SPIP 3.0

Mai 2012 — mis à jour le : Novembre 2012
Toutes les versions de cet article : [ar] [en]

SPIP 3 : une nouvelle version marquée par la réécriture complète de l'espace privé en squelettes, une forte modularisation et une boucle DATA révolutionnaire.

SPIP 2.0 avait apporté les fonctionnalités permettant de l'utiliser comme un « framework » pour le développement d'applications Web, au-delà de sa vocation initiale de simple outil de publication.

SPIP 3 pousse la logique du « framework » jusqu'à se l'appliquer à lui-même : l'ensemble de l'espace privé de SPIP a été recodé en squelettes, sur la base des outils et fonctions proposés par le langage de squelettes de SPIP.

Cette remise à plat de l'espace privé a été l'occasion de repenser le fonctionnement des objets éditoriaux et de normaliser leur usage pour le rendre le plus générique possible : la plupart des particularités historiques de chaque objet (et les exceptions de traitement associées dans le code de SPIP) ont été gommées pour être ramenées à une simple déclaration.

La création de nouveaux objets éditoriaux et la personnalisation des objets existants deviennent ainsi beaucoup plus faciles et plus rapides.

SPIP 3 achève également la découpe du logiciel en plugins comme SPIP 2 l'avait amorcé : l'ensemble des fonctionnalités proposées par SPIP 2 repose dorénavant sur un noyau **SPIP 3** accompagné de 23 plugins.

La découpe complète du noyau a permis de compléter ses API et points d'entrée pour les développeurs de plugins.

SPIP 3 se nourrit fortement des développements de la communauté SPIP-Zone, et marque une forte inversion : ce n'est plus le noyau qui tire les développements des plugins, mais les développements de plugins qui nourrissent l'avancée de SPIP par leurs expérimentations [1].

Parmi de nombreuses autres nouveautés fonctionnelles, **SPIP 3** introduit une nouvelle boucle DATA qui permet enfin de boucler sur tout type de données et plus seulement sur les tables SQL. Il devient ainsi possible de parcourir soit une énumération, soit un fichier CSV, XML, YAML... Plus fort encore, la boucle DATA permet directement de boucler sur une URL : il devient possible de parcourir directement une feuille de calcul Google Spreadsheet, un calendrier en ligne , une liste de vidéos sur Youtube , des photos sur Flickr … le Web devient votre base de données !

Prérequis : SPIP 3.0 requiert désormais au minimum une version 5.1.0 de PHP pour pouvoir fonctionner correctement.

Un nouvel espace privé [2]

Visuellement, l'espace privé demeure dans la continuité des versions précédentes, complété de nouvelles icônes, avec :

- pour les icônes principales de navigation, un jeu spécifique réalisé pour SPIP par Sébastien Desbenoît ,
- pour le reste de la page, l'adoption d'un jeu dérivé de FatCow.

La refonte de l'interface est essentiellement technique, même si elle a également permis de résoudre un certain nombre de défauts ergonomiques historiques.

La navigation principale est repensée et réorganisée. Le menu déroulant, désormais entièrement navigable au clavier, intègre aussi des temporisations pour faciliter son utilisation à la souris et être plus robuste aux mouvements mal contrôlés.

L'ensemble de l'espace privé est donc réécrit sous forme de squelettes rangés dans le répertoire `prive/squelettes` organisés par sous-dossiers correspondants à un découpage de la page en blocs. Ce découpage est utilisé notamment pour le rafraîchissement des morceaux de page en Ajax, évitant le rechargement complet des pages et fluidifiant les interactions [3].

De nombreux points d'entrée (utilisés en premier lieu par les plugins du noyau) sont disponibles sous forme de pipeline pour permettre aux plugins d'étendre les fonctionnalités de l'espace privé.

Les pages de l'espace privé sont réorganisées et renommées. En particulier toutes les pages d'objets éditoriaux utilisent un nommage cohérent fondé sur le nom de l'objet.

La réécriture a également permis de remettre à niveau la structure complète du HTML et son accessibilité. Un référentiel a été mis en place pour que les développeurs de plugins puissent réutiliser les mêmes structures, et que les ajouts d'interface restent homogènes avec le noyau.

Toutes les listes d'objet sont des squelettes qu'il devient facile de personnaliser. Elles bénéficient toutes du tri par colonne et de pagination ajax.

Un système d'échafaudage des pages d'objet permet de construire automatiquement une interface minimale pour les nouveaux objets déclarés par les plugins. Chaque élément de cette interface par défaut peut ensuite être personnalisé en fonction des particularités de chaque objet : [4].

