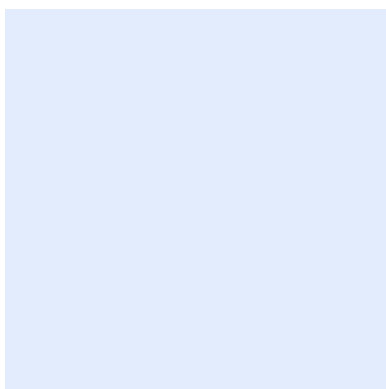


Référence SPIP/N/2017.003 – Ed. 1.3



## **GUIDE CONCEPTION DU PLUGIN NOIZETIER**



**FICHE D'IDENTIFICATION**

<b>Rédacteur</b>	Eric Lupinacci
<b>Projet</b>	SPIP
<b>Étude</b>	Conception du plugin noiZetier
<b>Nature du document</b>	Guide
<b>Date</b>	15/01/2019
<b>Nom du fichier</b>	Guide N - Le plugin noiZetier.docx
<b>Référence</b>	SPIP/N/2017.003 – Ed. 1.3
<b>Dernière mise à jour</b>	19/09/2019 17:06:57
<b>Langue du document</b>	Français
<b>Nombre de pages</b>	43

## TABLE DES MATIERES

<b>1.</b>	<b>INTRODUCTION</b>	<b>6</b>
<b>2.</b>	<b>CONCEPTS</b>	<b>6</b>
<b>2.1</b>	<b>PAGES SPIP, COMPOSITIONS ET BLOCS Z</b>	<b>6</b>
2.1.1	LES PAGES SPIP	6
2.1.2	LES COMPOSITIONS	6
2.1.3	LES BLOCS Z	7
<b>2.2</b>	<b>LES PAGES DU NOIZETIER</b>	<b>7</b>
2.2.1	LES PAGES EXPLICITES	7
2.2.2	LES COMPOSITIONS VIRTUELLES	7
<b>2.3</b>	<b>LES BLOCS DU NOIZETIER</b>	<b>7</b>
<b>2.4</b>	<b>UTILISATION DES CONCEPTS N-CORE DANS LE NOIZETIER</b>	<b>8</b>
<b>3.</b>	<b>PERIMETRE DU NOIZETIER</b>	<b>9</b>
<b>3.1</b>	<b>L'API DE GESTION DES PAGES</b>	<b>9</b>
<b>3.2</b>	<b>L'API DE GESTION DES BLOCS</b>	<b>11</b>
<b>3.3</b>	<b>L'API DE GESTION DES OBJETS</b>	<b>11</b>
<b>3.4</b>	<b>COMPLEMENT DE L'API DE GESTION DES CONTENEURS</b>	<b>12</b>
<b>4.</b>	<b>FONCTIONNEMENT DU NOIZETIER</b>	<b>14</b>
<b>4.1</b>	<b>L'INTERFACE PRIVEE DE GESTION DES PAGES</b>	<b>14</b>
4.1.1	CHARGER LES PAGES	14
4.1.2	AFFICHER LES PAGES	14
4.1.3	EDITER UNE PAGE	15
4.1.4	CREER UNE COMPOSITION VIRTUELLE	15
4.1.5	ACTIVER LES COMPOSITIONS SUR UN TYPE D'OBJET	16
<b>4.2</b>	<b>L'INTERFACE PRIVEE DE GESTION DES NOISETTES</b>	<b>16</b>
4.2.1	AFFICHER LES NOISETTES D'UNE PAGE	16
4.2.2	AJOUTER UNE NOISETTE	17
4.2.3	MODIFIER LE PARAMETRAGE D'UNE NOISETTE	18
4.2.4	SUPPRIMER UNE NOISETTE	18
4.2.5	DEPLACER UNE NOISETTE	18
4.2.6	DUPLIQUER UNE NOISETTE AU SEIN DE SON CONTENEUR	18
4.2.7	PROPAGER UNE NOISETTE DANS D'AUTRES PAGES	19
4.2.8	VIDER UN BLOC OU UNE PAGE	19
<b>4.3</b>	<b>L'INTERFACE PRIVEE DE GESTION DES CONTENUS</b>	<b>19</b>
<b>4.4</b>	<b>LA CONFIGURATION DU NOIZETIER</b>	<b>19</b>
4.4.1	CONFIGURER LE NOIZETIER	19

