"id_salarie";s;1:"1";s:7:"college";s:5:"Cadre";}.
```

Cela vous permet bien sûr de vérifier qu'il y a bien ce que vous cherchez, mais l'affichage laisse à désirer ... [1]

Que faire pour remédier à cela ? Utiliser #SESSION_SET bien sûr, avec la balise #EXTRA !!

Je voudrais afficher le collège :

Dans une boucle AUTEURS, je vais ajouter à ma session le collège de la personne identifiée :

```
<BOUCLE_auteur(AUTEURS){id_auteur=#SESSION{id_auteur}}{tout}>#SESSION_SET{c
ollege,(#EXTRA|college)}
</BOUCLE_auteur>
```

Et voilà ! Maintenant, sur n'importe quelle page du site, affichez le collège :

```
[(#SESSION{college})]
```

Vous pouvez l'utiliser tout seul, ou dans une boucle, soit pour afficher le collège, soit dans un critère spécifique, par ex.

```
<BOUCLE_articles(ARTICLES) {id_rubrique}{titre_mot = #SESSION{college}}
{par num titre, date} {doublons} {pagination 20}>
```

Aller plus loin ?

Rappelons-nous que spip 2.0.x permet maintenant d'ouvrir des connexion à des bases de données extérieures à celles de spip, même distantes ... Quoi de plus simple, maintenant, d'ajouter une ou des données issues de ces bases à la session ouverte ... Une (ou plusieurs)boucles à l'ouverture, quelques #SESSION_SET*variable,valeur*, et j'ai à ma disposition, rapidement, tout ce que dont j'ai besoin pour dynamiser mon site ...

Notes

[1] rappel : pour afficher en mode *brut* (pour vérification) le contenu détaillé de la balise, il est possible d'utiliser l'écriture :
`[<pre>(#SESSION*|unserialize|var_export{1})</pre>]`

#SET et #GET

Décembre 2009 — maj : mars 2010

La balise `#SET` définit une variable qui sera utilisable dans l'ensemble du squelette.
La balise `#GET` permet ensuite de récupérer cette variable.

`#SET{variable,valeur}`
`#GET{variable}` retourne "valeur".

La variable définie par `#SET` peut être une chaîne de caractères ou un tableau.

- Une *chaîne de caractères* :

```
#SET{ma_chaine, article de #NOM}
```

`#GET{ma_chaine}` affichera :
article de Toto

- Un *tableau* :

```
#SET{mon_tablo, #ARRAY{a,un,b,deux,c,trois,d,quatre}}
```

`[(#GET{mon_tablo}|foreach)]` affichera :

- a => un
- b => deux
- c => trois
- d => quatre

Usages avancés avec les chaînes de caractères

- Une *expression régulière* :

```
#SET{reg1, '(^.*/[^/]+[/])'}
#SET{reg2, '\.[\w-_]+?'}
<BOUCLE_def(DOCUMENTS){tout}>
[(#FICHIER|replace{#GET{reg2}}|replace{#GET{reg1}})]<hr />
</BOUCLE_def>
```

permet de récupérer et d'afficher le nom du fichier d'un document, débarrassé de son chemin d'accès (*path*) et de son extension.
Par exemple, si `#FICHIER` est *IMG/pdf/Stats_Sejour_au_311207.pdf* nous afficherons *Stats_Sejour_au_311207*.

L'intérêt ici d'utiliser les balises `#SET` et `#GET` pour l'expression régulière est de permettre l'utilisation des crochets ([et]) dans le filtre `|replace`.

- Le *résultat* retourné par un `INCLURE` :

```
[(#SET{mon_retour, #INCLURE{fond=calculs}})]
```

Valeur par défaut

Si `variable` n'a pas été définie par un `#SET{variable, valeur}` préalable, il est possible de préciser une valeur par défaut lors de l'insertion de la balise #GET :
`#GET{variable, valeur_par_defaut}` retournera « valeur_par_defaut, » si « variable » n'a pas été définie auparavant.

Attention : cette méthode *n'attribue* pas « valeur_par_defaut » à « variable ».

Portée de la variable définie par #SET

Les valeurs définies par la balise `#SET` restent locales au squelette où elles ont été définies.

Ainsi, la valeur d'une variable définie par un SET *dans un fichier inclu* ne pourra pas être récupérée par le GET correspondant *dans le fichier incluant*.

#SOUSTITRE

Décembre 2008

- **`#SOUSTITRE`** affiche le soustitre.

#SQUELETTE

Mars 2009

`#SQUELETTE` retourne le chemin du squelette courant.

#SURTITRE

Décembre 2008

- **`#SURTITRE`** affiche le surtitre.

#TAILLE

8 mars

`#TAILLE` affiche le poids en octets du document. Pour de gros fichiers, cette valeur devient rapidement inutilisable ; on pourra donc lui appliquer le filtre `|taille_en_octets`, qui affichera successivement en octets, en kilooctets, ou même en mégaoctets :

```
[(#TAILLE|taille_en_octets)]
```

#TEXTE

Décembre 2008

- **#TEXTE** affiche le texte principal de l'article.

#TITRE

Décembre 2008

- **#TITRE** affiche le titre de l'article.

#TOTAL_BOUCLE

Septembre 2009 — maj : Mars 2010

La balise #TOTAL_BOUCLE retourne le nombre total de résultats affichés par la boucle. On peut l'utiliser dans la boucle, dans ses parties *optionnelles* — avant et après — ou même dans la partie *alternative* après la boucle.
Par exemple, pour afficher le nombre de documents associés à un article :

```
<BOUCLE_art(ARTICLES){id_article}>
  <BOUCLE_doc(DOCUMENTS) {id_article}></BOUCLE_doc>
    [il y a (#TOTAL_BOUCLE) document(s).]
  <//B_doc>
</BOUCLE_art>
```

#TOTAL_UNIQUE

Octobre 2009

La balise **#TOTAL_UNIQUE** affiche le nombre d'éléments fitrés par le filtre |unique.

On utilisera **#TOTAL_UNIQUE** pour afficher le nombre de fois où #BALISE a été filtrée par |unique et **#TOTAL_UNIQUE{famille}** pour afficher le nombre de fois où #BALISE a été filtrée par |unique{famille}.

#TYPE_DOCUMENT

Mars 2010

`#TYPE_DOCUMENT` affiche le type du document.

par exemple, pour un fichier d'extension « ods » et de mime_type « application/vnd.oasis.opendocument.spreadsheet », `#TYPE_DOCUMENT` affichera « opendocument spreadsheet ».

#URL_ARTICLE

Janvier 2009 — maj : avril 2010

La balise `#URL_ARTICLE` affiche l'url *relative* (c'est à dire sans la partie `http://nom_domaine.ext/`) de l'article retourné par une boucle ARTICLES.

Cette balise permet ainsi d'écrire des liens : `<a href="#URL_ARTICLE"> ... </a>` indépendamment du choix de configuration effectué pour le *type d'urls*.
L'affichage retourné pourra être :

- spip.php ?article146 *(page)*
- Titre_de_l_article.html *(propres2)*
- article146.html *(html)*
- ...

En cas de changement de type d'urls, la balise affichera toujours la *bonne* url.

En dehors d'une boucle ARTICLES, il est possible de demander l'affichage de l'url d'un article spécifique en passant son id comme argument à la balise : `#URL_ARTICLE{44}` affichera alors l'url de l'article d'id 44.

#URL_AUTEUR

Juin 2010

#URL_AUTEUR affiche l'adresse de la page spip.php ?auteurxxx.

Dans la boucle AUTEURS, #URL_AUTEUR affiche l'adresse de la page de l'auteur : `spip.php?auteurxxx` (où xxx est le numéro d'identifiant de l'auteur).

Exemple de boucle AUTEURS, qui, dans une boucle ARTICLES, affiche le nom de l'auteur de l'article avec un lien vers sa page :

```
<BOUCLE_auteur(AUTEURS) {id_article}>
 <a href="#URL_AUTEUR">#NOM</a>
</BOUCLE_auteur>
```

Exemple de boucle AUTEURS qui affiche le logo de l'auteur aligné à gauche, avec un lien vers la page de l'auteur [1] :

```
[(#LOGO_AUTEUR{#URL_AUTEUR,left})]
```

Notes

[1] avant spip 2.1 on écrivait : `[(#LOGO_AUTEUR|left|#URL_AUTEUR)]`

#URL_DOCUMENT

Mars 2010

`#URL_DOCUMENT` affiche l'URL relative (depuis IMG/) du document.

Associée à `#LOGO_DOCUMENT`, cette balise génère un lien cliquable : `[(#LOGO_DOCUMENT|#URL_DOCUMENT)]` affichera une vignette cliquable pointant vers le document.

#URL_RUBRIQUE

avril 2010

La balise `#URL_RUBRIQUE` affiche l'url *relative* (c'est à dire sans la partie `http://nom_domaine.ext/`) de la rubrique retournée par une boucle RUBRIQUES ou ARTICLES.

Cette balise permet ainsi d'écrire des liens : `<a href="#URL_RUBRIQUE"> ... </a>` indépendamment du choix de configuration effectué pour le *type d'urls*.
L'affichage retourné pourra être :

- spip.php ?rubrique18 *(page)*
- -Titre_de_la_rubrique-.html *(propres2)*
- rubrique18.html *(html)*
- ...

En cas de changement de type d'urls, la balise affichera toujours la *bonne* url.

En dehors d'une boucle RUBRIQUES ou ARTICLES, il est possible de demander l'affichage de l'url d'une rubrique spécifique en passant son id comme argument à la balise : `#URL_RUBRIQUE{12}` affichera alors l'url de la rubrique d'id 12.

#URL_SITE_SPIP

Mars 2010

`#URL_SITE_SPIP` affiche l'adresse du site. Elle ne comprend pas le / final, ainsi vous pouvez créer un lien du type `#URL_SITE_SPIP/#URL_ARTICLE`
Pour connaître l'url de votre site, vous devez cliquer sur "Configuration" et vous verrez cette information inscrite dans le champ "URL de votre site".

#URL_PAGE

Août 2010

`#URL_PAGE` retourne une url de type « page » (cf. Utiliser des URLs personnalisées), vers la page passée en paramètre et qui pourra être utilisée dans un lien.
Par exemple, pour accéder à la page générée par le squelette `toto.html`, située dans votre dossier "squelettes", `#URL_PAGE{toto}` généra automatiquement l'url `spip.php?page=toto`.
Un second paramètre est autorisé pour ajouter des paramètres à l'url. Exemple `#URL_PAGE{toto,id_article=#ID_ARTICLE}` générera l'url `spip.php?page=toto&id_article=XXX`.

Si vous souhaitez transmettre davantage de paramètres, il faudra préférer la syntaxe suivante : `[(#URL_PAGE{message}|parametre_url{'id_article',#ID_ARTICLE}|parametre_url{'lang',#LANG})]` qui donnera `spip.php?page=message&id_article=XXX&lang=YY`.

#URL_SITE, #NOM_SITE

Décembre 2008

- **#NOM_SITE** et **#URL_SITE** affichent le nom et l'url du « lien hypertexte » de l'article (si vous avez activé cette option).

#VAL

Juin 2009 — maj : Décembre 2009

La Balise #VAL prend un paramètre qui est retourné tel quel.

Elle permet par exemple d'appliquer des filtres sur des constantes.

Par exemple `[(#VAL{<:chaine_de_langue:>}|image_typo)]` (qui est équivalent à `<:chaine_de_langue|image_typo:>`)

ou encore

Si vous désirez attribuer une feuille de styles (.css) par jour de la semaine, vous pouvez procéder de la manière suivante :

```
[<link rel="stylesheet"
href="squelettes/styles/jour_(#VAL{w}|date{#DATE|strtotime}).css"
type="text/css" />]
```

Des feuilles de styles nommées respectivement `jour_0.css` à `jour_6.css` (traduire : jour_dimanche.css à jour_samedi.css)

#VISITES

Décembre 2008

- **#VISITES** affiche le nombre total de visites sur cet article.

Critères

{branche}

Décembre 2008

- **`{branche}`** sélectionne l'ensemble des articles de la rubrique ET de ses sous-rubriques. (C'est une sorte d'extension du critère `{id_secteur}`. Toutefois, à l'inverse de `{id_secteur=2}`, il n'est pas possible d'appeler directement une *branche* en faisant par exemple `{branche=2}` : techniquement parlant, il faut que la rubrique en question figure dans le contexte courant. Ce critère est à utiliser avec parcimonie : si votre site est bien structuré, vous ne devriez pas en avoir besoin, sauf dans des cas très particuliers.)

{collecte}

Octobre 2009

Le critère `{collecte}` permet de spécifier l'interclassement (la « *collation* ») pour la requête générée par une boucle (appel de la clause « *COLLATE* » de MySQL [1]).

Le critère `{collecte}`, permet de forcer la requête sql, générée par la boucle à laquelle il est passé, à utiliser un interclassement spécifique pour la clause « *ORDER BY* » (critère `{par ...}`).

Exemple

Supposons une base en utf-8 avec ses tables en utf8 (DEFAULT CHARSET = utf8)

dans ce jeu de caractère (CHARACTER_SET_NAME) il est possible d'utiliser 21 *collations* (interclassements) différentes [2] :

COLLATION_NAME	CHARACTER_SET_NAME
utf8_bin	utf8
utf8_czech_ci	utf8
utf8_danish_ci	utf8
utf8_esperanto_ci	utf8
utf8_estonian_ci	utf8
utf8_general_ci	utf8
utf8_hungarian_ci	utf8
utf8_icelandic_ci	utf8
utf8_latvian_ci	utf8
utf8_lithuanian_ci	utf8
utf8_persian_ci	utf8
utf8_polish_ci	utf8

COLLATION_NAME CHARACTER_SET_NAME

utf8_roman_ci	utf8
utf8_romanian_ci	utf8
utf8_slovak_ci	utf8
utf8_slovenian_ci	utf8
utf8_spanish2_ci	utf8
utf8_spanish_ci	utf8
utf8_swedish_ci	utf8
utf8_turkish_ci	utf8
utf8_unicode_ci	utf8

À partir de là, imaginons que dans la table « *spip_articles* » de cette base, table par défaut en interclassement *utf8_general_ci*, se trouvent les articles titrés :

Nana
nina
Nina
Niña
Ñiña
ñina
Ñina
Nono
Nunu

La boucle :

```
<BOUCLE_a(ARTICLES) {par titre} {"<br />"}>
 #TITRE
</BOUCLE_a>
```

retournera ces articles dans l'ordre alphabétique correspondant à l'interclassement par défaut de la table :

Nana
nina
Ñina
Nina
Niña
Ñiña
ñina
Nono
Nunu

La boucle :

```
#SET{collation, utf8_spanish_ci}
<BOUCLE_a(ARTICLES) {par titre} {collecte #GET{collation}} {"<br />"}>
 #TITRE
</BOUCLE_a>
```

retournera, elle, ces articles dans l'ordre alphabétique correspondant à l'interclassement demandé :

Nana
nina
Nina
Niña
Nono
Nunu
Ñina
ñina
Ñiña

Ainsi les **ñ** (*n à tilde)* seront classés après les **n** (*n nus*).
Cela correspond en effet à l'ordre alphabétique *castillan* :

- « Le **ñ** est la quinzième lettre de version castillane de l'alphabet latin, *entre le N et le O* dans l'ordre alphabétique. »

La requête MySQL produite est alors :

```
 SELECT articles.titre, articles.lang
     FROM spip_articles AS `articles`
   WHERE articles.statut = 'publie'
       AND articles.date < '9999-12-31'
ORDER BY articles.titre COLLATE utf8_spanish2_ci
```

Limite

Il n'est pas possible de faire fonctionner le critère `{collecte}` en lui affectant « *directement* » sa valeur dans la liste des arguments de boucle : les `{collecte utf8_spanish_ci}`, `{collecte 'utf8_spanish_ci'}`, `{collecte{utf8_spanish_ci}}`... ne fonctionneront pas et afficheront une erreur php.
Il n'est possible de l'utiliser que par `#GET{}` après l'avoir ***préalablement*** déclaré en `#SET{}`

Notes

[1] Voir la documentation officielle pour la clause « COLLATE » de Mysql : http://dev.mysql.com/doc/refman/5.0... et : Un exemple de l'effet de collation

[2] Dans le jeu de caractère latin1, 8 collations sont accessibles :

COLLATION_NAME	CHARACTER_SET_NAME
latin1_bin	latin1
latin1_danish_ci	latin1
latin1_general_ci	latin1
latin1_general_cs	latin1
latin1_german1_ci	latin1
latin1_german2_ci	latin1
latin1_spanish_ci	latin1
latin1_swedish_ci	latin1

{critère ?}

Septembre 2009

Un **critère conditionnel** *(critère associé à l'opérateur logique* ***?*** *)* ne sera pris en compte par sa boucle que si les données requises par ce critère sont présentes dans l'environnement d'exécution de la boucle.