Tous les formulaires interactifs sont écrits sous forme de formulaires CVT, ce qui permet d'assurer des interactions de qualité avec gestion systématique des messages d'erreur, vérification de saisie, ajax robuste à volonté, saisie multipage si nécessaire… Ils sont ainsi tous facilement extensibles par les plugins.

Autre nouveauté, en mode édition, SPIP 3 prend en compte le retour à la ligne simple (il n'est désormais plus nécessaire d'utiliser « _ »).

Enfin, l'espace privé intègre un mécanisme technique de thèmes qui permet de personnaliser les feuilles de style ou tout ou partie des icônes. Ce mécanisme pourra être utilisé par un plugin proposant des variantes activables par chaque utilisateur en fonction de ses préférences.

Modularisation en plugins

SPIP 3 achève donc la découpe du logiciel en plugins. Son noyau conserve la gestion des articles, rubriques et auteurs ; le langage de squelettes et l'ossature de l'espace privé. Toutes les autres fonctionnalités sont externalisées dans des plugins, automatiquement installés dans la distribution par défaut de **SPIP 3**. Ces plugins fournis par défaut sont désormais placés dans un dossier `plugins-dist/` [5]

Cette découpe du noyau a permis de compléter un certain nombre d'API implicites, maintenant à disposition des développeurs de plugins qui peuvent ainsi ajouter des fonctionnalités sans limite, en prenant modèle sur les plugins natifs de SPIP.

Dans le domaine des plugins, **SPIP 3** amène également une refonte importante du formalisme de déclaration XML. Outre une lisibilité améliorée, cette évolution intègre la question des traductions de plugins, ainsi que les outils d'alimentation automatique d'un annuaire des plugins de SPIP et un outil de chargement.

Par ailleurs, les plugins peuvent bénéficier des fonctions de mise à jour de base de données du core (avec reprise sur timeout) par simple déclaration de la procédure.

Les plugins distribués par défaut avec SPIP sont :

- **Brèves** s'occupe de l'objet éditorial Brèves.
- **Compagnon** propose l'affichage de textes pédagogiques d'information et d'accompagnement lors des premiers accès à l'espace privé.
- **Compresseur** déjà présent dans SPIP 2.1, prend en charge l'optimisation des performances du site en compressant et concaténant les feuilles de style CSS et les fichiers JavaScript embarqués dans la page. Cette version du plugin améliore des feuilles par média. Utilisation de @media pour concaténer tous les médias en un seul fichier sans en modifier l'ordre de chargement. Gestion des urls absolues sans protocole pour que les feuilles de style compressées fonctionnent sur http et https.
- **Dump** assure la gestion des sauvegardes et restaurations. La fonctionnalité a été complètement ré-écrite pour assurer une sauvegarde complète et fiable. Le format de sauvegarde est maintenant SQLite et toutes les tables sont systématiquement conservées. Votre hébergement doit donc disposer de SQLite. Si ce n'est pas le cas, réalisez vos sauvegardes via phpmyadmin ou toute autre interface de gestion de la base de données fournie par votre hébergeur.
- **Forum** gère l'objet éditorial *forum*, tant pour le site public que pour les interactions éditoriales dans l'espace privé. Outre une interface de modération repensée, les forums sont maintenant utilisables sur tous les objets éditoriaux de SPIP (natifs ou ajoutés par des plugins).
- **Images**, déjà dans SPIP 2.1, prend en charge tous les filtres d'images et de couleurs dans les squelettes.
- **jQuery UI** implémente dans SPIP la librairie complémentaire. jQuery UI facilite la création de composants graphiques dynamiques : onglets, drag & drop, barres de progression, widgets, effets… [6]
- **Mediabox** intègre dans SPIP par défaut une boîte *pop-in* pour visualiser les médias ou proposer des interactions [7].
- **Medias [8]** prend en charge la gestion des documents et images. Il propose une refonte de l'interface et rend les documents utilisables sur n'importe quel objet éditorial.
- **Mots** apporte la gestion des mots-clés et groupes de mots. Les mots-clés sont maintenant utilisables sur tous les objets éditoriaux.
- **Organiseur** assure les fonctions de messagerie et calendrier interne de l'espace privé. L'interface est complètement refondue. La messagerie interne bénéficie de notification par email, et le calendrier repose sur la librairie FullCalendar qui permet une navigation fluide.
- **Pétitions** gère les pétitions sur les articles et propose une interface de modération repensée.