4.4.2	EXPORTER OU IMPORTER DES DONNEES	20
4.5	L'INTERFACE PUBLIQUE DE GESTION DES NOISETTES	21
5.	<b>DONNEES DU NOISETIER</b>	<b>22</b>
5.1	<b>LA STRUCTURE DES DONNEES</b>	<b>22</b>
5.1.1	LES PAGES	22
5.1.2	LES BLOCS	23
5.1.3	LES TYPES DE NOISETTES	23
5.1.4	LES NOISETTES	24
5.2	<b>LES ESPACES DE STOCKAGE DU NOISETIER</b>	<b>24</b>
6.	<b>MISE EN OEUVRE DES SERVICES N-CORE</b>	<b>25</b>
6.1	<b>LA GESTION DES TYPES DE NOISETTE</b>	<b>25</b>
6.1.1	LE SERVICE <code>TYPE_NOISETTE_COMPLETER()</code>	25
6.1.2	LE SERVICE <code>TYPE_NOISETTE_STOCKER()</code>	25
6.1.3	LE SERVICE <code>TYPE_NOISETTE_DECRIRE()</code>	26
6.1.4	LE SERVICE <code>TYPE_NOISETTE_LISTER()</code>	26
6.1.5	LE SERVICE <code>TYPE_NOISETTE_INITIALISER_AJAX()</code>	27
6.1.6	LE SERVICE <code>TYPE_NOISETTE_INITIALISER_INCLUSION()</code>	28
6.1.7	LE SERVICE <code>TYPE_NOISETTE_INITIALISER_DOSSIER()</code>	28
6.1.8	CONCLUSION	28
6.2	<b>LA GESTION DES CONTENEURS</b>	<b>28</b>
6.2.1	LE SERVICE <code>CONTENEUR_IDENTIFIER()</code>	29
6.2.2	LE SERVICE <code>CONTENEUR_CONSTRUIRE()</code>	29
6.2.3	LE SERVICE <code>CONTENEUR_VERIFIER()</code>	30
6.2.4	LE SERVICE <code>CONTENEUR_DESTOCKER()</code>	30
6.2.5	CONCLUSION	31
6.3	<b>LA GESTION DES NOISETTES</b>	<b>31</b>
6.3.1	LE SERVICE <code>NOISETTE_COMPLETER()</code>	31
6.3.2	LE SERVICE <code>NOISETTE_RANGER()</code>	32
6.3.3	LE SERVICE <code>NOISETTE_CHANGER_CONTENEUR()</code>	32
6.3.4	LE SERVICE <code>NOISETTE_STOCKER()</code>	33
6.3.5	LE SERVICE <code>NOISETTE_DESTOCKER()</code>	33
6.3.6	LE SERVICE <code>NOISETTE_DECRIRE()</code>	34
6.3.7	LE SERVICE <code>NOISETTE_LISTER()</code>	35
6.3.8	LE SERVICE <code>NOISETTE_INITIALISER_ENCAPSULATION()</code>	36
6.3.9	CONCLUSION	36
7.	<b>MISE EN OEUVRE DE L'API N-CORE</b>	<b>37</b>
7.1	<b>LA GESTION DES TYPES DE NOISETTE</b>	<b>37</b>
7.1.1	CHARGER LES TYPES DE NOISETTES	37
7.1.2	LIRE UN TYPE DE NOISETTE	37

7.1.3	REPERTORIER LES TYPES DE NOISETTE	37
<b>7.2</b>	<b>LA GESTION DES CONTENEURS</b>	<b>37</b>
7.2.1	IDENTIFIER UN CONTENEUR	37
7.2.2	TESTER UN CONTENEUR	38
7.2.3	VIDER UN CONTENEUR	38
7.2.4	LES BESOINS SPECIFIQUES DU NOISETIER	38
<b>7.3</b>	<b>LA GESTION DES NOISETTES</b>	<b>38</b>
7.3.1	AJOUTER UNE NOISETTE A UN CONTENEUR	38
7.3.2	SUPPRIMER UNE NOISETTE D'UN CONTENEUR	39
7.3.3	DEPLACER UNE NOISETTE DANS UN CONTENEUR	39
7.3.4	DUPLIQUER UNE NOISETTE DANS UN CONTENEUR	39
<b>7.4</b>	<b>LA COMPILATION DES NOISETTES</b>	<b>39</b>
<b>7.5</b>	<b>LE MODE VOIR_NOISETTES</b>	<b>39</b>
<b>7.6</b>	<b>CONCLUSION</b>	<b>40</b>
<b>8.</b>	<b>REGLES DE CODAGE</b>	<b>41</b>
<b>8.1</b>	<b>NOMMAGE DES FONCTIONS</b>	<b>41</b>
<b>8.2</b>	<b>ARGUMENTS STANDARDISES</b>	<b>41</b>
<b>9.</b>	<b>LES FICHIERS YAML / XML</b>	<b>42</b>
<b>9.1</b>	<b>LES TYPES DE PAGE EXPLICITES</b>	<b>42</b>
9.1.1	LE NOUVEAU FICHIER YAML	42
9.1.2	LE FICHIER XML (DEPRECIE)	42
<b>9.2</b>	<b>LES COMPOSITIONS EXPLICITES</b>	<b>42</b>
<b>9.3</b>	<b>LES BLOCS Z</b>	<b>43</b>

# 1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception de la branche 3 du plugin noiZetier basée sur le framework N-Core de gestion des noisettes.