```
- <BOUCLE_abc(ARTICLES) {id_rubrique ?}>
```

- S'il existe un id_rubrique *(nn)* dans l'environnement d'exécution de la boucle, alors la boucle ne sélectionnera que les données correspondantes à l'égalité **`id_rubrique = #ENV{id_rubrique}`**
- S'il n'existe pas d'id_rubrique dans l'environnement d'exécution de la boucle, le critère sera totalement **ignoré**.

- On notera que `{id_rubrique ?}` est équivalent à `{id_rubrique?}`.

{critère IN valeur1, valeur2[, valeur3,..., valeurN]}

Août 2009 — maj : avril 2010

`{... IN ...}` limite l'affichage aux résultats ayant un critère appartenant à un ensemble fini.

`{xxxx IN a,b,c,d}` limite l'affichage aux résultats ayant le critère xxxx égal à a, b, c ou d.

Les résultats sont triés dans l'ordre indiqué (sauf demande explicite d'un autre critère de tri).
Il est aussi possible de sélectionner des chaînes de caractères, par exemple avec `{titre IN 'Chine', 'Japon'}`

Il est également possible de passer en argument un tableau.

Il peut s'agir d'un tableau défini par une balise #ARRAY ou bien d'un tableau provenant d'une balise `#ENV**{mon_post}`.

Si `#ENV{mon_post}` est un tableau (venant par exemple de saisies de formulaire dont l'attribut name se termine par `[]`), et si les filtres d'analyse ont été désactivés en suffixant cette balise par une double étoile, alors chaque élément du tableau sera considéré comme argument de IN, SPIP appliquant les filtres de sécurité sur chacun d'eux [1].

Le squelette standard formulaire_forum_previsu fournit un exemple d'utilisation avec une boucle MOTS ayant le critère `{id_mot IN #ENV**{ajouter_mot}}` :
cette boucle sélectionne seulement les mots-clés appartenant à un ensemble indiqué dynamiquement. Ici, cet ensemble aura été construit par le formulaire du squelette standard « choix_mots », qui utilise l'attribut `name=ajouter_mot[]`.

Utilisation conditionnelle de l'opérateur IN dans un critère de boucle

exemple d'utilisation :

- si #ID_ARTICLE est présent dans l'environnement et quelle que soit sa valeur :

```
<BOUCLE_a(ARTICLES) {id_article ?IN 11,12,13}>
```
n'affichera que les articles dont l'id **est dans** la liste IN

```
<BOUCLE_a(ARTICLES) {id_article ?!IN 11,12,13}>
```
n'affichera que les articles dont l'id **n'est pas dans** la liste IN [2]

- si #ID_ARTICLE n'est pas présent dans l'environnement :

```
<BOUCLE_a(ARTICLES) {id_article ?IN 11,12,13}>
```
affichera **tous** les articles **y compris** ceux dont l'id est dans la liste IN

```
<BOUCLE_a(ARTICLES) {id_article ?!IN 11,12,13}>
```
affichera **tous** les articles **y compris** ceux dont l'id est dans la liste IN [2]

Notes

[1] *voir aussi la page de la balise # ENV*

[2] pour rappel : `{id_article !IN 11,12,13}` sélectionne les articles dont l'id n'est pas dans la liste 11, 12, 13 (voir *critère !opérateur valeur*)

{critère LIKE valeur}

Août 2009

Le critère `{... LIKE ...}` correspond au critère de comparaison SQL LIKE.

Par exemple, pour rechercher les AUTEURS dont le login *commence* par paul :
`{login LIKE paul%}`

Nous pouvons aussi utiliser une balise SPIP dans le critère. Dans ce cas vous devez penser à l'entourer de parenthèses.

Par exemple, pour rechercher les email *contenant* la valeur de la balise `#ENV{email}` :
`{email LIKE %(#ENV{email})%}`

Autre exemple, pour trouver les email qui ne *contiennent pas* le critére de recherche, il est possible de faire :
`{!email LIKE %(#ENV{email})%}`

{critère ?opérateur valeur}

Mars 2010

Prendre en compte le critère uniquement si ***pour ce critère*** une valeur est présente dans l'environnement.

Le critère ne sera pris en compte par la boucle que si une variable de **même nom** est présente dans l'environnement.
- par exemple :

```
{date ?!= #ENV{date}},
{email ?LIKE %(#ENV{email})%}
```

- ou encore :

- `<BOUCLE_abc(ARTICLES) {titre ?<= #ENV{titre}}>`
    - *S'il existe une variable de nom `titre` dans l'environnement d'exécution de la boucle, alors la boucle ne sélectionnera que les données correspondantes à `titre <= #ENV{titre}` (les articles dont le titre est alphabétiquement placé avant le titre présent dans l'environnement)*
    - *S'il n'existe pas de `titre` dans l'environnement d'exécution de la boucle, le critère sera totalement ignoré (tous les articles seront sélectionnés).*

- On notera que si peu importe la présence ou non d'espaces avant ou après l'opérateur (*`{titre ?<= #ENV{titre}}` est équivalent à `{titre?<=#ENV{titre}}`*), le **`?`** doit être **collé** à l'opérateur.

{ !critère opérateur valeur}

Août 2009

Opérateur logique du Complémentaire, indique de prendre le complément à l'opération.

Cette écriture est principalement utile pour les boucles faisant intervenir des jointures multiples entre tables : par exemple mots-clé/articles ou documents/articles...
Elle permet de sélectionner non seulement les données ne répondant pas à l'équation mais

aussi celles qui ne sont pas concernées par le critère (une table dont la jointure avec une autre table est inexistante).

`{!titre_mot = 'X'}` : retourne toutes les données sauf celles liées au mot-clé 'X' ; c'est à dire **y compris les données liées à aucun mot-clé**.

`{!id_rubrique < 12}` : retourne les données des rubriques dont l'id `#ID_RUBRIQUE` n'est pas strictement inférieur à 12 (*donc qui est supérieur ou égal à 12*).

`{!id_auteur IN 1, 2}` : retourne toutes les données dont l'id auteur n'est ni 1, ni 2.

L'exemple suivant est plus complexe, il indique qu'il ne faut pas prendre en compte les auteurs dont le login contient la chaîne `'utilisateur'`.

```
{!login LIKE %utilisateur%}
```

{critère !opérateur valeur}

Août 2009

Opérateur logique de la négation ; il inverse le sens de l'opérateur qui suit.

`{id_rubrique != 12}` : retourne les données des rubriques dont l'id `(#ID_RUBRIQUE)` *n'est pas égal* à 12.

`{titre !== ^(a|e|i|o|u|y)}` : retourne les données de l'objet (article, rubrique...) dont le titre `(#TITRE)` *ne commence pas* par une voyelle (insensible à la casse).

`{extension !IN gif, jpg, png}` : retourne les données des documents dont l'extension *n'est pas* dans la liste gif, jpg ou png.

{date}

Décembre 2008

- Les critères **`{date}`** (ou `{date=...}` ou `{date==...}`) permettent de sélectionner un article en fonction de la date passée dans l'URL.

{doublons}

6 janvier 2010

`{doublons}` permet d'interdire l'affichage des résultats déjà affichés dans d'autres boucles utilisant ce critère.

historique : A partir de [SPIP 1.2] et jusqu'à [SPIP 1.7.2], seules les boucles ARTICLES, RUBRIQUES, DOCUMENTS et SITES acceptaient ce critère.

Le critère `{doublons}` une fois découvert devient rapidement indispensable à vos squelettes [1]. S'il est d'un emploi un poil curieux, il permet des tris très intéressants mais qui n'apparaissent pas immédiatement à la lecture de la documentation de base sur Spip.net qui nous dit :

Le critère `{doublons}` ou `{unique}` [2] : permet d'interdire l'affichage des résultats déjà affichés dans d'autres boucles utilisant ce critère. Ces deux critères sont rigoureusement identiques.

Identiques ? Pas tout à fait ! Au fil des versions de SPIP, doublons s'est enrichi de possibilités supplémentaires :
- on peut nommer ses doublons, donc en faire coexister plusieurs dans un même code,
- il permet aussi des pirouettes spipiennes avec l'anti-doublons !

Utilisation de base : ne pas retrouver des éléments déjà traités dans la page

Un exemple nous est donné dans le squelette sommaire de dist [3]

```
<BOUCLE_articles_recents(ARTICLES) {par date}{inverse} {0,2} {doublons}>
      éléments à afficher, par exemple #TITRE
</BOUCLE_articles_recents>
<BOUCLE_autres_articles(ARTICLES) {par date}{inverse} {doublons} >
      éléments à afficher, par exemple #TITRE
</BOUCLE_articles>
```

Il s'agit de lister les articles du site par ordre chronologique inversé **ET** de réserver aux deux derniers articles publiés un traitement particulier. Comme vous le voyez, les boucles "_articles_recents" et "_autres_articles" sont construites de la même manière. En toute logique, elles doivent donc retourner la **même** liste d'articles.

C'est grâce au travail du critère "doublons" que les 2 articles les plus récents, déjà traités dans la première boucle, ne se retrouveront pas dans la liste affiché avec la boucle "_autres_articles" .

Autre utilisation courante : exclure des éléments

Notre grand classique : exclure suivant un mot-clé

On voit régulièrement sur la liste spip-users posé ce type de problème :

"Je n'arrive pas a exclure des éléments en fonction de leur mot-clé. j'essaie :

```
<BOUCLE_rubriques(RUBRIQUES) {racine}
{titre_mot!=invisible}{par num titre, titre}>
```

mais cela ne fonctionne pas"

Et pour cause !

Ce que veut l'utilisateur ici, c'est sélectionner toutes les rubriques qui n'ont pas reçu le mot clé "invisible". Or, ce que comprend la base de données avec `{titre_mot != invisible}`, c'est qu'elle doit sélectionner toutes les rubriques qui ONT un mot clé ET que le dit mot clé soit différent de "invisible".

Et cela change tout. Car dans le résultat figurera par exemple une rubrique à laquelle a été affecté un mot clé "bidule", donc différent de "invisible" (ok !), y compris si la rubrique est liée au mot clé "invisible" (arg !) et, n'y figurera pas, une rubrique qui n'a aucun mot clé (l'inverse du résultat souhaité !). [4]

La solution : enchaîner une boucle vide qui sélectionne selon le mot-clé et une autre boucle qui retourne les résultats en utilisant le critère `{doublons}`.

En reprenant notre exemple ci-dessus cela donne :

```
<BOUCLE_exclure(RUBRIQUES) {racine}
{titre_mot=invisible}{doublons}>
</BOUCLE_exclure>
```

Cette boucle sélectionne ***toutes les rubriques qui ont le mot-clé "invisible"*** , mais elle n'affiche **rien**.

```
<BOUCLE_rubriques(RUBRIQUES) {racine}{par num titre, titre}
 {doublons}>
{le traitement  à effectuer ici}
</BOUCLE_rubriques>
```

Cette deuxième boucle va sélectionner grâce au critère doublons toutes ***les autres rubriques*** et leur appliquera le traitement choisi.

Nommer les doublons pour en utiliser plusieurs dans le même fichier

Objectif : gérer sur une page d'accueil l'affichage de liens vers des articles et vers des communiqués. La présentation des deux derniers articles publiés et des deux derniers communiqués est différente des autres.

Pour l'exemple, on retrouve ici le même schéma déjà vu avec les boucles de la DIST. Il s'agit simplement ici de faire cohabiter sans conflit des boucles très proches. Nommer les doublons évitera que les tris de l'une interfèrent dans les tris de l'autre.

```
<BOUCLE_communiques_recents(ARTICLES) {!par date}{id_mot=1}
 {0,2} {doublons com}>
éléments à afficher, par exemple #TITRE
</BOUCLE_communiques_recents>
<BOUCLE_autres_communiques(ARTICLES) {!par date}{id_mot=1} {doublons
com} >
éléments à afficher, par exemple #TITRE
</BOUCLE_autres_communiques>

<BOUCLE_articles_recents(ARTICLES) {!par date} {0,2} {doublons
art}>
éléments à afficher, par exemple #TITRE
</BOUCLE_articles_recents>
<BOUCLE_autres_articles(ARTICLES) {!par date} {doublons art} >
éléments à afficher, par exemple #TITRE
</BOUCLE_articles>
```

De manière générale, nommer ses doublons est une bonne pratique pour éviter tout télescopage actuel mais aussi futur (des squelettes cela évoluent). C'est aussi un élément qui donne de la lisibilité à votre code.

Dans le cadre d'une utilisation avancée, vous pouvez essayer l'utilisation de balises SPIP. Par exemple : `{doublons #TITRE}` ou même `{doublons #_maboubouclе:TITRE}` voire même `{doublons (#_maboubouclе:TITRE|supprimer_numero)}`.

Utilisation avancée : anti-doublons ou comment constituer une pile de données à traiter

Mécanique de l'anti-doublons

Ici "doublons" va permettre de rassembler le résultats de plusieurs boucles utilisant différents critères et " !doublons" d'appliquer à cet empilement d'items les traitements souhaités.

Le schéma d'utilisation est celui-ci :

On sélectionne un première série d'articles (on n'affiche rien)...

```
<BOUCLE0(ARTICLES){id_mot=2}{doublons A}></BOUCLE0>
```

...puis une deuxième série d'articles (on n'affiche toujours rien)....

```
<BOUCLE1(ARTICLES){id_auteur=1}{doublons A}></BOUCLE1>
```

... on trie selon ses besoins et on affiche le tout grâce à l'anti-doublons.

```
<BOUCLE2(ARTICLES){par date}{!doublons A}>#TITRE<br></BOUCLE2>
```

Un Exemple d'anti-doublons

Objectif : faire une boucle qui récupère les articles de toutes les rubriques à l'exception des rubriques 2 et 3, ET pour ce qui concerne les articles de la rubrique 4, seulement ceux de moins de 60 jours.

La solution : il nous faut une première boucle qui ira chercher tous les articles en excluant ceux des rubriques 2, 3 mais aussi 4...

```
<BOUCLE0(ARTICLES) {id_rubrique !IN 2,3,4}{doublons tri1}></BOUCLE0>
```

....puisqu'il faut dédier à cette rubrique une deuxième boucle...

```
<BOUCLE1(ARTICLES) {id_rubrique=4}{age<60}{doublons tri1}></BOUCLE1>
```

...et c'est dans une dernière boucle que l'on affiche grâce à l'anti-doublons les articles sélectionnés triés avec les critères de son choix.