- **Porte plume**, déjà dans SPIP 2.1, assure l'aide à la saisie.
- **Révisions** assure le versionage utilisable sur tous les objets éditoriaux de SPIP. L'interface de gestion des révisions est modernisée.
- **SafeHTML**, déjà dans SPIP 2.1, prend en charge la sécurisation des contenus externes potentiellement dangereux.
- **Sites** assure le fonctionnement de l'objet éditorial « Site syndiqué » ainsi que des articles syndiqués. Une interface de modération des articles syndiqués est proposée.
- **Squelettes par rubriques** assure le fonctionnement des squelettes suffixés (article-2.html pour les articles de la rubrique 2 et de ses sous-rubriques).
- **Statistiques** assure le calcul des statistiques du site et des articles, et propose un affichage complètement refondu.
- **Support vieux navigateur**, déjà dans SPIP 2.1, propose des fonctions JavaScript activables pour permettre aux vieilles versions de navigateurs d'afficher correctement votre site.
- **Svp** permet la gestion complète des plugins : installation, activation, mise à jour, recherche…
- **Textwheel** est un moteur typographique qui supporte intégralement les raccourcis de SPIP, décrits désormais dans un fichier de règles au format YAML. Ce moteur permet une nette accélération du traitement des raccourcis (jusqu'à 2 fois plus rapide sur certains contenus), et facilite l'évolution et la personnalisation des raccourcis.
- **Urls Étendues** supporte les URLs propres ou arborescentes, et propose une interface de configuration ainsi qu'une interface de gestion avancée optionnelle pour gérer finement les URLs de chaque page.
- **Vertèbres**, déjà dans SPIP 2.1, permet au webmestre d'afficher le contenu d'une table SQL de SPIP au moyen d'un squelette généré automatiquement d'après la structure SQL de la table. Cet affichage se fait maintenant dans l'espace privé.

Nouveaux squelettes par défaut

Pour marquer le coup, les squelettes par défaut ont été révisés. Par la répartition en plusieurs feuilles de styles ré-utilisables indépendamment (méthode Daisy , cf. Un, deux, trois… feuilles CSS !), ils constituent désormais un cadre de travail HTML/CSS modulaire, immédiatement utilisable pour démarrer un site. Ils proposent notamment un traitement typographique complet et harmonieux, inspiré du framework Blueprint. L'historique `spip_style.css` disparaît au profit de **`spip.css`**, qui peut être utilisée en complément de votre framework habituel.

Les squelettes adoptent une nouvelle structure HTML de base qui anticipe le HTML5, « responsive » par défaut, et embarque des sélecteurs conditionnels pour plus de souplesse CSS.

De nouvelles fonctionnalités

Le support de SQLite a été nettement amélioré, et c'est le format de base de données proposé par défaut lors d'une nouvelle installation (quand il est supporté par le serveur), compte tenu de sa simplicité de mise en œuvre [9].

La gestion des objets éditoriaux est généralisée. Par suite, les auteurs, documents, mots-clés, forums, révisions et logos sont utilisables sur n'importe quel objet (tous ceux de SPIP mais aussi tout objet éditorial ajouté par un plugin avec la nouvelle API de déclaration). La gestion du statut, de la date de publication, de la langue et de la traduction est également généralisée et utilisable sur tout objet par simple déclaration. Lorsque le signalement de l'édition concourante est activé, tout objet édité est désormais pris en compte dans le signalement.

L'identité du site est complétée d'un slogan du site, entre titre et descriptif long, utilisable avec la balise **`#SLOGAN_SITE_SPIP`**.

L'écran de sécurité est intégré en standard.

Les logs (fichiers de trace) bénéficient d'une gestion par niveau d'importance. Par défaut, les logs sont beaucoup moins verbeux en production et ne signalent que les anomalies ou informations importantes. Le niveau de log peut être ajusté pour le debug ou le développement, permettant ainsi d'avoir toutes les informations nécessaires.

Le cron périodique qui reposait sur des fichiers a été supprimé au profit d'une file de tâches en attente (qui gère aussi les tâches périodiques). Cette file de tâches permet de programmer des actions asynchrones [10] et propose une API simple d'utilisation pour les développeurs de plugin.

L'ajax des squelettes (`{ajax}` sur les `INCLURE`) bénéficie d'une prise en charge des attributs ARIA pour améliorer son accessibilité. De plus, l'historique de navigation est aussi automatiquement pris en charge grâce à l'API « History » de HTML5 [11]. L'URL du navigateur est donc mise à jour automatiquement sur les liens ajax, et permet les retours en arrière sans soucis.

Les modèles utilisés dans du texte éditorial reçoivent automatiquement l'environnement du squelette dans lequel est affiché le texte [12].

Les formulaires des l'espace privé utilisent les possibilités de HTML5 avec par exemple les attributs `required` ou `placeholder`.

Les squelettes peuvent maintenant échapper les caractères `# [ ] ( ) { } < >` en utilisant une barre oblique `\`. Il devient possible d'écrire des conditions par exemple sur une case à cocher de formulaire de manière bien plus élégante, sans que les crochets de l'attribut *name* n'interfèrent avec

les crochets de la balise englobante :

```
[(#ENV{param}|oui)
   <label for='tendresse'>Avec tendresse ?</label>
   <input type='checkbox' name='avec_quoi\[\]' id='tendresse'
checked='checked' value='tendresse' />
]
```

Raccourcis

Un raccourci pour les abréviations a été introduit nativement : `[SNCF|Société Nationale des Chemins de fer Français]` (ou `[CMS|Content Management System{en}]`) pour indiquer une langue différente du texte principal) génère une balise **abbr** [13].

Les modèles de document peuvent prendre en argument la largeur ou la hauteur pour réduire leur dimension : **<docxx|largeur=150>**.

Les tableaux à 2 entrées (horizontale et verticale) sont désormais gérés correctement.

Il n'est désormais plus nécessaire d'utiliser le raccourci « _ » pour produire un retour ligne simple. Pour retrouver le fonctionnement antérieur, ce nouveau comportement est débrayable en utilisant un `define('_AUTOBR', '');` dans son fichier `mes_options.php`.

Boucles

Le système de boucles de SPIP a été généralisé pour s'appliquer non plus directement sur une table SQL mais sur un itérateur. Un itérateur SQL assure le fonctionnement historique des boucles sur les tables SQL, mais il devient possible de boucler sur toute donnée itérable : Les itérateurs de SPIP

Premier exemple de nouvelle application : la boucle **DATA** étend le fonctionnement des boucles à toute information structurée sous forme de tableau. Il devient possible de boucler sur un fichier CSV, sur une url distante qui renvoie une information en JSON... :

- Exemples de `BOUCLE(DATA)`
- http://zzz.rezo.net/La-boucle-iCalendar.html
- http://zzz.rezo.net/Exemples-de-boucles-YQL.html

Pour assurer la migration facile des squelettes qui utilisaient des boucles `POUR` et `CONDITION`, celles-ci sont également supportées comme cas particuliers de la boucle `DATA`, dans la même syntaxe que celle qui existait.

Critères

{si ...} : permet de conditionner l'exécution d'une boucle à la condition exprimée dans le critère, qui ne dépend pas des données en base mais du contexte du squelette.

{tri ...} : associé à la balise `#TRI` pour des tris faciles.

{feuille} : permet de sélectionner les rubriques sans enfants (celles tout en bas de la hiérarchie).

{noeud} : permet de sélectionner les rubriques qui ont des enfants.

{!racine} : exclut les rubriques de la racine.

{profondeur=3} : permet de sélectionner les rubriques du 3e niveau (la racine est le niveau 0, les rubriques secteurs sont le niveau 1, puis les rubriques du dessous le niveau 2 etc ...).

Balises

Les modèles peuvent avoir un cache s'ils contiennent une balise **#CACHE** (mais par défaut ils n'en ont pas, comme auparavant).

Dans l'espace privé, les squelettes n'ont pas de cache non plus, sauf si une balise `#CACHE` est présente.

#LOGO_DOCUMENT peut prendre en argument le mode d'affichage du logo :

- *auto* (par défaut, et c'est le fonctionnement historique) affiche automatiquement la vignette du document si existante, sinon un aperçu ; et sinon l'icône correspondant au type de document.
- *icone* indique que c'est l'icône correspondant au type du fichier qui doit être affichée
- *apercu* affiche un aperçu de l'image exclusivement, même si une vignette existe.
- *vignette* affiche la vignette du document si elle existe, ou sinon rien.

La balise **#SPIP_CRON** disparaît et n'a plus d'effet dans les squelettes où elle serait encore utilisée.

#BOUTON_ACTION{libellé, url, classe, confirm, title, callback} génère un mini formulaire HTML avec un seul bouton qui affiche « *libellé* » et déclenche au clic un POST vers « *url* ». Cette balise est à utiliser de préférence à un lien quand la page « *url* » modifie la base de données. Si la « *classe* » contient la valeur « *ajax* » le bouton déclenchera un rechargement du bloc ajax qui l'inclue. « *confirm* » permet d'indiquer un message pour faire confirmer l'action par l'utilisateur, « *title* » le contenu de l'attribut homonyme sur le bouton, et « *callback* » une fonction javascript à appeler lors du clic sur le bouton.