```
<BOUCLE2(ARTICLES){par date}{!doublons tri1}>#TITRE<br></BOUCLE2>
```

P.-S.

Article original : http://www.spip-contrib.net/Le-critere-doublons-sa-mecanique.
Publié sur spip.net par Teddy

Notes

[1] Quelques contributions sur ce site l'utilisent :

- http://www.spip-contrib.net/Comment-exclure-des-articles,
- http://www.spip-contrib.net/Exclure-selon-un-mot-cle,
- http://www.spip-contrib.net/Menu-depliant-tout-Spip,
- http://www.spip-contrib.net/Afficher-les-articles-dans-un,
- http://www.spip-contrib.net/Afficher-5-vignettes-consecutives

[2] Attention, ne pas confondre le critère "unique" et le filtre "unique" dont on peut trouver des applications :

- ici : http://www.spip-contrib.net/Usage-du-filtre-unique-pour
- et là : http://www.spip-contrib.net/Utiliser-le-filtre-unique-pour.

[3] Il s'agit du squelette par défaut de SPIP que l'on trouve dans le dossier /DIST. Rappel : vous ne devez pas toucher à ces fichiers et ranger les fichiers de votre propre squelette dans un dossier /squelettes à créer au même niveau que /DIST, c'est à dire à la racine du site.

[4] ce point est tiré de http://permalink.gmane.org/gmane.comp.web.spip.user/122659 (Merci Cédric)

{fusion champ_sql}

Octobre 2010

Le critère `fusion` permet de regrouper les résultats d'une boucle SPIP selon les valeurs distinctes d'un champ.

Si la boucle sans le critère `{fusion champ_sql}` ramène plusieurs enregistrements ayant la même valeur de `champ_sql`, alors, la boucle avec le critère `{fusion champ_sql}` ne présentera qu'un seul de ceux ci.

Pour cela, la requête SQL générée intègre une instruction `GROUP BY`.

Exemple 'pédagogique'

La boucle suivante affichera les titres de tous les articles du site :

```
<BOUCLE_tous(ARTICLES)>
      #TITRE
</BOUCLE_tous>
```

La boucle suivante, avec un critère `fusion` portant sur la rubrique, partira également de tous les articles, mais les regroupera par id_rubrique identique, et ne présentera au final qu'un seul article par rubrique.

```
<BOUCLE_extrait(ARTICLES){fusion id_rubrique}>
      #TITRE
```

```
</BOUCLE_extrait>
```

Exemples ’appliqués’

La boucle suivante affiche les 10 messages de forum les plus récents, triés par date, et émanant d’une IP distincte :

```
<BOUCLE_comment(FORUMS){plat}{par date}{inverse}{0,10}{fusion ip}>
       <INCLURE{fond=inc/intro_commentaire}{id_forum}>
</BOUCLE_comment>
```

Il est également possible d’utiliser des expressions SQL en paramètre du critère. La boucle suivante affichera la liste des années pour lesquelles il y a au moins un article :

```
<BOUCLE_annees_avec_articles(ARTICLES){par date}{inverse}{fusion
YEAR(date)}>
      [(#DATE|annee)]
</BOUCLE_annees_avec_articles>
```

Attention

Un critère de tri `{par xxx}` combiné au critère `{fusion champ_sql}` ne retournera pas forcément les résultats attendus.

- `<BOUCLE_a1(ARTICLES){tout}{par id_article}>` ramène les articles triés par id_article (du 1 au 10000 par exemple).

- `<BOUCLE_a2(ARTICLES){par id_article}{fusion id_rubrique}>` ramène les articles, triés par id_article après regroupement par id_rubrique, mais ces regroupements sont faits en vrac et, a priori, sans tris particuliers !

{id_article}

Décembre 2008

- **`{id_article}`** sélectionne l’article dont l’identifiant est `id_article`. Comme l’identifiant de chaque article est unique, ce critère ne retourne qu’une ou zéro réponse.

{id_auteur}

Décembre 2008

- **`{id_auteur}`** sélectionne les articles correspondant à cet identifiant d’auteur (utile pour indiquer la liste des articles écrits par un auteur).

{id_groupe}

Décembre 2008

- **`{id_groupe=zzzz}`** permet de sélectionner les articles liés à un groupe de mots-clés ; principe identique au `{type_mot}` précédent, mais puisque l'on travaille avec un identifiant (numéro du groupe), la syntaxe sera plus « propre ». [Nota : Ce critère n'est pas (en l'état actuel du développement de SPIP) cumulable avec le précédent `{type_mot=yyyy}`]

{id_mot}

Décembre 2008

- **`{id_mot}`** sélectionne les articles correspondant à cet identifiant de mot-clé (utile pour indiquer la liste des articles traitant d'un sujet donné).

{id_rubrique}

Décembre 2008

- **`{id_rubrique}`** sélectionne les articles contenus dans la rubrique dont l'identifiant est `id_rubrique`.

{id_secteur}

Décembre 2008

- **`{id_secteur}`** sélectionne les articles dans ce secteur (un secteur est une rubrique qui ne dépend d'aucune autre rubrique, c'est-à-dire située à la racine du site).

{lang}

Décembre 2008

- **`{lang}`** sélectionne les articles de la langue demandée dans l'adresse de la page.

{origine_traduction}

Décembre 2008 — maj : novembre 2010

- **{origine_traduction}** sélectionne les articles qui servent de base à des versions traduites (les articles "originaux").

Voir sur **programmer.spip.org** pour des exemples

{recherche}

Décembre 2008 — maj : Janvier 2009

- **{recherche}** sélectionne les articles correspondant aux mots indiqués dans l'interface de recherche (moteur de recherche incorporé à SPIP).

L'article consacré au moteur de recherche et aux balises #DEBUT_SURLIGNE et #FIN_SURLIGNE donne plus d'informations concernant la gestion de l'affichage des résultats.

{titre_mot}, {type_mot}

Décembre 2008 — maj : Février 2009

- **{titre_mot=xxxx}**, ou **{type_mot=yyyy}** sélectionne respectivement les articles liés au mot-clé dont le nom est « xxxx », ou liés à des mots-clés du groupe de mots-clés « yyyy ». Si l'on donne plusieurs critères `{titre_mot=xxxx}` (ou plusieurs `{type_mot=yyyy}`), on sélectionnera ceux qui auront tous ces mots à la fois.

{tous}

Octobre 2009

Le critère `{tous}` est un *alias* du critère `{tout}`.

{tout}

Octobre 2009

Le critère **{tout}** permet de sélectionner les données d'une table comme si aucun autre critère restrictif n'était appliqué.
Ainsi son utilisation relève-t'elle plus de l'*aide-mémoire* pour le webmestre puisque l'on obtient le même résultat en n'indiquant aucun critère.

- **<BOUCLE_x(RUBRIQUES) {tout}>**
    - toutes les rubriques sont sélectionnées, c'est-à-dire les rubriques vides *en plus* des rubriques contenant des éléments publiés.
      On réservera ce choix à des besoins très spécifiques ; en effet, par défaut, SPIP n'affiche pas sur le site public les rubriques qui ne contiennent aucun élément actif, afin de garantir que le site ne propose pas de « culs de sac » (navigation vers des pages ne proposant aucun contenu).
- **<BOUCLE_x(ARTICLES) {tout}>**
    - tous les articles **publiés** *(statut = publie)* sont sélectionnés dans toutes les rubriques. Utile notamment pour afficher les articles les plus récents (dans l'intégralité du site) sur la page d'accueil.
- **<BOUCLE_x(AUTEURS) {tout}>**
    - tous les auteurs **sauf ceux mis à la poubelle** *(statut != 5poubelle)* sont sélectionnés, qu'ils aient écrit un article ou non, qu'ils soient administrateur, administrateur restreint, rédacteur, visiteur ou nouvel inscrit.
- **<BOUCLE_x(DOCUMENTS) {tout}>**
    - tous les documents **sauf ceux en mode vignette** *(mode != vignette)* sont sélectionnés dans l'intégralité du site.
- **<BOUCLE_x(MOTS) {tout}>**
    - tous les mots-clefs sont sélectionnés dans l'intégralité du site.
- **<BOUCLE_x(BREVES) {tout}>**
    - toutes les brèves **publiées** *(statut = publie)* sont sélectionnées dans l'intégralité du site.
- **<BOUCLE_x(FORUMS) {tout}>**
    - tous les forums **publiés** *(statut = publie)* sont sélectionnés dans l'intégralité du site sans prendre en compte leur hiérarchie. Avec ce critère, vous pouvez sélectionner tous les messages quelle que soit leur position dans un thread (dans la limite des autres critères, bien sûr). Cela permet par exemple d'afficher les messages par ordre strictement chronologique, ou de compter le nombre total de contributions dans un forum.

Pour faire sauter les dernières restrictions de sélection (“*where statut=publie*” par exemple) et sélectionner *véritablement* **tous les éléments d'une table**, on peut utiliser la notation `<BOUCLE_yy(prefixe_table)>` où l'on précise en *bas de casse* le nom exact —*préfixe compris*— de la table de recherche (attention : pas de 's' pour spip_forum).
par exemple pour afficher tous les articles quel que soit leur statut (*prepa, prop, publie, poubelle, refuse*) : `<BOUCLE_yy(spip_articles)>`

Noter que ce critère possède un alias : le critère `{tous}`.

{traduction}

Décembre 2008

- **{traduction}** sélectionne les traductions de l'article courant en différentes langues.

{vu}

le critère `{vu = ...}` permet de différencier les documents qui ont déjà été traités par l'objet auquel ils sont rattachés.

Le critère « vu » porte sur une association *document*<->*objet* (car un même document peut être lié à plusieurs objets : article, rubrique, etc.). Aussi est-il mis-à-jour à chaque modification du-dit objet.

Si le document est « intégré » dans un objet (typiquement : `<imgXX>` ou `<docYY|left>` ou encore `<embZZ>` dans le texte d'un article par exemple), alors cette association document-objet sera notée comme « *`vu = oui`* ».

Par exemple, pour afficher le portfolio d'un article, on peut utiliser, à l'intérieur d'une *boucle articles*, une *boucle documents* avec le critère `{vu = non}` : ainsi, s'afficheront dans le portfolio **uniquement** les documents qui n'ont pas déjà été affichés dans le corps de l'article.

Les filtres

|abs_url

Juillet 2009 — maj : Novembre 2009

Le filtre `|abs_url` combine les filtres **`|liens_absolus`** et **`|url_absolue`** et peut donc s'appliquer aussi bien à un texte (`#TEXTE`, `#DESCRIPTIF`, etc.) qu'à une balise d'url (`#URL_ARTICLE`, `#URL_RUBRIQUE`, etc.) et transforme tous les liens en liens absolus (avec l'url complète du site).

|affdate

Juillet 2009

Ce filtre s'applique aux dates

- `|affdate` affiche la date sous forme de texte, par exemple « 13 janvier 2001 ».

[SPIP 1.8] étend la notation de ce filtre. On peut lui passer un paramètre de formatage de la date, correspondant à un format spip (« `'saison'` », etc.) ou à un format de la commande php *date* (« `'Y-m-d'` »). Par exemple :

-
  - `[(#DATE|affdate{'Y-m'})]` affichera numériquement l'année et le mois de la date filtrée séparés par un tiret,
  - la notation `[(#DATE|affdate{'saison'})]` est totalement équivalente à : `[(#DATE|saison)]`.

- Il existe aussi des variantes de *affdate* qui fournissent des raccourcis : `|affdate_court`, `|affdate_jourcourt` et `|affdate_mois_annee`.

|affdate_court

Juillet 2009

Ce filtre s'applique aux dates

- `|affdate_court` affiche le nom du mois et le numéros du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors `|affdate_court` affichera seulement le mois et l'année sans le numéros du jour : « Novembre 2004 ».

`|affdate_court` est une variante du filtre `|affdate`

|affdate_jourcourt

Juillet 2009

Ce filtre s'applique aux dates

- `|affdate_jourcourt` affiche le nom du mois et la valeur numérique du jour, e.g. « 19 Avril ». Si la date n'est pas dans l'année actuelle, alors l'année est aussi affichée : « 1 Novembre 2004 ».

`|affdate_jourcourt` est une variante du filtre `|affdate`

|affdate_mois_annee

Juillet 2009

Ce filtre s'applique aux dates

- `|affdate_mois_annee` affiche seulement le mois et l'année : « Avril 2005 », « Novembre 2003 ».

`|affdate_mois_annee` est une variante de `|affdate`

|afficher_enclosures

Octobre 2009

Dans le cadre d'une syndication, le filtre `|afficher_enclosures` appliqué à la balise #TAGS affichera l'icone du trombone pour les documents joints de l'article syndiqué (plutôt qu'un lien classqiue).
`[(#TAGS|afficher_enclosures)]`

Voir aussi les informations détaillées sur la page de la balise #TAGS

|afficher_tags

Octobre 2009

Dans le cadre d'une syndication, le filtre `|afficher_tags` appliqué à la balise #TAGS affichera pour l'article syndiqué au choix :
les tags/mots-clés : `[(#TAGS|afficher_tags{tag})]`,
la rubrique/catégorie : `[(#TAGS|afficher_tags{directory})]`,
les documents joints/podcast : `[(#TAGS|afficher_tags{enclosure})]`.

(Par défaut `[(#TAGS|afficher_tags)]` est équivalent à `[(#TAGS|afficher_tags{'tag,directory'})]`.)

Plus d'informations détaillées sur la page de la balise #TAGS

|agenda_affiche

Juillet 2009

- 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.

|agenda_connu

juillet 2010

Le filtre `|agenda_connu` teste si son argument est l'un des quatre types de calendriers connus (`jour`, `semaine`, `mois` ou `periode`).

Ce filtre est utilisé par les 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.

|agenda_memo

juillet 2010

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.

|alterner

Octobre 2009

Le filtre `|alterner{a, b, c, ...}` appliqué à une balise numérique (le plus souvent #COMPTEUR_BOUCLE) à l'intérieur d'une boucle affiche à tour de rôle et dans l'ordre chacun de ses arguments à chaque changement de valeur de la balise.

Par exemple, `[(#COMPTEUR_BOUCLE|alterner{white, yellow})]` affichera « white » à la première itération de la boucle, « yellow » à la deuxième, « white » à la troisième, « yellow » à la quatrième, etc. Ainsi, on peut faire une liste d'article qui utilise une couleur différente pour les lignes paires et impaires :

```
<B_lesarticles>
<ul>
<BOUCLE_lesarticles(ARTICLES) {par titre}>
<li style="background: [(#COMPTEUR_BOUCLE|alterner{white,
yellow})];">#TITRE</li>
</BOUCLE_lesarticles>
</ul>
</B_lesarticles>
```

Ce filtre n'est pas binaire : il est possible de lui faire afficher une donnée tous les x changements de valeur ; la boucle suivante affichera les titres d'article par bloc de 4 en changeant de couleur à chaque bloc :

```
<BOUCLE_lesarticles(ARTICLES) {par titre}>
[(#COMPTEUR_BOUCLE|alterner{<p style="color:red;">, '', '', '', <p
style="color:green;">, '', '', ''})]
#TITRE<br>
[(#COMPTEUR_BOUCLE|alterner{'', '', '', </p>})]
</BOUCLE_lesarticles>
[(#TOTAL_BOUCLE|modulo{4}|=={0}|non)</p>]
</B_lesarticles>
```

|ancre_url

avril 2010

Le filtre `|ancre_url{intitulé}` ajoute une ancre à l'url sur laquelle il est appliqué (ou modifie l'ancre déjà existante attachée à cette url).

Exemple `[(#URL_ARTICLE|ancre_url{le_paragraphe})]` produira l'url de l'article suffixée de `#le_paragraphe`.

|annee

Juillet 2009 — maj : Septembre 2009

Le filtre `|annee` affiche numériquement, sur 4 chiffres, l'année de la date sur laquelle il s'applique.
`[(#DATE|annee)]`

|appliquer_filtre

juillet 2010

Pour utiliser un filtre qui est défini dans un plugin, quand on est pas certain que le plugin est installé et activé, il ne faut pas, depuis SPIP 2.1 appeler directement le filtre (`[(#BALISE|fonction_specifique)]`), car cela génèrerait une erreur de compilation, mais passer par un appel à `appliquer_filtre` :

```
[(#BALISE|appliquer_filtre{fonction_specifique})]
```

Les paramètres à passer au filtre `fonction_specifique` sont passés en paramètre supplémentaires :

```
[(#BALISE|appliquer_filtre{fonction_specifique, param1, param2})]
```

|attribut_html

Octobre 2009

Le filtre `|attribut_html` appliqué à une balise de texte traduira le rendu html de celle-ci dans un format utilisable sans dommage dans un attribut html (alt="...", title="...", value="...").

Par exemple, à partir d'une balise #DESCRIPTIF qui retourne normalement le source html suivant : `2 est "supérieur" à 1 & c'est noté : 2 > 1`, pour renseigner l'attribut alt d'une image on utilisera :