#INFO_XXX{article, 13} : permet de retrouver le `#XXX` sans faire une boucle `ARTICLES` sur l'article `13` (utilisable pour toute boucle et tout champ de la boucle : `#INFO_TITRE{article,13}`, `#INFO_NOM{auteur,2}`...)

#CONFIG{nom} permet d'afficher la valeur de la meta de configuration *nom*. Si la meta est un tableau, il est possible d'accéder directement à des sous valeurs avec la syntaxe `#CONFIG{nom/sousvaleur}` qui renverra la *sousvaleur* de la meta *nom*. Pour accéder à la meta d'une table spécifique à un plugin, il faut utiliser la syntaxe `#CONFIG{/metamonplugin/nom}` : en commençant par un `/`, on indique que l'on veut la meta nom de la table spip_metamonplugin au lieu de la table spip_meta de SPIP. Dans tous les cas, il est possible d'indiquer en second argument la valeur par défaut qui doit être utilisée si la meta n'existe pas encore : `#CONFIG{nom,valeurpardefaut}`

À l'instar de `#CONFIG{nom/sousvaleur}` les balises **#ENV**, **#GET**, **#SESSION** permettent d'utiliser des barres obliques pour obtenir des sous valeurs de tableau tel que :`#ENV{nom/sousvaleur}`, `#GET{tableau/cle/souscle}`, `#SESSION{prefs/couleur}`.

#PUBLIE : pour tester l'état (publié ou non) d'un objet. S'utilise dans une boucle, porte implicitement sur l'objet en cours ou avec des arguments explicites : `[(#PUBLIE{article, 3}|oui) ... ]`, sur tout autre objet.

#CLE et **#VALEUR** sont les deux balises qui permettent d'afficher la clé et la valeur dans une boucle `DATA` `#VALEUR{x}` permet d'afficher la sous valeur de x si `#VALEUR` est un tableau (équivalent à `[(#VALEUR|table_valeur{x})]`). On peut enchaîner plusieurs sous index `#VALEUR{x/y/z}` pour afficher une valeur dans un sous-niveau du tableau.

#PRODUIRE : renvoie le nom d'un fichier statique produit à partir d'un squelette. Utile pour une feuille de style calculée ou un fichier javascript calculé. La syntaxe est la même que pour la balise `#INCLURE` : `#PRODUIRE{fond=mafeuille.css,couleur=ffffff}` pour utiliser le squelette `mafeuille.css.html`

#LISTE{a,b,c} renvoie simplement un tableau contenant les valeurs a,b et c. C'est une écriture simplifiée de `#ARRAY{0,a,1,b,2,c}`.

#TOTAL_UNIQUE renvoie le nombre d'éléments affichés par l'intermédiaire du filtre `|unique`. Si on utilise `|unique{nom}`, il faut faire référence au même nom dans la balise : `#TOTAL_UNIQUE{nom}`

Dans l'espace privé :

#AIDER{surtitre} permet d'afficher un lien vers la section *surtitre* de l'aide de SPIP.

#BOITE_OUVRIR, **#BOITE_PIED** et **#BOITE_FERMER** permettent d'utiliser les boites stylées de l'espace privé :

- `#BOITE_OUVRIR{titre,classe}` ouvre une boîte. le *titre* peut être omis. Les classes stylées dans l'espace privé sont : simple, info, note, raccourcis et important ;
- le HTML qui suit est inclus dans la boîte ;
- `#BOITE_PIED` permet de passer au pied de la boîte quand un pied est nécessaire.
- `#BOITE_FERMER` ferme la boîte.

Pour de nombreux exemples, voir la page de l'espace privé `ecrire/?exec=charte_boites` proposée par le plugin http://plugins.spip.net/dev

#FORMULAIRE_RECHERCHE_ECRIRE affiche le formulaire de recherche de l'espace privé. On peut lui indiquer en premier argument l'url vers laquelle il doit pointer, et une classe en second argument. En présence de javascript le formulaire se comporte comme un lien vers url avec le paramètre recherche fourni par la saisie. Si on indique la classe ajax en second argument, le formulaire provoque le rechargement du bloc ajax incluant, comme un lien ajax donc : `#FORMULAIRE_RECHERCHE_ECRIRE{#SELF, ajax}`

#CHEMIN_IMAGE{article-24.png} renvoie le chemin vers l'icône article-24.png du thème de l'espace privé en cours d'utilisation par l'auteur connecté.

Filtres

|lien_ou_expose permet de construire simplement un menu de plusieurs liens en exposant celui qui est sélectionné par un `<strong>` et en gardant un lien `<a>` sinon. Exemple : `[(#URL_PAGE{mapage}|lien_ou_expose{libelle, #ENV{page}|==}{mapage}, classe, title, rel})]`

|singulier_ou_pluriel affiche une chaîne ou une autre en fonction du nombre sur lequel il

est appliqué :
`[(#TOTAL_BOUCLE|singulier_ou_pluriel{1_article,nb_articles})]`.
1_article et *nb_articles* sont des chaînes de langue qui reçoivent le nombre en argument `@nb@`. Si le nombre est zéro, le filtre n'affiche rien.

|balise_img permet de construire rapidement une balise HTML `<img>` à partir d'un nom de fichier en renseignant systématiquement le width et le height pour accélérer le rendu de la page :
`[(#CHEMIN{monimage.png}|balise_img{alt,class})]`

|affdate_debut_fin affiche sous forme sympathique un intervalle de temps entre une date de début et une date de fin, en prenant en compte le fait que la date de début et la date de fin sont ou non le même jour, le même mois, la même année, et en prenant en compte l'affichage de l'heure ou non (second argument, *oui* ou *non*. Exemple :
`[(#DATE_DEBUT|affdate_debut_fin{#DATE_FIN,horaireouinon})]`

|timestamp ajoute au nom d'un fichier un horodatage sous la forme ?1234567890 où le nombre représente la date du fichier compté en secondes depuis le 1er janvier 1970. Ce filtre est utile pour référencer par exemple des feuilles de styles en s'assurant que le navigateur les rechargera quand leur url change : `[<link rel="stylesheet" href="(#CHEMIN{css/perso.css}|timestamp)" type="text/css" />]`

|objet_icone renvoie l'icône standard d'un objet de SPIP. Par défaut, la taille 24px est renvoyée sauf si la taille 16 ou 32 est demandée en argument :
`[(#OBJET|objet_icone{16})]`

|objet_info renvoie une propriété d'un objet de SPIP telle que déclarée (ou automatiquement renseignée par SPIP) via l'API *declarer_tables_objets_sql*

|objet_afficher_nb affiche le nombre d'objets en prenant en compte le nombre auquel le filtre s'applique et les chaînes de langue déclarées pour l'objet :
`[(#TOTAL_BOUCLE|objet_afficher_nb{auteur})]` affichera par exemple *1 auteur* ou *23 auteurs*.

|wrap encadre un texte d'une balise html : `[(#TITRE|wrap{<h3>})]` produit le même résultat que `[<h3>(#TITRE)</h3>]`

|generer_info_entite affiche le champ d'un objet SPIP :
`[(#ID_ARTICLE|generer_info_entite{article,titre})]` renvoie le titre mis en forme de l'article `#ID_ARTICLE`. C'est équivalent ici à
`[(#INFO_TITRE{article,#ID_ARTICLE})]`.

Dans l'espace privé :

|icone_horizontale affiche une icône de l'espace privé au format horizontal. La syntaxe est : `[(#URL|icone_horizontale{libelle,icone,fonction})]`. *icone* peut être désigné en abrégé (*article*) et sera alors au format 24px ou explicitement (*article-24.png*). Le 3ème argument *fonction* précise une famille d'action associée à l'icône et peut être au choix *add*, *del*, *edit*, *new*, ou omis. Exemple
`[(#URL_ECRIRE{auteur_edit,new=oui}|icone_horizontale{<:icone_creer_nouvel_auteur:>,auteur,r`

|icone_verticale affiche une icône de l'espace privé au format vertical. La syntaxe est la même que pour `|icone_horizontale`

|bouton_action_horizontal utilise la même syntaxe que `|icone_horizontale` et affichera un résultat visuellement semblable. Cependant le markup HTML sera du même type que celui de `#BOUTON_ACTION` pour déclencher un POST vers l'url et est donc utile quand l'url modifie la base de données.

|sinon_interdire_acces placé à n'importe quel endroit de la page permet de bloquer l'accès à celle-ci quand l'expression à laquelle il s'applique est fausse. Dans ce cas une page d'erreur est affichée, sauf si une url de redirection est fournie en argument (on peut aussi alors fournir un status http de redirection en second argument). On l'utilise généralement sur une vérification d'autorisation
`[(#AUTORISER{modifier,article,#ID_ARTICLE}|sinon_interdire_acces)]`

JavaScript

SPIP 3 utilise la version 1.7.2 de jQuery et intègre désormais jQuery UI en version 1.8.20. Il devient possible d'utiliser une version différente de jQuery dans le site public en surchargeant simplement `javascript/jquery.js` dans le dossier `squelettes/` sans risquer de casser l'espace privé.