```
<img alt="[(#DESCRIPTIF|attribut_html)]"...
```

qui retournera le source html : `<img alt="2 est &quot;sup&#233;rieur&quot; &#224; 1 &#38; c&#39;est not&#233; : 2 &gt; 1"...`

Noter que ce filtre supprime les tags html : `<strong>bonjour</strong> l'été !` devenant : `bonjour l&#39;&#233;t&#233; !`

|balise_img

avril 2010

Le filtre `|balise_img` génère un tag image complet (`<img src="...`) à partir du nom de fichier auquel il est appliqué.

Il est possible de passer une valeur pour un `alt` et une `class` éventuels en fournissant ces deux arguments au filtre :
`|balise_img{val_alt, val_class}`.

Exemple :
```
[(#CHEMIN{monimage.png}|balise_img{paysage, spip_doc})]
```
produira le tag :
```
<img src='le_rep/monimage.png' width='180' height='56" alt='paysage'
class='spip_doc' />.
```

|choixsiegal{valeur, sioui, sinon}

Juillet 2009

`|choixsiegal{valeur, sioui, sinon}` affichera "sioui" si la valeur de la balise est égale à "valeur" ou "sinon" si ce n'est pas égale à la valeur souhaitée.
Exemple :

```
<BOUCLE_blog(ARTICLES) {par date} {inverse} {"<br />"}>
[<hr /><h1>(#ID_ARTICLE|choixsiegal{10, "IMPORTANT", ""})</h1>]
#TITRE ...
</BOUCLE_blog>
```

affichera le mot "IMPORTANT" si l'`ID_ARTICLE` est égal à "10" ou rien s'il n'est pas égal à "10".

|choixsivide{sioui, sinon}

Août 2009

Le filtre `|choixsivide{sioui, sinon}` est équivalent au filtre `|?{sioui, sinon}`. Il prend un ou deux paramètres :
- **sioui** est la valeur à afficher à la place de l'élément filtré si celui-ci est vide.
- **sinon** est la valeur à afficher si l'élément filtré est non vide.

`[(#TEXTE|choixsivide{"pas de texte", #TEXTE})]` affiche « pas de texte » si le champ `#TEXTE` n'a pas été rempli dans la partie privée ; affiche le contenu de `#TEXTE` s'il a été renseigné.

|compacte

Octobre 2009

Le filtre `|compacte` appliqué à un fichier css ou javascript [1] en crée une copie compactée, de taille réduite, où ont été supprimés les commentaires, les déclarations vides, les espaces superflus etc. [2] (pour les fichiers de script Javascript, le filtre utilise la classe *JavaScriptPacker*).

Après avoir archivé ce nouveau fichier dans le répertoire local/ le filtre en retourne le lien`[<link rel="stylesheet" href="(#CHEMIN{spip_style.css}|compacte)" type="text/css" media="all" />]` retournera quelque chose comme : `<link rel="stylesheet" href="local/cache-css/spip_style-compacte-092e.css" type="text/css" media="all" />`

Notes

[1] Le fichier auquel s'applique le filtre doit avoir « .js » ou « .css » comme extension.

[2] pour les fichiers de css, le filtre transforme toutes les url contenues en url absolues

```
#div {
 background : url(../prive/images/fond-gris-anim.gif);  /** image
modifiable **/
 width : 48px;
 height : 48px;
 margin : auto;
}
```

devenant

```
#div{background:url(http://www.domaine.tld/prive/images/fond-gris-
anim.gif);width:48px;height:48px;margin:auto;}
```

|concat{valeur1, valeur2, ...}

Juillet 2009

`|concat{valeur1,valeur2,...}` permet de concaténer plusieurs chaînes.

|couleur_*

11 juin 2010

On pourra consulter l'article « Couleurs automatiques » de la présente documentation.

|couleur_extraire

Le filtre `||couleur_extraire` appliqué à une balise image (logo, image typo...) retourne la valeur RVB hexadécimale d'une couleur présente dans l'image.

Fonctionnement :

Le filtre commence par redimensionner l'image originale en un carré de 20 pixels par 20 pixels, pour que la couleur extraite soit représentative des couleurs réellement présentes dans l'image (en évitant de tomber sur un pixel isolé).

Puis il retourne la valeur de la couleur du pixel situé légèrement au dessus du centre de l'image (par défaut : coordonnées 10, 6).

Il est possible de forcer la sélection d'un pixel particulier, c'est à dire de préciser au filtre dans quelle partie de l'image originale il doit sélectionner sa couleur moyenne.

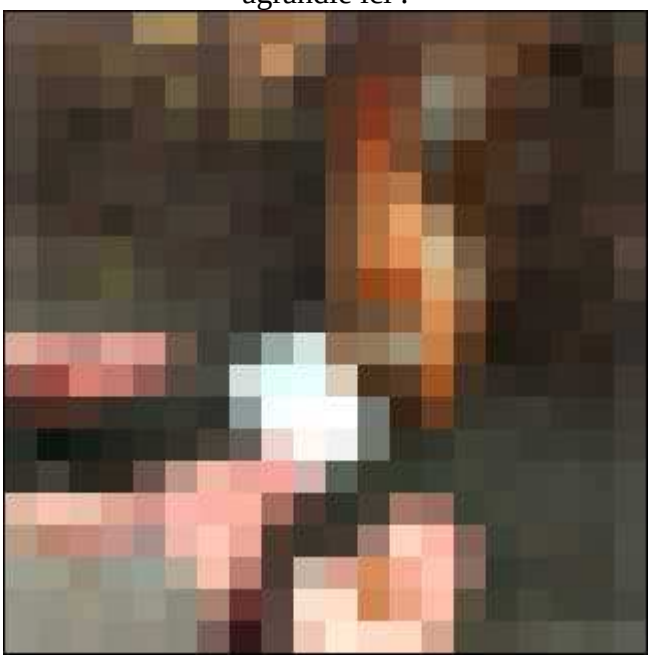
Pour cela on passera au filtre les coordonnées du pixel sous la forme `||couleur_extraire{x, y}` où « x,y » devront se situer dans une fourchette de 0,0 (*coin supérieur gauche*) à 20,20 (*coin inférieur droit*).

Exemple :

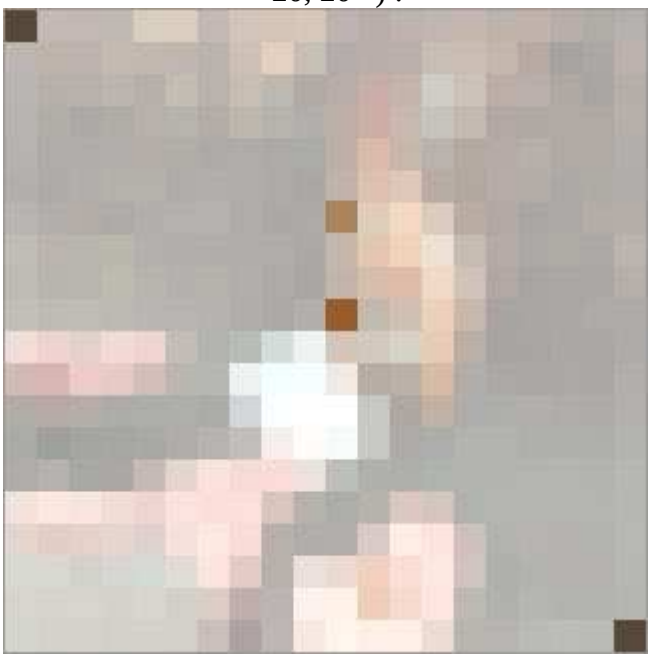
À partir de l'image originale ci-dessous :

le filtre crée une image temporaire réduite :

agrandie ici :



puis sélectionne le pixel spécifié (ci-dessous, respectivement : « 0, 0 » ; « 10, 6 » ; « 10, 10 » ; « 20, 20 ») :



Enfin, le filtre retournera la valeur colorimétrique de ce pixel.
Dans cet exemple :
`554839` pour `||couleur_extraire{0, 0}` ;
`aa8454` pour `||couleur_extraire` (par défaut) ;
`995c2c` pour `||couleur_extraire{10, 10}` ;
`544738` pour `||couleur_extraire{20, 20}`.

Attention : le filtre retourne la valeur *brute* de la couleur ; pour l'utiliser dans un style css, ne pas oublier de faire précéder cette valeur d'un `#`.

|couleur_eclaircir et |couleur_foncer

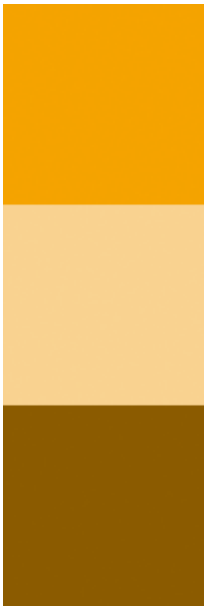
Associés au filtre `|couleur_extraire`, les filtres `|couleur_eclaircir` ou `|couleur_foncer` permettent d'obtenir des déclinaisons (camaïeux) de couleurs, l'ensemble étant très cohérent.

À partir de la couleur extraite d'une image (imaginons par exemple le logo d'une rubrique) :

les 3 codes suivants :

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

retourneront :



Palette

|copie_locale

avril 2010

Le filtre `|copie_locale` crée une copie locale d'un fichier distant (image, texte, pdf, html,...) provenant d'un autre domaine.

Appliqué à une URL, ce filtre tente de copier le fichier sur lequel pointe cette url puis, si l'opération a aboutie, retourne l'url *relative* de la copie locale du ficher (par exemple « *IMG/distant/jpg/l_image.jpg* »). Si l'opération de copie a échouée, le filtre ne retourne rien.

L'URL du fichier que l'on veut copier en local peut
- soit être indiquée explicitement :

`[(#VAL{url_du_fichier_distant}|copie_locale)]` ;
- soit provenir d'une table externe :
`[(#URL_FICHIER_DISTANT|copie_locale)]` ;
- soit provenir d'une balise enclosure d'un flux rss syndiqué :
`[(#TAGS|afficher_tags{enclosure}|extraire_attribut{href}|copie_locale)]`.

Mise à jour de la copie

Le filtre accepte un argument optionnel : `|copie_locale{mode}` qui peut prendre 4 valeurs.
- **auto** (par défaut) :
dans ce cas une copie en local du fichier distant est réalisée dans IMG/distant/. Cette copie n'est effectuée qu'une fois. Elle n'est pas remise à jour au recalcul de la page si le fichier distant a été mis à jour et si son URL est restée identique.
- **force** :
la copie locale est réalisée à chaque *recalcul* de la page. Le fichier local peut donc éventuellement être mis à jour. (À utiliser avec précautions, cette fonction étant consommatrice de ressources).
- **modif** :
la copie locale déjà présente ne sera mise à jour que si elle est antérieure à l'entête « *If-Modified-Since* » du fichier original.
- **test** :
le filtre ne fait que vérifier qu'il existe bien déjà une copie en local du fichier donné en URL et retourne alors l'url de cette copie. Sinon, il ne fait ni ne retourne rien.

Note :
Pour appliquer un filtre image sur un fichier distant, il est indispensable de lui appliquer précédemment le filtre `|copie_locale`.

Exemple :
```
[(#VAL{http://www.spip-
contrib.net/IMG/siteon0.png}|copie_locale|image_reduire{100})]
```

|couper

Septembre 2009

Le filtre `|couper` coupe un texte après un nombre de caractères paramétrable. Il essaie de ne pas couper les mots et enlève le formatage du texte. Si le texte original est plus long que l'extrait coupé, alors « (...) » est ajouté à la fin de l'extrait.
Le filtre coupe par défaut à 50 caractères, mais on peut spécifier une autre longueur en la passant comme paramètre au filtre, par exemple : `[(#TEXTE|couper{80})]`.

De plus, ce filtre nettoie le texte de toutes ses balises html (ou formatage de texte) : il n'y a plus de liens, de gras, de code, de cadre, etc. Nous avons, donc, un texte brut.

Il est possible de personnaliser les points de suite de ce filtre de la manière suivante :

```
[(#TEXTE|couper{80, '...'})]
[(#TEXTE|couper{80, ' <img src="truc.png" alt="">'})]
```

Noter aussi qu'il est fréquent que la coupe ne corresponde pas *exactement* au nombre demandé de caractères (les mots n'étant pas coupés).

|date_822

Septembre 2009

Le filtre `|date_822` formate une balise de type date (`#DATE`, `#DATE_REDAC`, etc.) selon le standard RFC 822 (utilisé pour `<pubdate>` dans certains flux RSS 2.0 — notamment photocast).

Exemple :
la date `2008-10-03 08:22:00` sera affichée `Fri, 03 Oct 2008 08:22:00 +0200`

|date_relative

juillet 2010

Le filtre `|date_relative` appliqué à toute balise `#DATE...` affichera une information liée à la balise selon la date du contexte.

Exemple : nous sommes le vendredi 25 septembre 2009 14h20. Un article a été publié le mercredi 23 septembre 2009 à 10h20.

```
[(#DATE|date_relative)]
```

donnera :
Il y a 2 jours et 4 heures.

A l'inverse, si votre site affiche les articles quelque soit leur date de publication, un article publié le dimanche 27 septembre 2009 à 18h20 affichera ceci :
Dans 2 jours et 4 heures.

|direction_css

Octobre 2009

Le filtre `|direction_css` appliqué à un fichier css (feuille de styles) y remplace toutes les occurrences de *left* par *right* et de *right* par *left* [1].

En fonction de la langue de l'environnement [2] le filtre permet donc d'« inverser » les règles de placement d'un fichier CSS.

Ce filtre commence par chercher un éventuel fichier (inversé) existant et, s'il ne le trouve pas, le crée dans le répertoire *local/cache-css/*

Si le fichier de styles inclu d'autres fichiers de styles placés sur le même domaine (host) (règle `@import url` dans le fichier), le filtre propage ses modifications aux fichiers concernés.

Notes

[1] Attention : le filtre ne traite pas (ne modifie pas) les règles du type : `#div { margin: 10px 15px 10px 50px; }` ; il vous faut donc préciser dans votre fichier de styles original :

`#div { margin-top: 10px; margin-left: 15px; margin-bottom: 10px; margin-right: 50px; }` pour que les inversions puissent y être appliquées.

[2] Il est possible de passer un argument au filtre pour forcer une direction d'écriture voulue : `|direction_css{rtl}` ou `|direction_css{ltr}`.

|div

Septembre 2009

Le filtre `|div{xx}` est un filtre d'opérations mathématiques.
Il retourne le résultat de la division (le *quotient*) de la valeur de la balise par xx.

Si le retour de la balise n'est pas de type numérique, il est considéré comme 0 (zéro) et le filtre retourne 0 (zéro) : `[(#TEXTE|div{18})]` retournera *« 0 »*.

Il est possible de donner un nombre négatif comme argument du filtre :
`[(#TOTAL_BOUCLE|div{-5.2})]`.

Attention, si *xx* est un nombre *à virgule*, sa notation doit utiliser *le point* en lieu et place de la virgule : `[(#TOTAL_BOUCLE|div{5.2})]`.

|entites_html

Novembre 2009

Le filtre `|entites_html` appliqué à une balise de texte traduira le rendu html de celle-ci en transformant en entitées html les caractères en dehors du charset de la page ainsi que les `"`, `<` et `>`.
Ceci permet d'insérer le texte d'une balise dans un `<textarea> </textarea>` sans dommages.

Par exemple, à partir d'une balise #DESCRIPTIF qui retourne normalement le source html suivant : `2 est "supérieur" à 1 & c'est noté : 2 > 1`,

```
<textarea>[(#DESCRIPTIF|entites_html)]</textarea>
```

retournera le source html :

```
<textarea>2 est &amp;quot;supérieur&amp;quot; à 1 &amp;amp; c'est noté : 2 &amp;gt;
1</textarea>
```

|env_to_params

Octobre 2009

Récupère les paramètres passés à un modèle audio ou vidéo et en crée les correspondances pour le tag html `<object>`.

Utilisé dans un squelette de modèle, dans une boucle DOCUMENTS et à l'intérieur d'un tag html `<object>`, ce filtre, spécifiquement appliqué à la balise ENV :

`[(#ENV*|env_to_params)]`, récupère les paramètres passés au modèle (`<emb54|center|autoplay=true|loop=false>`) et crée pour chacun d'eux le tag html `<param value="..." name="..." />` correspondant.

Exemple :
alors qu'un simple appel de modèle `<video386>` produira par défaut :

```
<param value="IMG/mov/mon_film.mov" name="movie"/>
<param value="IMG/mov/mon_film.mov" name="src"/>
<param value="386" name="id_video"/>
<param value="" name="class"/>
```

un appel plus argumenté

```
<video386
     |center
     |border=6
     |autostart=false
     |autoload=false
     |loop=3
     |controls=true>
```

produira :

```
<param value="IMG/mov/mon_film.mov" name="movie"/>
<param value="IMG/mov/mon_film.mov" name="src"/>
<param value="386" name="id_video"/>
<param value="" name="class"/>
<param value="center" name="center"/>
<param value="6" name="border"/>
<param value="false" name="autostart"/>
<param value="false" name="autoload"/>
<param value="3" name="loop"/>
<param value="true" name="controls"/>
```

|et

Mars 2010

Le filtre **`|et`** (peut aussi être écrit **`|and`**) permet de vérifier la présence de 2 éléments.

exemples :

```
#
[(#CHAPO|et{#PS}) Il y a un chapo et un post-scriptum ]
#
[(#CHAPO|et{#PS}|non) Il n'y a pas à la fois un chapo et un post-scriptum ]
```

voir aussi les filtres **`|ou`** et **`|xou`**

|extraire_attribut

Octobre 2009

Le filtre `|extraire_attribut{nom_attribut}` permet de récupérer l'attribut précisé d'un tag html.

Par exemple :
si la balise `#LOGO_SITE_SPIP` génère le code html `<img width="353" height="120" style="height: 120px; width: 353px;" class="spip_logos" alt="SPIP" src="local/cache-vignettes/L353xH120/siteon0-1de24.jpg"/>`,

alors :

- `[(#LOGO_SITE_SPIP||extraire_attribut{src})]` retournera `local/cache-vignettes/L353xH120/siteon0-1de24.jpg`
- `[(#LOGO_SITE_SPIP||extraire_attribut{width})]` retournera `353`
- `[(#LOGO_SITE_SPIP||extraire_attribut{style})]` retournera `height: 120px; width: 353px;`

|extraire_balise

Octobre 2009

Le filtre `|extraire_balise{nom_tag}` permet de récupérer le premier tag html de nom « *nom_tag* » rencontré dans le source html généré par la balise à laquelle il est appliqué.

Exemple :
Nous syndiquons un photoblog qui diffuse systématiquement un petit commentaire suivi d'une photographie. Cette dernière se présente sous la forme d'une balise HTML `<img src="... />`. Une fois ce photoblog syndiqué dans notre site, nous pouvons décider de n'afficher que la photo, sans le commentaire : `[(#DESCRIPTIF|extraire_balise{img})]`

À partir de là tout est possible :

- afficher l'url de l'image :
  `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{src})]`
- afficher la largeur de l'image :
  `[(#DESCRIPTIF|extraire_balise{img}|extraire_attribut{width})]`
- affecter un style particlulier à l'image :
  `[(#DESCRIPTIF|extraire_balise{img}|inserer_attribut{style, border: double red 4px})]`

|fichier

Septembre 2009 — maj : 20 août

Le filtre `|fichier` ne peut s'appliquer qu'à une balise #LOGO. Il retourne le chemin d'accès (le *path*) vers le fichier correspondant au logo associé au document.

exemples : `[(#LOGO_DOCUMENT|fichier)]` peut retourner « *prive/vignettes/pdf.png* » (cas d'une vignette standard de SPIP), ou bien « *local/cache-vignettes/L112xH54/mon_fichier.png* » (cas d'une vignette personnalisée), ou encore « *png/image_1.png* » (pas de vignette).

Attention : bien qu'il s'applique à une balise #LOGO ce filtre ne doit pas être utilisé avec un *double pipe* (`||fichier` ne fonctionnera pas) ;
ne pas confondre ce *filtre* avec la *balise* homonyme #FICHIER.

Évolution : Ce filtre devient obsolète à partir de SPIP 2.1 et on utilisera en remplacement : `#LOGO_DOCUMENT**`.

|find

Décembre 2009

Le filtre `|find` appliqué à un tableau vérifie la présence d'une valeur dans le tableau

Le filtre `|find` permet de vérifier la présence d'une valeur dans le tableau auquel il est appliqué :

```
[(#GET{mon_tableau}|find{la_valeur}|oui) Ceci s'affiche si la valeur
«la_valeur» est bien présente dans le tableau «mon_tableau».]
```

Si `#GET{mon_tableau}` n'est pas un tableau (array au sens PHP), la balise filtrée par `|find` ne retourne rien.

|foreach

Avril 2010

Le filtre `|foreach` utilise le modèle *squelettes-dist/modeles/foreach.html* pour afficher de façon présentable les couples clef=> valeur du tableau auquel il est appliqué.

- Pour `#SET{tablo, #ARRAY{0,a,1,b,2,c,3,d}}`,
`[(#GET{tablo}|foreach)]` retourne :

- 0=> a
- 1=> b
- 2=> c
- 3=> d

- Pour `&tab[]=aa&tab[]=bb&tab[]=cc&tab[]=dd` passé dans l'URL,
`[(#ENV{tab}|foreach)]` retourne :

- 0=> aa
- 1=> bb
- 2=> cc
- 3=> dd

- Si la balise à laquelle est appliquée le filtre `|foreach` est un tableau *linéarisé* (voir les fonctions PHP), il est indispensable de lui appliquer le filtre `|unserialize` après l'avoir *étoilée* préalablement :
par exemple : `[(#ENV*|unserialize|foreach)]`.

- On peut forcer l'utilisation d'un modèle autre que *foreach.html* en appelant :
`[(#GET{tablo}|foreach{mon_modele})]` où *mon_modele.html* est le modèle personnalisé à utiliser et que l'on aura pris soin de placer dans `squelettes/modeles/`.

- Exemple : afficher la liste des noms des plugins actifs utilisés sur le site

Le fichier de modèle `modeles/liste_plug.html` contiendra :

```
[<li>(#ENV{valeur}|strtolower)</li>]
```

et sera appelé par :

```
[<ul>(#PLUGIN*|unserialize|foreach{liste_plug})</ul>]
```

|form_hidden

juillet 2010

Si on fait un formulaire qui utilise comme action un lien comprenant des arguments (par exemple, quand on utilise la balise #SELF avec le type d'url par défaut), il faut remettre ces valeurs dans des champs *hidden* ; cette fonction calcule les champs en question. À utiliser par exemple comme suit :

```
<form action="#SELF">
[(#SELF|form_hidden)]
...
</form>
```

|hauteur

Septembre 2009

Le filtre `|hauteur` appliqué à un élément image (balise LOGO, image typographique, ...) retourne la hauteur calculée de cet élément une fois filtré.
La hauteur retournée est exprimée en pixels.

Exemple : après avoir appliqué le filtre `|image_reduire{250, 500}` à un logo, pour connaître la hauteur réelle de l'image obtenue, on écrira `[(#LOGO_DOCUMENT||image_reduire{250,500}|hauteur)]`.

|heures

Septembre 2009

Le filtre `|heures` affiche l'heure de la date sur laquelle il s'applique.

```
[(#DATE|heures)]
```

L'heure retournée varie de 00 à 24.

Si la date sur laquelle est appliquée le filtre ne contient pas d'information horaire (`[(#VAL{2009-10-24}|heures)]`), le filtre retournera 0 (zéro).

|image_passe_partout

juillet 2010

Alors que `image_reduire` produit la plus petite image tenant dans un rectangle, `image_passe_partout` produit la plus grande image qui remplit ce rectangle

Suivi d'un `image_recadre` avec les mêmes dimensions il permet de produire des vignettes de dimensions imposées, sans déformation

Exemple

```
[(#FICHIER
    |image_passe_partout{70,70}
    |image_recadre{70,70,center}
    |image_aplatir{jpg,ffffff}
    |inserer_attribut{class,spip_logos})]
```

|inserer_attribut

Octobre 2009

Le filtre `|inserer_attribut{attribut, valeur}` permet de modifier ou ajouter un attribut html dans un tag html généré par la balise à laquelle il est appliqué.

Par exemple :
`[(#LOGO_DOCUMENT||inserer_attribut{alt, #TITRE|attribut_html})]` modifie l'attribut « *alt* » du tag html « *img* » généré par la balise #LOGO_DOCUMENT en lui affectant comme valeur le titre du document.

Il est aussi possible de vider un attribut en lui passant une valeur vide :
`[(#DESCRIPTIF|inserer_attribut{class, ''})]`

Notes : Le filtre n'intervient que sur la première occurrence rencontrée de « attribut » ; si aucune occurrence n'est relevée, alors il s'applique sur le premier « tag » rencontré.
ainsi `[(#TEXTE|inserer_attribut{class, perso})]` modifiera le premier attribut « *class* » rencontré dans #TEXTE et *seulement celui-ci* ou bien, si aucun attribut « *class* » n'est présent dans #TEXTE, créera un `class="perso"` dans le premier tag html présent dans #TEXTE (par défaut, dans le `<p>` ouvrant).

|is_null

novembre 2010

Une fonction PHP utilisée comme filtre et qui s'applique à la balise `#ENV`. Elle permet de tester l'existence ou non d'une variable dans l'environnement SPIP.

Il est parfois nécessaire de différencier une variable de « contenu vide » d'une variable « inexistante ». Ce filtre est là pour vous y aider. Il s'applique à la balise `#ENV{blabla}` et permet donc de tester l'existence ou non de la variable "blabla" dans l'environnement du squelette.

```
[(#ENV{blabla}|is_null)]
```

Ce code retourne "1" si la variable "blabla" est nulle (inexistante) ou rien si elle est non nulle (présente). **Attention** : ne pas confondre « variable nulle », c'est à dire inexistante, et « variable vide », c'est à dire présente dans l'environnement mais sans valeur affectée.

Ce filtre ne s'applique **pas** aux variables déclarées avec #SET dont la valeur n'est **pas** présente dans l'environnement mais uniquement dans le squelette.

|jour

Septembre 2009

Le filtre `|jour` affiche numériquement le jour de la date sur laquelle il s'applique.
`[(#DATE|jour)]`

L'affichage retourné varie de 1 à 31.

|largeur

Septembre 2009

Le filtre `|largeur` appliqué à un élément image (balise LOGO, image typographique, ...) retourne la largeur calculée de cet élément une fois filtré.
La largeur retournée est exprimée en pixels.

Exemple : après avoir appliqué le filtre `|image_reduire{500}` à un logo, pour connaître la largeur réelle de l'image obtenue, on écrira
`[(#LOGO_DOCUMENT||image_reduire{500}|largeur)]`.

|liens_absolus

Juillet 2009 — maj : Novembre 2009

Le filtre `|liens_absolus` s'applique sur une balise de texte (`#TEXTE`, `#DESCRIPTIF`, etc.) et transforme tous les liens que celui-ci contient (`<a href=...`, `<link href:...`, `<img src=...` et `<script src=...`) en liens absolus (avec l'url complète du site).

exemple :
```
<img src="local/cache-vignettes/L500xH333/belle_image.jpg"..
```
sera traduit en
```
<img src="http://le_site.fr/local/cache-
vignettes/L500xH333/belle_image.jpg"...
```

ou encore
```
<a href="spip.php?page=article&amp;id_article=25"...
```
donnera
```
<a href="http://le_site.fr/spip.php?page=article&amp;id_article=25"...
```

- Ce filtre est particulièrement utile dans des squelettes de fil rss par exemple.

|lien_ou_expose

16 avril

Le filtre `|lien_ou_expose` retourne soit un lien cliquable soit un simple intitulé.

Appliqué à une URL, le filtre `|lien_ou_expose{intitulé, condition}` affiche en fonction du résultat de la condition soit le lien cliquable vers cette url, soit seulement l'intitulé passé en argument.

Exemple :
`[(#URL_RUBRIQUE|lien_ou_expose{#TITRE, #ENV{test}|=={en_cours}})]`
affichera *s'il existe une variable de nom « test » et de valeur « en_cours » dans l'environnement* :

Titre_de_la_rubrique

affichera sinon :

Titre_de_la_rubrique

Le filtre accepte aussi **3 arguments supplémentaires optionnels** « class », « title » et « rel »

```
[(#URL_RUBRIQUE|lien_ou_expose{#TITRE, #ENV{test}|=={en_cours}, spip_lien,
la super rubrique, biblioentry})]
```

produisant alors soit le souce html suivant (en fonction de la condition) :

```
<a href='-Titre_de_la_rubrique-.html' class='spip_lien' title='la super
rubrique' rel='biblioentry'>Titre_de_la_rubrique</a>
```

soit :

```
<strong class='on'>Titre_de_la_rubrique</strong>
```

|liens_ouvrants

Octobre 2009

Le filtre `|liens_ouvrants`, appliqué sur une balise de texte, transforme les liens SPIP qui mènent vers des sites extérieurs pour qu'ils s'ouvrent dans une nouvelle fenêtre ou onglet (« popup ») ; c'est l'équivalent du « *target="_blank"* » du HTML (lire aussi : Des liens qui ouvrent une nouvelle fenêtre).

|lignes_longues

Octobre 2009

Le filtre `|lignes_longues`, introduit des césures dans les mots « trops longs » en y insérant des espaces sécables qui permettent alors le passage à la ligne (utile, par exemple, pour afficher des urls dans une colonne étroite).

Ainsi, si le #TEXTE d'un article contient un lien comme :

```
<a
href="http://www.spip.net/spip.php?page=recherche&recherche=lignes_longues"
>http://www.spip.net/spip.php?page=recherche&recherche=lignes_longues</a>
```

alors

```
<p style="width:30px;">
[(#TEXTE|lignes_longues{20})]
</p>
```

affichera

```
http://www.spip.net
/spip.
php?page=recherche&
recherche=lignes_longues
```

Noter que la césure conserve les tag html ; dans l'exemple ci-dessus, ***tout** le lien* est cliquable comme on peut le voir dans le source html.

Ce filtre coupe par défaut à 70 caractères, mais on peut spécifier une autre longueur en passant un paramètre au filtre, par exemple :
`[(#TEXTE|lignes_longues{40})]`.

|match

Septembre 2009 — maj : 23 octobre

Le filtre `|match` utilise une expression rationnelle (*ne pas hésiter à consulter la page wikipedia dédiée : Expression_rationnelle)* pour afficher un *motif* présent dans la balise, ou rien si absent.

Par exemple :
récupérer le premier mot du titre `[(#TITRE|match{^\w+?})]` ;
afficher "toto" si présent dans le titre : `[(#TITRE|match{toto})]`

Il est possible de passer 3 arguments à ce filtre :
le motif (ou *« pattern »*) : ex. `^a trouver$` ;
les options (ou *modificateurs*) : ex. `Uims` ;
le numéro de parenthèse capturante (par défaut l'intégralité du motif).

Mais il est aussi permis de ne passer que 2 arguments : le motif et le numéro de parenthèse capturante (ex. `#BALISE|match{toto_(\d+)$, 1}` qui ne retournera que le ou les chiffres finaux qui sont directement précédés par le *mot* 'toto_').

Attention :
S'il n'est pas possible de déclarer une *classe de caractères* comme argument du filtre : `|match{[a-zA-Z]+}` produira une erreur de compilation due à laprésence des crochets `[` et `]`, il est néanmoins possible d'écrire des regexp complexes utilisant les classes de caractères en les définissant préalablement :