Liens ajax : Les liens `.ajax` ne cassent plus l'historique de navigation sur les navigateurs qui supportent l'API HTML5 History (Firefox, Safari, Chrome à la date de cet article). C'est-à-dire que lorsqu'on clique sur un lien ajax de SPIP qui recharge une partie de la page, l'URL est mise à jour dans le navigateur et le visiteur peut cliquer sur *Précédent* pour revenir en arrière.

Classes spéciales sur les liens ajax :

- `.nohistory` indique que le lien n'affecte pas l'historique de navigation lorsqu'il est cliqué ;

- `.preload` indique à SPIP que le contenu du lien ajax doit être préchargé au moment où la page est chargée. Ainsi le clic sur le lien produira une mise à jour immédiate ;
- `.nocache` indique à SPIP que le contenu du lien ajax ne doit pas être mis en cache. Ainsi plusieurs clics sur le même lien provoqueront autant de chargements depuis le serveur (par défaut, seul le premier chargement à lieu pour une url donnée et le contenu est ensuite mémorisé par le navigateur).

Rechargement télécommandé de blocs ajax : Les liens `.ajax` permettent par défaut le rechargement du bloc ajax qui les contient, mais il est parfois nécessaire de provoquer le rechargement d'un autre bloc ajax de la page.

Pour cela, il devient possible de nommer les blocs ajax au moment de leur inclusion : `<INCLURE{fond=...,ajax=nomdubloc} />`. Le bloc ajax ainsi nommé peut ensuite être rechargé via l'appel de `ajaxReload('nomdubloc');`. Il est possible de passer en second argument une liste d'options contenant :

- callback : fonction callback qui doit être appelée après le chargement ajax du bloc
- args : liste d'argument qui seront passés à l'url lors du chargement du bloc (permet de modifier le `#ENV` du bloc mis à jour) ;
- history : indique si le rechargement affecte ou non l'historique de navigation (faux par défaut). Exemple :

```
ajaxReload('nomdubloc', {
   callback:function(){alert('fini');},
   args:{id_article:3},
   history:false
   });
```

`ajaxReload` peut être utilisé également sur un sélecteur jQuery auquel cas il provoquera le rechargement du plus petit bloc ajax qui contient l'élément ciblé. Il ne prend alors qu'un argument possible (le tableau d'options) : `$('#contenu').ajaxReload({args:{id_article:3}})`

Animations javascript : Lors des interactions déclenchées par l'utilisateur, il est possible de déclencher plusieurs animations type qui s'appliquent toutes sur un sélecteur `jQuery` :

- `jQuery('#monid').animateLoading()` déclenche l'animation de chargement sur `#monid` (automatiquement déclenchée lors des rechargements ajax)
- `jQuery('#monid').endLoading()` arrête l'animation de chargement sur `#monid` (automatiquement déclenchée lors des rechargements ajax)
- `jQuery('#monid').animateAppend()` anime le bloc `#monid` pour montrer qu'il vient d'être ajouté par l'interaction
- `jQuery('#monid').animateRemove()` anime le bloc `#monid` pour montrer qu'il est supprimé par l'interaction

Fonctions utilitaires : `parametre_url()` s'utilise comme sa version PHP de parametre_url pour ajouter, supprimer, récupérer des arguments d'une URL :

- `parametre_url(url,arg,value)` ajoute `arg=value` à `url` et retourne l'url modifiée ;
- `parametre_url(url,arg,'')` supprime la valeur de `arg` dans `url` et retourne l'url modifiée ;
- `parametre_url(url,arg)` retourne la valeur de `arg` dans `url`

`$('a').followLink()` suit le lien en simulant un clic dessus (prend en compte le fait que le lien est ou non un lien ajax).

Popin dans l'espace privé : Un lien vers une page de l'espace privé avec une classe `.popin` provoque l'ouverture d'une popin qui contient le coeur de la page pointée (bloc `contenu/`). Le lien conserve sa capacité à être ouvert dans un autre onglet via clic droit et la page complète est affichée dans ce cas.

API

SPIP 3 introduit plusieurs **API génériques** dans la gestion des objets éditoriaux :

- une gestion générique des tables de liens d'un objet avec n'importe quel autre objet : c'est l'API **editer_liens**
- La création de nouveaux objets éditoriaux est simplifiée par une API déclarative
- une gestion générique des fonctions d'insertion, modification ou publication des objets éditoriaux : l'API **editer_objet**
- **`objet_test_si_publie($objet, $id_objet)`** permet de tester facilement si un objet est publié ou non en fonction de son statut tel que déclaré (et en fonction d'une éventuelle post-publication)
- **`generer_url_ecrire_objet($objet,$id_objet,$args, $ancre)`** renvoie l'url vers la page de l'objet dans l'espace privé. `$args` et `$ancre` sont optionnels.
- le pipeline **`optimiser_base_disparus`** est appelé lors du nettoyage en base des objets à la poubelles et des liens morts qui en résultent

SPIP 3 dispose également d'une **gestion de file de tâches** qui permet de programmer l'exécution future ou simplement asynchone (ASAP mais sans faire attendre l'utilisateur) de fonctions.