```
#SET{ma_regexp, ^[[:space:]]*([0-9]+)([.)]|Â?°)[[:space:]]+}
[(#DESCRIPTIF|match{#GET{ma_regexp, Uims, 1}})]
```

|minutes

Septembre 2009

Le filtre `|minutes` affiche les minutes de la date sur laquelle il s'applique.
`[(#DATE|minutes)]`

L'affichage retourné varie de 00 à 59.

Si la date sur laquelle est appliquée le filtre ne contient pas d'information horaire (`[(#VAL{2009-10-24}|minutes)]`), le filtre retournera 0 (zéro).

|modulo

Septembre 2009 — maj : avril 2010

Le filtre `|modulo{xx}` est un filtre d'opérations mathématiques.
Il retourne le *reste* de la division de la valeur de la balise par xx.

Si le retour de la balise n'est pas de type numérique, il est considéré comme 0 (zéro) et le filtre retourne 0 (zéro).

Ce filtre accepte un deuxième argument qui doit être numérique (nombre entier ou nombre à virgule) et qui sera ajouté au résultat :`[(#VAL{20}|modulo{3, 2.4})]` retournera *« 4.4 »*.

Exemple :
pour faire varier un affichage dans une boucle à chaque *tour de boucle* :
`[(#COMPTEUR_BOUCLE|modulo{3})]` affichera « 0 » pour le premier résultat retourné par la boucle, puis « 1 » pour le second résultat, « 2 » pour le troisième, puis reviendra à « 0 » pour le quatrième, « 1 » pour le cinquième...

`|modulo{x}` retourne donc toujours un entier compris entre 0 et son premier argument « x » *moins 1*.

Attention :
si 0 (zéro) est passé comme argument du filtre (`[(#VAL{20}|modulo{0})]` par exemple) un *warning* php —*Warning : Division by zero*— sera émis.

|moins

Septembre 2009

Le filtre `|moins{xx}` est un filtre d'opérations mathématiques.
Il retourne le résultat de la soustraction (la *différence*) de xx à la valeur de la balise.

Si le retour de la balise n'est pas de type numérique, il est considéré comme 0 (zéro) et le filtre retourne l'opposé de la valeur de son argument : `[(#TEXTE|moins{18})]` retournera *« -18 »*.

Il est possible de donner un nombre négatif comme argument du filtre :
`[(#TOTAL_BOUCLE|moins{-5.2})]` mais dans ce cas, sans doute est-il préférable d'utiliser le filtre `|plus`.

Attention, si *xx* est un nombre *à virgule*, sa notation doit utiliser *le point* en lieu et place de la virgule : `[(#TOTAL_BOUCLE|moins{5.2})]`.

|mois

Septembre 2009

Le filtre `|mois` affiche numériquement, sur 2 chiffres, le mois de la date sur laquelle il s'applique.
`[(#DATE|mois)]`

L'affichage retourné varie de 01 à 12.

|mult

Septembre 2009

Le filtre `|mult{xx}` est un filtre d'opérations mathématiques.
Il retourne le résultat de la multiplication (le *produit*) de la valeur de la balise par xx.

Si le retour de la balise n'est pas de type numérique, il est considéré comme 0 (zéro) et le filtre retourne 0 (zéro) : `[(#TEXTE|mult{18})]` retournera *« 0 »*.

Il est possible de donner un nombre négatif comme argument du filtre :
`[(#TOTAL_BOUCLE|mult{-5.2})]`.

Attention, si *xx* est un nombre *à virgule*, sa notation doit utiliser *le point* en lieu et place de la virgule : `[(#TOTAL_BOUCLE|mult{5.2})]`.

|nom_jour

Septembre 2009

Le filtre `|nom_jour`, appliqué à une balise qui retourne une date valide, affiche l'intitulé littéral du jour (lundi, mardi...).

Exemple : pour un article publié le 22-01-2008,
`[(#DATE|nom_jour)]` affichera « *mardi* ».

Ce filtre peut utiliser un argument (soit « *abbr* », soit « *initiale* ») qui modifiera l'affichage résultant en abréviation ou en initiale :
`[(#VAL{2008-01-22}|nom_jour{abbr})]` affichera « *mar.* »
`[(#VAL{2008-01-22}|nom_jour{initiale})]` affichera « *m.* »

|nom_mois

Septembre 2009

Le filtre `|nom_mois`, appliqué à une balise qui retourne une date valide, affiche l'intitulé littéral du mois (janvier, février...).

exemple : pour un article publié le 22-01-2008,
`[(#DATE|nom_mois)]` affichera « *janvier* ».

|non

Mars 2010

Le filtre **|non** (peut aussi être écrit **|not**) retourne soit rien soit un espace.
Équivalent à `|?{'',' '}` ou `|?{''}`, il permet de retourner un contenu vide pour signaler que les parties optionnelles de la balise ne doivent pas 'afficher.

exemples :
```
[(#TITRE|strlen|>{30}|non) Ce titre est court ]
[(#EMAIL|non) #FORMULAIRE_SAISIE_EMAIL ]
```

voir aussi le filtre **|oui**

|ou

17 juin 2010

Le filtre **|ou** (peut aussi être écrit **|or**) permet de vérifier la présence de l'un des 2 éléments.

exemples :
```
[(#CHAPO|ou{#PS}) Il y a soit un chapo, soit un post-scriptum, soit les
deux ]
[(#CHAPO|ou{#PS}|non) Il n'y a ni chapo seul, ni post-scriptum seul, ni
chapo et post-scriptum ]
```

voir aussi les filtres **|et** et **|xou**

|oui

8 mars

Le filtre **|oui** (peut aussi être écrit **|yes**) retourne soit un espace soit rien.
Équivalent à `|?{' ',''}` ou `|?{' '}`, il permet de retourner un contenu non vide (un espace) pour signaler que les parties optionnelles de la balise doivent s'afficher.

exemples :
```
[(#TITRE|strlen|>{30}|oui) Ce titre est long ]
[(#EMAIL|oui) #FORMULAIRE_CONTACT ]
```

voir aussi le filtre **|non**

|parametre_url

26 juillet 2010

Le filtre **|parametre_url{***nom de variable,valeur***}**, appliqué à une balise d'url (par exemple : `#SELF`, `#URL_ARTICLE`, `#URL_PAGE{sommaire}`...), lui ajoute ou lui retire un paramètre.

Par exemple :
`[(#SELF|parametre_url{pays,france})]` ajoutera à l'url de la page en cours soit :

`?pays=france` s'il s'agit du premier paramètre passé à cette url,
soit :
`&pays=france` si l'url possède déjà au moins un paramètre.

Typiquement, on placera la balise dans l'attribut href d'un lien :
```
<a href="[(#SELF|parametre_url{pays,france})]">France</a>
```

Pour récupérer ensuite la valeur du paramètre passé dans l'url, on utilisera la balise `#ENV`.
Ainsi, dans l'exemple ci-dessus `#ENV{pays}` retournera `france`.

Passer plusieurs paramètres à l'URL

On peut enchaîner les filtres pour passer en url une suite de paramètres et leurs valeurs :
```
[(#SELF|parametre_url{pays,france}|parametre_url{monnaie,euro})]
```

Si l'on désire passer *une même valeur à plusieurs paramètres*, on utilisera l'écriture :
```
[(#SELF|parametre_url{pays|lieu|terrain,france})]
```
qui retournera l'url de la page en cours argumentée de :
```
pays=france&lieu=france&terrain=france
```

Supprimer des paramètres existants

Pour faire disparaitre un paramètre d'url existant, il faut le vider de sa valeur en le déclarant à nouveau, mais explicitement vide : `[(#SELF|parametre_url{pays,''})]`

On peut supprimer plusieurs paramètres à la fois avec l'écriture :
```
[(#SELF|parametre_url{pays|lieu|terrain,''})]
```

|plus

Septembre 2009

Le filtre `|plus{xx}` est un filtre d'opérations mathématiques.
Il retourne le résultat de l'addition (la *somme*) de la valeur de la balise et xx.

Si le retour de la balise n'est pas de type numérique, il est considéré comme 0 (zéro) et le filtre retourne son propre argument : `[(#TEXTE|plus{18})]` retournera *« 18 »*.

Il est possible de donner un nombre négatif comme argument du filtre :
`[(#TOTAL_BOUCLE|plus{-5.2})]` mais dans ce cas, sans doute est-il préférable d'utiliser le filtre `|moins`.

Attention, si *xx* est un nombre *à virgule*, sa notation doit utiliser *le point* en lieu et place de la virgule : `[(#TOTAL_BOUCLE|plus{5.2})]`.

|PtoBR

Octobre 2009 — maj : avril 2010

Le filtre `|PtoBR` transforme les sauts de paragraphe en simples retours à la ligne, ce qui permet de « resserrer » une mise en page, par exemple à l'intérieur d'un sommaire.

Ce filtre permet aussi de supprimer *occasionnellement* l'encadrement systématique (voir l'alinéa « *Paragraphes* » de l'article SPIP 2.0) des balises de texte par les tags `<p>` et `</p>`

Par exemple, une balise #DESCRIPTIF qui retourne normalement le source html suivant :

```
<p><strong>William Shakespeare :</strong></p><p style="color:navy;">né à
Stratford-sur-Avon dans une maison sous les tuiles de laquelle était
cachée...</p>
```

retournera, une fois le filtre appliqué :

```
<strong>William Shakespeare :</strong>
<br />né à Stratford-sur-Avon dans une maison sous les tuiles de laquelle
était cachée...
```

|push

Décembre 2009

`|push` permet d'ajouter une valeur à la fin d'un tableau.

Le filtre `|push` s'applique à une balise contenant un *tableau PHP* (voir `#ARRAY`) et y ajoute une nouvelle valeur.

La clé est incrémentée de 1 à chaque nouvelle valeur. Si aucune clé n'a été définie pour la première valeur, le tableau est indexé à partir de 0 (la première clé est 0).

```
#SET{un_tableau,#ARRAY{5,une_valeur}}

#SET{un_tableau,#GET{un_tableau}|push{une_autre_valeur}}

#SET{un_tableau,#GET{un_tableau}|push{une_troisieme_valeur}}
```

`[(#GET{un_tableau}|foreach)]` affiche :

- 5=> une_valeur
- 6=> une_autre_valeur
- 7=> une_troisieme_valeur

La première clé est 5 parce qu'elle a été précisée lors de la création du tableau (premier `#SET`).

Exemple courant d'usage de `|push` :

```
#SET{mes_articles, #ARRAY}
<BOUCLE_mes_articles(ARTICLES){...}>
#SET{mes_articles, #GET{mes_articles}|push{#ID_ARTICLE}}
</BOUCLE_mes_articles>
```

Si les articles sélectionnés par la BOUCLE portent les numéros 2, 5 et 10, `[(#GET{mes_articles}|foreach)]` retourne :

- 0=>2
- 1=>5
- 2=>10

Ici, la première clé est zéro car aucune clé n'a été définie préalablement.

Contrairement à la fonction PHP `array_push`, `|push` ne produit pas de message d'erreur si la balise filtrée n'est pas un tableau.

|replace

Septembre 2009 — maj : Novembre 2009

Le filtre `|replace` utilise une expression rationnelle (*ne pas hésiter à consulter la page wikipedia dédiée : Expression_rationnelle)* pour supprimer ou remplacer toutes les occurences d'un *motif* dans la balise.

Utilisé avec un seul paramètre, une expression régulière, le motif sera supprimé.

Par exemple pour supprimer tous les "notaXX" du texte `[(#TEXTE|replace{nota\d*})]`.

Lorsqu'un deuxième paramètre est fourni au filtre, les occurences du motif seront remplacées par cette valeur.
Par exemple pour remplacer tous les *« 2005 »* **ou** *« 2006 »* du texte par *« 2007 »* :

```
[(#TEXTE|replace{200(5|6), 2007})]
```

.

Les arguments du filtre peuvent n'être que de simples textes :
remplacer tous les *« au temps »* par *« autant »* : `[(#TEXTE|replace{au temps, autant})]`

Un troisième paramètre peut encore être passé au filtre. Il correspondra alors aux *options* (ou *modificateurs*) à appliquer à la regexp :
`[(#TEXTE|replace{^ceci$, cela, UimsS})]`

Attention :
S'il n'est pas possible de déclarer une *classe de caractères* comme argument du filtre : `|replace{[0-9]+}` produira une erreur de compilation due à la présence des crochets `[` et `]`, il est néanmoins possible d'écrire des regexp complexes utilisant les classes de caractères en les définissant préalablement :

```
#SET{ma_regexp, ^[[:space:]]*([0-9]+)([.)]|Â?°)[[:space:]]+}
[(#DESCRIPTIF|replace{#GET{ma_regexp}, __, Uims}})]
```

|safehtml

Octobre 2009

Les contenus HTML provenant de l'extérieur (par retour de formulaire ou syndication) sont par définition considérés comme « non contrôlés », et donc potentiellement problématiques s'ils contiennent des balises mal fermées, des scripts javascript, php, sql...

SPIP leur applique donc systématiquement le filtre `safehtml` avant affichage.

Cela concerne tout particulièrement les flux syndiqués, les forums et les pétitions.

|saison

Septembre 2009

Le filtre `|saison`, appliqué à une balise qui retourne une date valide, affiche l'intitulé littéral de la saison correspondante (été, automne...).

exemple : pour un article publié le 22-01-2008,
`[(#DATE|saison)]` affichera « *hiver* ».

les saisons sont définies comme suit :

- du 21-03 au 20-06 : *printemps*
- du 21-06 au 20-09 : *été*
- du 21-09 au 20-12 : *automne*
- du 21-12 au 20-03 : *hiver*

|secondes

Septembre 2009

Le filtre `|secondes` affiche les secondes de la date sur laquelle il s'applique.
`[(#DATE|secondes)]`

L'affichage produit varie de 00 à 59.

Si la date sur laquelle est appliquée le filtre ne contient pas d'information horaire (`[(#VAL{2009-10-24}|secondes)]`), le filtre retournera 0 (zéro).

|singulier_ou_pluriel{ xxx:chaîne_un, xxx:chaîne_plusieurs}

avril 2010

Le filtre `|singulier_ou_pluriel` permet de faire varier un affichage en fonction de la valeur numérique retournée par la balise à laquelle il est appliqué.

Le filtre `|singulier_ou_pluriel` est la plupart du temps utilisé pour accorder (singulier ou pluriel) les chaînes de langue en fonction du retour d'une balise.

Exemple

Dans notre fichier *local_xx.php*, nous déclarons :

```
'nombre_truc_un'   => 'il y a 1 retour dans cette boucle.',
'nombre_truc_plus' => 'il y a @nb@ retours dans cette boucle.'
```

Puis nous écrivons notre boucle :

```
<BOUCLE_a(ARTICLES) {critère...}>
 ...
</BOUCLE_a>
[(#TOTAL_BOUCLE|singulier_ou_pluriel{local:nombre_truc_un,
local:nombre_truc_plus, nb})]
</B_a>
```

- Si la boucle ne retourne qu'un enregistrement, elle affichera :

```
il y a 1 retour dans cette boucle.
```

- Si la boucle retourne 36 enregistrements (par exemple), elle affichera :

```
il y a 36 retours dans cette boucle.
```

Noter que l'on peut passer un 3e argument à ce filtre : `nb` qui permet d'intégrer dans la chaîne de langue la valeur numérique retournée par la balise.

Attention :
Ce filtre attend explicitement des *item* de langue comme arguments. Il ne faudra donc pas lui passer des *balises* de langue :
`|singulier_ou_pluriel{module:item_un, module:item_plus}`
est la bonne écriture ;
`|singulier_ou_pluriel{<:module:item_un:>, <:module:item_plus:>}`
est une **mauvaise** écriture.

| ?{sioui, sinon}

Août 2009

Le filtre `|?{sioui,sinon}` est une version évoluée de `|sinon`. Il prend un ou deux paramètres :
- **sioui** est la valeur à afficher *à la place de* l'élément filtré si celui-ci est non vide.
- **sinon** est optionnel. C'est la valeur à afficher si l'élément filtré est vide.

`[(#TEXTE|?{#TEXTE, "pas de texte"})]` affiche le contenu de `#TEXTE` s'il a été renseigné ; « pas de texte » si celui-ci est vide.

|sinon

Septembre 2009

Le filtre `|sinon` affiche l'argument qui lui est donné si le retour produit par la balise à laquelle il s'applique est vide :

```
[(#TEXTE|sinon{"pas de texte"})]
```

affichera le contenu de #TEXTE s'il existe, et *« pas de texte »* si #TEXTE ne retourne rien.

L'argument du filtre peut être plus complexe qu'un simple texte :

```
[(#TEXTE|sinon{#INCLURE{fond=ma_page, env, id_article}})]
```

|supprimer_numero

Octobre 2009

Le filtre `|supprimer_numero` s'utilise pour ne pas afficher le préfixe numéroté [1] d'un titre.

Ainsi, pour le titre *« `0118. Titre de mon article` »*, `[(#TITRE|supprimer_numero)]` affichera *« `Titre de mon article` »*.

Notes

[1] Le format des préfixes numérotés est :
un nombre de n chiffres, suivi d'un point ou d'une parenthèse fermante, suivie d'un espace.
exemples :

`015. Le titre`
`51) Le titre`

|supprimer_tags

Octobre 2009

Le filtre `|supprimer_tags` supprime du retour de la balise à laquelle il s'applique tous les tags html (`<...>`) tout en conservant le *contenu* de ces même tags.

Par exemple, pour une balise #DESCRIPTIF qui contiendrait : `<p><strong>William Shakespeare</strong> naquit à Stratford-sur-Avon, <span class='spip_document_164 spip_documents spip_documents_center' ><img src='IMG/png/maison-2.png' width="68" height="43" alt="" /></span> dans une maison sous les tuiles de laquelle était cachée...</p>` retournera, une fois le filtre appliqué :

```
William Shakespeare naquit à Stratford-sur-Avon,  dans une maison sous les tuiles de laquelle était cachée...
```

Noter que le filtre supprime aussi tous les **`<`** rencontrés ; ainsi `18 + x < 21 ; x = ?` deviendra `18 + x  21 ; x = ?`

|table_valeur

Décembre 2009

`|table_valeur` retourne la valeur associée à une clé dans un tableau.

`|table_valeur{clé,valeur par défaut}`

Le filtre `|table_valeur` (depuis SPIP 1.9) s'applique à une balise contenant un tableau, prend une clé du tableau comme paramètre et retourne la valeur associée à cette clé. Voir la balise #ARRAY pour une explication plus générale sur les tableaux.

Exemple :

```
#SET{un_tableau,#ARRAY{0,rouge,1,bleu,2,vert}}
[(#GET{un_tableau}|table_valeur{2})]
```

retourne *vert*.

Le paramètre optionnel *valeur par défaut* permet de préciser une valeur à retourner si la valeur n'est pas trouvée (la clé n'existe pas ou la balise filtrée n'est pas un tableau).

```
[(#GET{un_tableau}|table_valeur{3,noir})]
```

retourne *noir* parce qu'il n'y a pas de clé *3*.

`|table_valeur` fonctionne aussi avec des tableaux "linéarisés", par exemple, les tableaux retournés par les balises `#ENV` et `#CONFIG` (mais ces balises disposent de leur propre syntaxe pour accéder aux valeurs du tableau).

Le tableau ci-dessus linéarisé (Voir les fonctions serialize et unserialize en PHP) donne ceci :

```
a:3:{i:0;s:5:"rouge";i:1;s:4:"bleu";i:2;s:4:"vert";}
```

```
[(#GET{un_tableau}|table_valeur{2})]
```

retourne toujours *vert*.

|taille_en_octets

Septembre 2009

Le filtre `|taille_en_octets` traduit un nombre d'octets (25678906) en une chaîne de caractères plus explicite (*« 24.4 Mo »*).

exemple, dans une boucle DOCUMENTS :

```
#TITRE - #TYPE_DOCUMENT[  - (#TAILLE|taille_en_octets)])
```

affichera *« Mon document - JPEG - 82.6 Ko »*

|texte_backend

Octobre 2009

Le filtre `|texte_backend` appliqué à une balise de texte traduira le rendu html de celle-ci dans un format compatible avec les flux XML.
Il est utilisé, par exemple, dans le squelette *backend.html* qui génére un fil RSS.

Ce filtre transforme les liens en liens absolus, importe les entitées html et échappe les tags html.

Par exemple, une balise #DESCRIPTIF qui retourne normalement le source html suivant :

```
<p><strong>William Shakespeare</strong> naquit à Stratford-sur-Avon, <span
class='spip_document_164 spip_documents spip_documents_center' ><img
src='IMG/png/maison-2.png' width="68" height="43" alt="" /></span> dans une
maison sous les tuiles de laquelle était cachée...</p>
```

retournera, une fois le filtre appliqué :

```
&lt;p&gt;&lt;strong&gt;William Shakespeare&lt;/strong&gt; naquit &#224;
Stratford-sur-Avon, &lt;span class='spip_document_164 spip_documents
spip_documents_center' &gt;
&lt;img src='http://www.domaine.tld/IMG/png/maison-2.png'
width=&quot;68&quot; height=&quot;43&quot; alt=&quot;&quot;
/&gt;&lt;/span&gt; dans une maison sous les tuiles de laquelle &#233;tait
cach&#233;e...&lt;/p&gt;
```

|textebrut

Octobre 2009

Le filtre `|textebrut` transforme le retour de la balise sur laquelle il est appliqué en remplaçant les tags `<p>` et `<br />` par de simple retour à la ligne ; les doubles retours à la ligne par un retour simple et en transformant les espaces insécables et les doubles espaces par des espaces simples.

Il s'utilise, par exemple, pour renseigner un tag *meta* (html) : `[<meta name="description" content="(#DESCRIPTIF|textebrut)">]`.

|texte_script

Octobre 2009 — maj : Décembre 2009

le filtre `|texte_script` transforme tout retour de balise en une chaîne utilisable en toute sécurité dans un script PHP ou Javascript.

Par exemple le script php suivant :

```
<?php $x = '[(#TEXTE|texte_script)]'; ?>
```

Attention : utilisez bien le caractère ' (*simple quote*) et non " (*double quote*) pour délimiter votre variable php. En effet, dans le second cas, si votre texte contient le signe $, le résultat peut être catastrophique (affichage partiel, affichage d'autre chose, plantage php, etc.).

Appliqué sur une balise dans un squelette (html) hors script js ou php, ce filtre se contente d'échapper l'apostrophe : `c\'était l\'été`.

|traduire_nom_langue

Septembre 2009

`|traduire_nom_langue` s'applique à la balise `#LANG` et retourne un traduction du code de langue qu'elle retourne (fr, en, it, etc.) dans cette langue.

Remarque : Les traductions des codes sont faites dans la langue que représente ce code et suivent les conventions d'écriture de cette langue.

Ainsi « fr » sera traduit « français » en minuscule, alors que « es » sera traduit « Español » avec une majuscule.

|unique

Octobre 2009

Le filtre `|unique` retourne la valeur de la balise à laquelle il s'applique seulement si c'est la premiere fois qu'elle est rencontrée.
Il s'applique donc sur une balise placée à l'intérieur d'une boucle.

Il est possible, pour différencier plusieurs utilisations indépendantes au sein de la boucle, de passer un argument de nommage à ce filtre.
Par exemple : `[(#ID_SECTEUR|unique{en_tete})]` n'aura pas d'incidence sur (*n'empêchera pas l'affichage de*) : `[(#ID_SECTEUR|unique{corps})]`.

Le filtre accepte aussi un deuxième argument : *« 1 »* pour afficher le nombre de fois où la balise a été filtrée.
Par exemple : `[(#ID_SECTEUR|unique{corps, 1})]` affichera le total correspondant au nombre de fois où #ID_SECTEUR aura été filtré par `|unique{corps}`.
On préfèrera toutefois à cette notation l'utilisation (plus économique) de la balise spécifique `#TOTAL_UNIQUE{corps}` (Voir la page consacrée à la balise #TOTAL_UNIQUE).

- Le filtre `|unique` peut être intéressant pour, par exemple, afficher une liste d'articles par date :

```
<BOUCLE_blog(ARTICLES) {par date} {inverse} {"<br>"}>
 [<hr /><h1>(#DATE|affdate_mois_annee|unique)</h1>]
 #TITRE ...
</BOUCLE_blog>
```

La date ne sera affichée qu'à chaque changement de mois.

- Autre exemple :

```
<BOUCLE_blog2(ARTICLES) {par date} {inverse}>
 [<hr /><h1>(#DATE|annee|unique)</h1>]
 [<h2>(#DATE|affdate{'Y-m'}|unique|nom_mois)</h2>]
 <a href="#URL_ARTICLE">#TITRE</a><br />
</BOUCLE_blog2>
```

affichera une liste ressemblant à :

```
2005
 mars
   article de mars
   autre article de mars
 février
```

```
   article de février
2004
 décembre
   un article
```

Dans ce dernier exemple, on utilise la notation `affdate{'Y-m'}` pour afficher le nom du mois à chaque changement d'année. En effet :

- si l'on ne faisait que `[(#DATE|nom_mois|unique)]`, les noms de mois ne seraient affichés que la première année ;
- si le filtrage était `[(#DATE|unique|nom_mois)]`, on afficherait toutes les dates. En effet, `#DATE` retourne une date complète qui contient aussi les heures, minutes et secondes. Il y a donc une grande probabilité que les dates complètes de deux articles publiés le même jours soient différentes.
  C'est pourquoi on ne garde que le mois et l'année de la date avant de la passer au filtre *unique*.

|url_absolue

Juillet 2009 — maj : octobre 2010

- **`|url_absolue`** fonctionne de la même façon que le filtre **`|liens_absolus`**, mais s'applique à une balise qui retourne une url (par exemple `#URL_ARTICLE` ou `#URL_RUBRIQUE`...).

exemples :
`<a href="-Publications-"...`
sera traduit en `<a href="http://le_site.fr/-Publications-"...`

ou encore
`<a href="spip.php?rubrique67"...`
donnera
`<a href="http://le_site.fr/spip.php?rubrique67"...`

de même `|url_absolue` peut être appliqué à la balise `#CHEMIN` :
`[(#CHEMIN{polices/dustismo_bold.ttf}|url_absolue)]` retournera
`http://mon_domaine.tld/squelettes-dist/polices/dustismo_bold.ttf .`

Attention, cette balise conserve le protocole (http ou https) et le nom d'hôte de la page appelante. Pour forcer une certaine URL de base, il est possible de la passer en argument. Ceci peut s'avérer utile si votre site se décline en plusieurs sous-domaines mais que vous souhaitez forcer un certain préfixe. Par exemple, si votre site se décline en **a.le-site.fr**, **b.le-site.fr**, etc. :
`#CHEMIN{truc.css}|url_absolue{#URL_SITE_SPIP}`
donnera :
`http://le-site.fr/squelettes/truc.css`
et ce, même si la balise est appelée depuis **a.le-site.fr**.

|url_absolue_css

Octobre 2009

Le filtre `|url_absolue_css` appliqué à un fichier de styles, transforme toutes les url relatives présentes en url absolues.

Noter que le filtre `|compacte` utilise ce filtre dans son traitement.

|vider_attribut

Septembre 2009

Le filtre `vider_attribut{attribut}` est une variante de `inserer_attribut`. Il permet de supprimer les attributs html.
Par exemple, on peut vouloir réinitialiser un attribut avant de lui affecter une valeur spécifique :

```
[(#LOGO||vider_attribut{style}|inserer_attribut{style,'width:100px;'})]
```

Ce code commence par enlever complètement l'attribut 'style' sur l'image d'un logo avant de le remettre mais doté cette fois seulement de la valeur désirée.

|xou

Mars 2010

Le filtre **`|xou`** (peut aussi être écrit `|xor`) permet de vérifier la présence d'un seul élément parmi 2.

exemples :
```
[(#CHAPO|xou{#PS}) Il y a soit un chapo, soit un post-scriptum, mais pas
les 2 ]
[(#CHAPO|xou{#PS}|non) Il peut y avoir ni chapo ni post-scriptum ou bien un
chapo et un post-scriptum]
```

Voir aussi les filtres **`|et`** et **`|ou`**

Pour tous les filtres ci-après :

[1] si un nombre décimal est passé comme valeur, il faut utiliser le point et non la virgule

[2] la vérification tient compte de la casse (majuscules-minuscules) utilisée

|>{a}

Octobre 2009

Le filtre `|>{valeur}` retourne « vrai » si le résultat de la balise à laquelle il s'applique est strictement supérieur à « valeur ».

`{valeur}` peut être numérique [1] (5, 18.2, 84,...), ou alphanumérique [2] (abc, un mot,...)

Par exemple :
```
[Il y a (#TOTAL_BOUCLE|>{1543}|?{' vraiment beaucoup d'articles', ' trop d'articles'}) dans cette rubrique.].
```

|>={a}

Septembre 2009

Le filtre `|>={valeur}` retourne « vrai » si le résultat de la balise à laquelle il s'applique est supérieur ou égal à « valeur ».

`{valeur}` peut être numérique [1] (5, 18.2, 84,...), ou alphanumérique [2] (abc, un mot,...)

Par exemple :
`[(#TOTAL_BOUCLE) [(#TOTAL_BOUCLE|>={2}|?{'articles', 'article'})] dans cette rubrique.]` pour afficher correctement l'accord.

|<{a}

Octobre 2009

Le filtre `|<{valeur}` retourne « vrai » si le résultat de la balise à laquelle il s'applique est strictement inférieur à « valeur ».

`{valeur}` peut être numérique [1] (5, 18.2, 84,...), ou alphanumérique [2] (abc, un mot,...)

Par exemple :
`[(#TOTAL_BOUCLE) [(#TOTAL_BOUCLE|<{2}|?{'article', 'articles'})] dans cette rubrique.]` pour afficher correctement l'accord.

|<={a}

Septembre 2009

Le filtre `|<={valeur}` retourne « vrai » si le résultat de la balise à laquelle il s'applique est inférieur ou égal à « valeur ».

`{valeur}` peut être numérique [1] (5, 18.2, 84,...), ou alphanumérique [2] (abc, un mot,...)

Par exemple :
```
[(#TOTAL_BOUCLE|<={4})]
```

| !={a}

Septembre 2009

Le filtre `|!={valeur}` permet de vérifier la stricte inégalité entre l'élément filtré et *valeur*.

La vérification tient compte de la casse des caractères (« A » différent de « a »).

Par exemple : `<li [(#TITRE|!={édito}|?{'','id="edito"'})]>#TITRE</li>`

|=={a}

Septembre 2009

Le filtre `|=={valeur}` permet de vérifier la stricte égalité entre l'élément filtré et *valeur*.

La vérification tient compte de la casse des caractères (« A » différent de « a »).

Par exemple : `<li [(#TITRE|=={édito}|?{'id="edito"',''})]>#TITRE</li>`.

Variables et Constantes de personnalisation

Certains comportements des pages de votre site peuvent être modifiés au moyen de variables ou de constantes PHP. Ces variables ou constantes sont normalement définies par SPIP, mais, pour obtenir une personnalisation plus fine du site, le webmestre peut les modifier.
La plupart de ces personnalisations sont à définir dans un fichier `mes_options.php` que l'on prendra soin de placer dans le répertoire `config/` du site.
Ce fichier devra être un fichier php débutant donc par **`<?php`** et se terminant par **`?>`** (*attention de ne pas laisser d'espace ou de ligne vierge avant `<?php` ou après `?>`*).

_CNIL_PERIODE

Septembre 2010

À partir de la version 2.1.2 de SPIP, les adresses IP stockées dans la table « spip_forum » sont encryptées de manière non réversible (algorithme MD5) au bout de 4 mois.

Il est possible de modifier le délai avant cryptage des IP (4 mois par défaut) en définissant cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré).

Par exemple :