La mise en place de configuration des plugins est simplifiée par la prise en charge automatisée des formulaires **#FORMULAIRE_CONFIGURER_XXX** et par l'utilisation des fonctions **lire_config**, **ecrire_config**, **effacer_config** (voir Configurer une fonctionnalité de votre site, ou un plugin).

Les formulaires CVT permettent facilement de faire des formulaires en plusieurs pages, et supportent également la sauvegarde automatique de la saisie par simple déclaration. Un pipeline **formulaire_fond** permet de personnaliser le fond des formulaires (pour modifier le HTML du formulaire) [14]

Dans l'API SQL :

- la fonction **sql_skip** est ajoutée et permet de sauter un certain nombre de résultats.
- Les valeurs `null` dans sql_updateq et sql_insertq sont traduites en **NULL** SQL et ne sont plus assimilées à une chaîne vide.
- les fonctions **sql_demarrer_transaction** et **sql_terminer_transaction** permettent respectivement comme leur nom le suggère de démarrer ou de clôturer une transaction SQL [15].

Introduction d'un point d'entrée bac-à-sable, (**public/sandbox.php**) pour la compilation des squelettes, permet à un plugin de gérer la compilation des squelettes dans un ensemble de filtres et fonctions en liste blanche ou liste noire, et d'interdire le PHP dans les squelettes.

Spécialisation des critères : Il est possible de préfixer par le nom du serveur et/ou le nom de la table le nom de la fonction critère. La recherche se fait dans l'ordre :

1. `critere_serveur_TABLE_moncritere_dist`,
2. `critere_serveur_TABLE_moncritere`,
3. `critere_serveur_moncritere_dist`,
4. `critere_serveur_moncritere`,
5. `critere_TABLE_moncritere_dist`,
6. `critere_TABLE_moncritere`,
7. `critere_moncritere_dist`,
8. `critere_moncritere`

Dans le fichier `options.php` d'un plugin ou dans le fichier `mes_options.php`, **`$GLOBALS['marqueur_skel'] .= ":prefixe"`** permet de différencier le cache des squelettes compilés comme `$GLOBALS['marqueur'] .= ":prefixe"` permettait déjà de différencier le cache des squelettes.

Notes

[1] Outre le Porte-plume déjà intégré dans SPIP 2 et qui initiait ce mouvement, on peut citer notamment les plugins Afficher_objets, Base CSS, Navigation du privé, CVT multi-étapes, Forum, Job-queue, Mediabox, Médiathèque, TextWheel, URLs éditables, Comments, qui ont, pour certains, inspiré fortement le développement de SPIP lui-même ; pour d'autres été intégrés, avec quelquefois certaines évolutions.

[2] Ce chantier avait été amorcé depuis quelque temps sur la Zone, et celles et ceux qui d'entre vous utilisaient déjà le plugin *Navigation du privé* ou la *médiathèque* retrouveront vite leurs marques.

[3] Par exemple, le passage en édition sur un article devient instantané grâce au pré-chargement Ajax.

[4] http://core.spip.org/projects/spip/repository/revisions/17650

[5] en SPIP 2.1 ils étaient placés dans le dossier `extensions/` désormais inutile.

[6] Voir le pipeline associé sur programmer.spip.org

[7] C'est une alternative aux plugin Thickbox, LightBox... qui repose sur la librairie Colorbox

[8] Connu pour SPIP 2 sous le nom "Médiathèque"

[9] en revanche le support de PostGreSQL doit être considéré comme expérimental et incomplet pour un usage en production

[10] Par exemple, pour l'envoi d'un mail "dès que possible" qui permet de rendre la main à l'utilisateur immédiatement sans devoir attendre que le mail soit effectivement envoyé

[11] dans tous les navigateurs modernes à l'exception de Internet Explorer qui ne la supporte pas encore

[12] Voir Utiliser les modèles

[13] La balise <acronym> n'est plus utilisée en HTML5.

[14] http://core.spip.org/projects/spip/repository/revisions/18754

[15] Est ajouté de manière expérimentale également `sql_preferer_transaction` présent plutôt pour contourner un problème de lenteur sur des insertions multiples avec SQLite, qui sont exécutées plus rapidement en mode transactionnel. Cette fonction renvoie `true` uniquement en SQLite