```
// crypter les IP au bout de 2 mois
define('_CNIL_PERIODE', 3600*24*31*2);
```

Pour que SPIP ne crypte *jamais* les IP vous pouvez définir cette constante à zéro :

```
// ne jamais crypter les IP
define('_CNIL_PERIODE', 0);
```
P.-S.

Ce cryptage s'appuie sur l'article 226-20 du code pénal français et les bonnes pratiques éditées par la CNIL.

_DIR_PLUGINS_AUTO

Avril 2010

`_DIR_PLUGINS_AUTO` définit l'emplacement du dossier qui contiendra les plugins installables par le mécanisme de téléchargement de plugins intégré à SPIP.

`_DIR_PLUGINS_AUTO` est une chaîne de caractères correspondant à un chemin relatif à partir de la racine de SPIP.

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré).

Définir cette constante à la valeur `''` (chaîne vide) permet de désactiver complètement la fonctionnalité d'installation automatique.

Exemple :

```
// Désactiver l'installation par téléchargement depuis l'administration de
SPIP
define('_DIR_PLUGINS_AUTO', '');
```

Par défaut, cette constante vaut :

```
define('_DIR_PLUGINS_AUTO', _DIR_PLUGINS.'auto/');
```

C'est à dire : le sous-dossier *« auto/ »* dans le dossier *« plugins/ »*

_DOC_MAX_SIZE

Mars 2010

Cette constante définit le poids maximum en kilo-octets en-dessous duquel SPIP acceptera d'enregistrer un document [1].

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré) [2]

- Valeur par défaut : `0`

exemple :

```
// les documents de plus de 250 Ko ne seront pas enregistrés
define('_DOC_MAX_SIZE', 250);
```

Lorsque cette constante est définie à `0` (zéro), SPIP n'effectue pas de vérification lors de l'upload d'un document.

Voir aussi : _IMG_MAX_SIZE.

Notes

[1] de par le fonctionnement des formulaires d'upload, le fichier document sera quand même envoyé sur le serveur. Ce n'est qu'une fois « *uploadé* » que SPIP vérifiera la conformité de ce fichier avec la constante.

[2] Attention, cela génèrera un message PHP d'avertissement de type NOTICE : vérifiez que le serveur n'affiche pas ce type d'erreur.

_ID_WEBMESTRES

Avril 2010

`_ID_WEBMESTRES` définit le ou les administrateurs ayant le rôle de webmestre (ce qui donne des droits supplémentaires dans l'interface privée).

`_ID_WEBMESTRES` est une chaîne de caractères composée d'une suite d'id_auteur, séparés par le caractère `:` (deux points).

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré)

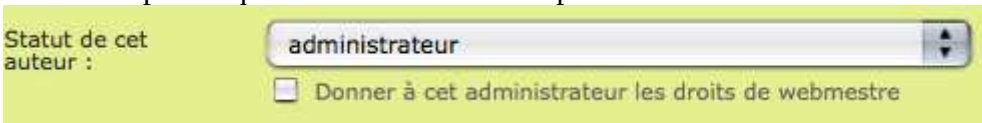
exemple :

```
// donner les droits de webmestre aux auteurs 1, 5 et 18
define('_ID_WEBMESTRES', '1:5:18');
```

Attention : les droits de webmestre ne seront donnés aux auteurs d'id 1, 5 et 18 que si et seulement si ces auteurs sont administrateurs *non restreints*.

Par défaut, celui qui installe SPIP est webmestre.

À noter que depuis la version 2.1 de SPIP, cette constante est dépréciée : on utilisera plutôt l'interface privée qui offre désormais cette possibilité.



_IMG_MAX_HEIGHT

18 mars 2010

Cette constante définit la hauteur maximale en pixels en-dessous de laquelle SPIP acceptera d'enregistrer une image [1].

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré) [2]

- Valeur par défaut : `0`

exemple :

```
// les images de plus de 500 pixels de haut ne seront pas enregistrées
define('_IMG_MAX_HEIGHT', 500);
```

Lorsque cette constante est définie à `0` (zéro), SPIP n'effectue pas de vérification lors de l'upload d'une image.

Voir aussi : _IMG_MAX_SIZE et _IMG_MAX_WIDTH.

Notes

[1] de par le fonctionnement des formulaires d'upload, le fichier image sera quand même envoyé sur le serveur. Ce n'est qu'une fois « *uploadé* » que SPIP vérifiera la conformité de ce fichier avec la constante.

[2] Attention, cela génèrera un message PHP d'avertissement de type NOTICE : vérifiez que le serveur n'affiche pas ce type d'erreur.

_IMG_MAX_SIZE

Mars 2010

Cette constante définit le poids maximum en kilo-octets en-dessous duquel SPIP acceptera d'enregistrer une image [1].

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré) [2]

- Valeur par défaut : `0`

exemple :

```
// les images de plus de 350 Ko ne seront pas enregistrées
define('_IMG_MAX_SIZE', 350);
```

Lorsque cette constante est définie à `0` (zéro), SPIP n'effectue pas de vérification lors de l'upload d'un document.

À ne pas confondre avec _IMG_MAX_WIDTH et _IMG_MAX_HEIGHT.

Notes

[1] de par le fonctionnement des formulaires d'upload, le fichier image sera quand même envoyé sur le serveur. Ce n'est qu'une fois « *uploadé* » que SPIP vérifiera la conformité de ce fichier avec la constante.

[2] Attention, cela génèrera un message PHP d'avertissement de type NOTICE : vérifiez que le serveur n'affiche pas ce type d'erreur.

_IMG_MAX_WIDTH

18 mars 2010

Cette constante définit la largeur maximale en pixels en-dessous de laquelle SPIP acceptera d'enregistrer une image [1].

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré) [2]

- Valeur par défaut : `0`

exemple :

```
// les images de plus de 500 pixels de large ne seront pas enregistrées
define('_IMG_MAX_WIDTH', 500);
```

Lorsque cette constante est définie à `0` (zéro), SPIP n'effectue pas de vérification lors de l'upload d'une image.

Voir aussi : _IMG_MAX_SIZE et _IMG_MAX_HEIGHT.

Notes

[1] de par le fonctionnement des formulaires d'upload, le fichier image sera quand même envoyé sur le serveur. Ce n'est qu'une fois « *uploadé* » que SPIP vérifiera la conformité de ce fichier avec la constante.

[2] Attention, cela génèrera un message PHP d'avertissement de type NOTICE : vérifiez que le serveur n'affiche pas ce type d'erreur.

_MAX_ART_AFFICHES

Avril 2010
`_MAX_ART_AFFICHES` permet de définir le nombre d'articles listés dans la boite « *Dans la même rubrique* » de la page `/?exec=articles&id_article=xxx` dans l'espace privé.

Il est possible de définir cette constante (par défaut SPIP affiche 10 articles) dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré)

Exemple :

```
// afficher une liste de 25 articles dans la boite 'Dans la même rubrique'
define('_MAX_ART_AFFICHES', '25');
```

_NOM_PERMANENTS_ACCESSIBLES

juin 2010

Le nom du répertoire des fichiers permanents accessibles par HTTP. Ce répertoire contient les images et documents qui sont téléchargés (ou plutôt téléversés) via l'interface privée de SPIP. Il contient également les logos des différents éléments de SPIP.

```
define('_NOM_PERMANENTS_ACCESSIBLES', "IMG/");
```

Cette constante est modifiable uniquement dans ecrire/inc_version.php

_NOM_PERMANENTS_INACCESSIBLES

juin 2010

Le nom du répertoire des fichiers permanents inaccessibles par HTTP. Il contient le fichier de configuration généré par SPIP lors de l'installation, mais également la base sqlite et le fichier chmod.php (qui permet de mieux gérer le mode d'accès en écriture déterminé par le serveur)

```
define('_NOM_PERMANENTS_INACCESSIBLES', "config/");
```

Cette constante est modifiable uniquement dans ecrire/inc_version.php

_NOM_TEMPORAIRES_ACCESSIBLES

juin 2010

Le nom du répertoire des fichiers temporaires accessibles par HTTP. Ce répertoire contient principalement le cache des images, feuilles de style et fichiers Javascript qui sont manipulés par SPIP.

```
define('_NOM_TEMPORAIRES_ACCESSIBLES', "local/");
```

Cette constante ne peut être modifiée que dans ecrire/inc_version.php

_NOM_TEMPORAIRES_INACCESSIBLES

juin 2010

le nom du répertoire des fichiers temporaires inaccessibles par HTTP. Il contient principalement les fichiers de log, les exports de la base, et la plupart des caches.

```
define('_NOM_TEMPORAIRES_INACCESSIBLES', "tmp/");
```

Application : si l'on souhaite placer les fichiers concernés ailleurs que dans tmp/, il suffit de changer la valeur de cette constante pour le nom du nouveau répertoire qui aura été choisi (ex. "tmp123456/")

Cette constante doit être modifiée directement dans ecrire/inc_version.php

_TRI_ARTICLES_RUBRIQUE

Avril 2010

`_TRI_ARTICLES_RUBRIQUE` permet de définir l'ordre de tri pour l'affichage des articles listés dans la boite « *Dans la même rubrique* » de la page `/?exec=articles&id_article=xxx` ainsi que l'ordre de tri pour l'affichage des articles listés dans la boite « *Tous les articles publiés dans cette rubrique* » de la page `/?exec=naviguer&id_rubrique=xx` dans l'espace privé.

Par défaut, SPIP ordonne les articles par date de rédaction (champ sql `date`) de la plus récente à la plus ancienne.

Il est possible de définir cette constante dans votre fichier `config/mes_options.php` (voir l'article qui lui est consacré)

Exemple :

```
// ordonner les listes d'articles dans l'espace privé de celui
// de date de modification la plus récente  à celui de date de
// modification la plus ancienne
define('_TRI_ARTICLES_RUBRIQUE', 'date_modif DESC');
```

$controler_dates_rss

Mai 2010

`$controler_dates_rss` définit si SPIP doit vérifier les dates des élèments des flux rss des sites syndiqués.

Lorsque SPIP intégre les articles issus d'un flux RSS dans sa base de données, il vérifie la date des articles :

- si l’article a plus d’un an
- ou s’il est daté de plus de deux jours dans le futur

Si l’une de ces conditions est remplie, SPIP considère alors qu’il y a une erreur dans la date de l’article, et remplace celle-ci par la date courante.

Il est possible de désactiver ce comportement en définissant la variable `$controler_dates_rss` comme `false` dans le fichier `config/mes_options.php` (voir l’article qui lui est consacré)

Exemple :

```
// ne pas modifier la date des articles syndiqués
$controler_dates_rss = false;
```

$filtrer_javascript

Mars 2010

la variable `$filtrer_javascript` permet de choisir le mode de sécurisation que SPIP appliquera sur les scripts javascript embarqués dans les rédactionnels.

Pour les scripts javascript contenus dans les articles, rubriques, champ bio des auteurs,... trois modes de traitement sont possibles :

- *parano* : -1
- *par défaut* : 0
- *ok* : 1

Le *mode parano* n’exécute le code ni dans l’espace privé ni lors de l’affichage public (le code est simplement affiché de manière inoffensive).
Le *mode par défaut* affiche le code en rouge (sans l’exécuter) dans l’espace privé mais l’exécute (sans l’afficher) lors de l’affichage public.
Le *mode ok* exécute le code partout (affichage public comme affichage dans l’espace privé).

Il est possible de changer la valeur de cette variable dans le fichier *mes_options.php* (à placer dans le répertoire config/) :

```
// sécuriser les scripts javascript en mode parano
$filtrer_javascript = -1;
```

$nombre_de_logs

25 juin 2010

Cette variable PHP définit le nombre maximal d'archives de chaque fichier de logs de SPIP (mysql.log, spip.log, prive_spip.log, etc..) qui sont placés dans _NOM_TEMPORAIRES_INACCESSIBLES.

En effet, une fois qu'un fichier de log a atteint sa taille maximale, il est sauvegardé sous un autre nom puis un fichier vide est créé.

$nombre_de_logs vaut 4 par défaut (ce qui correspond en fait à un fichier en écriture et 3 fichiers archivés).

Application : si l'on souhaite ne plus avoir de fichier de log, il suffit de rajouter dans mes_options.php la ligne suivante :

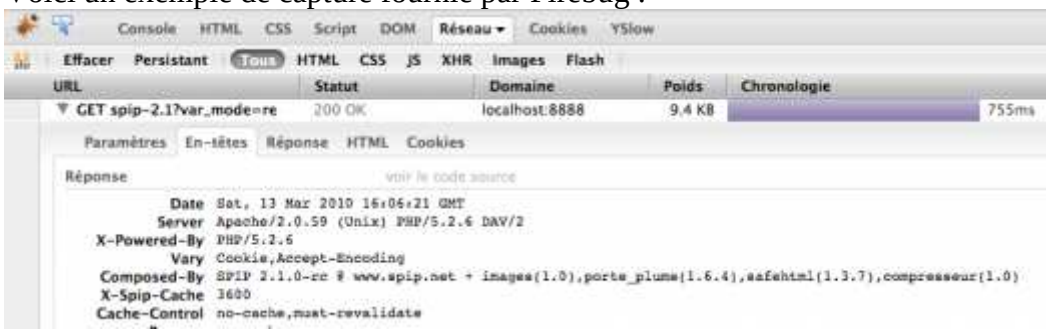
```
$nombre_de_logs = 0;
```

$spip_header_silencieux

25 juin 2010

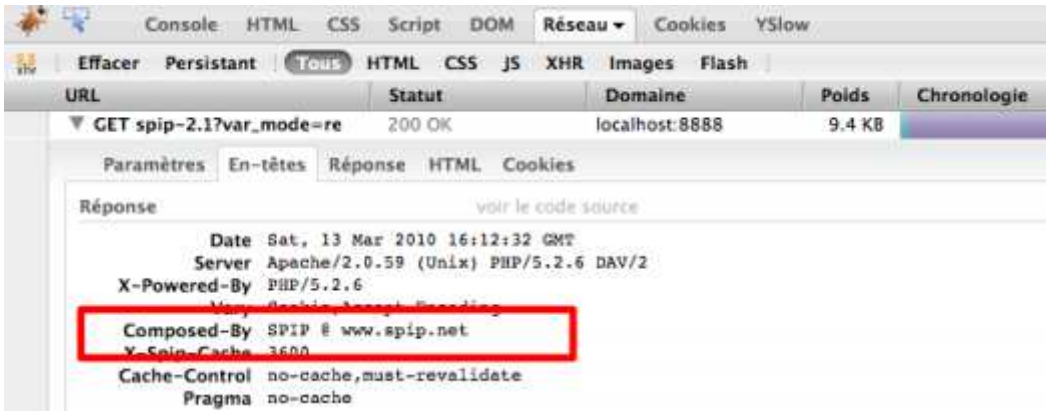
Cette variable permet de préciser si l'on souhaite que l'entête HTTP [1] contienne les informations relatives à la version de SPIP et aux plugins utilisés, ce que fait SPIP par défaut.

Voici un exemple de capture fournie par Firebug :



Pour limiter au maximum les informations contenues dans l'entête HTTP, il suffit de rajouter dans `config/mes_options.php` la ligne :

```
$spip_header_silencieux = 1;
```



Notes

[1] L'entête HTTP permet d'analyser ce que renvoie le site web (plus exactement le serveur web) lorsque vous demandez une URL depuis votre navigateur. Vous pouvez tester ce que retourne le serveur sur ce site : http://www.outils-referencement.com

$taille_des_logs

25 juin 2010

Cette variable PHP définit la taille maximale (en Ko) des fichiers de logs de SPIP (mysql.log, spip.log, prive_spip.log, etc..) qui sont placés dans _NOM_TEMPORAIRES_INACCESSIBLES.

Elle est personnalisable dans mes_options.php, et vaut 100 par défaut.

Application : si l'on souhaite ne plus avoir de fichier de log, il suffit de rajouter dans mes_options.php la ligne suivante :

```
$taille_des_logs = 0;
```