Outre l'application du plugin N-Core, cette nouvelle branche du noiZetier a été l'occasion de recoder la gestion des pages, des compositions et des blocs et de revoir toute l'interface utilisateur du privé et du public.

Cette branche du noiZetier est **dédiée aux squelettes Z**.

## 2. CONCEPTS

### 2.1 Pages SPIP, compositions et blocs Z

#### 2.1.1 Les pages SPIP

Par défaut, pour SPIP, une page est un **squelette HTML unique**, installé à la racine d'un dossier de squelettes du path et contenant trois éléments de base, à savoir :

- une balise `<html>` englobante précédée en général d'un doctype,
- une balise `<head>` include contenant les meta, CSS et autres inclusions JS,
- une balise `<body>` include contenant les éléments visibles de la page.

Une page est appelée via une url dont le chemin d'accès peut être exprimé de la façon suivante : `spip.php?page=identifiant_page`. Par exemple,

- pour la page d'accueil l'identifiant de la page étant `sommaire` le chemin d'accès s'exprimera par `spip.php?page=sommaire`,
- et pour un article l'identifiant de la page coïncidant avec le type d'objet `article`, le chemin d'accès s'exprimera par `spip.php?page=article&id_article=5`.

#### 2.1.2 Les compositions

Le plugin Compositions a étendu le concept de variantes de page (pour une rubrique) par la notion de composition de page qui permet d'afficher un squelette différent pour des objets d'un même type. Par exemple, il peut être nécessaire d'afficher certains objets article comme des articles de journal et certains autres comme des albums photos.

Une **composition est donc une page SPIP** dont l'identifiant est composé du type d'objet et d'un suffixe qui caractérise sémantiquement la composition. Pour mettre en œuvre l'exemple précédent la composition affichant l'album photos pourrait se nommer `article-album` et être désignée par le chemin d'accès `spip.php?page=article-album`, l'affichage de l'article de journal étant lui assuré par la page par défaut de l'objet, à savoir, la page `article`.

Dans l'identifiant de la composition `article-album` on reconnaît l'identifiant `article` du type d'objet (appelé aussi **type de page**) et le complément sémantique de la composition, soit `album`.

### 2.1.3 Les blocs Z

Le framework Z a fait émerger la notion de bloc, structurant un layout unique (le contenu de la balise `<body>`) appliqué à chacune des pages du site. Une page sous Z est donc devenue une collection de squelettes, un par bloc, dont le nom coïncide avec l'identifiant de la page concernée. Cela fonctionne aussi nativement avec les compositions.

Ainsi, pour la page article il existe plusieurs fichiers `article.html` répartis dans un dossier au nom du bloc (`content/article.html`, `aside/article.html`...). Si un fichier `article.html` n'existe pas pour un bloc du layout, Z utilisera, par défaut, le fichier `dist.html` du bloc.

Un des blocs Z, celui qui affiche le contenu principal de la page (`contenu` pour Z v1 et souvent `content` pour Z v2 même si il est possible dans cette version de choisir le nom) joue un rôle particulier. Il doit toujours exister et de fait c'est lui qui permet de « repérer » une page ou une composition sous Z. L'absence du **bloc principal** pour une page Z donnée provoque l'affichage d'une 404.

## 2.2 Les pages du noiZetier

### 2.2.1 Les pages explicites

Pour fonctionner, le noiZetier répertorie en premier la liste des pages « disponibles » pour recevoir des noisettes. Pour cela, il repère le fichier YAML ou XML ou à défaut le fichier HTML décrivant la page. Sous Z, il va donc chercher ces fichiers dans le bloc principal. Comme les compositions possèdent toujours un fichier XML elles sont nativement listées si le plugin Compositions est activé.

Dans la suite du document ces pages sont appelées **pages explicites**. Une page explicite peut donc être une **composition explicite** ou un **type de page explicite**.

### 2.2.2 Les compositions virtuelles

Le noiZetier possède une fonction permettant de générer à la volée une composition à partir d'une page explicite. Une telle **composition virtuelle** ne possède aucun fichier descriptif ni fichier HTML propre mais juste une liste de noisettes qui seront insérées dans le HTML du type page comme `article.html`. L'identifiant d'une composition virtuelle est similaire à celui d'une composition explicite.

Dans le noiZetier, pages explicites et compositions virtuelles sont désignées par le terme **pages**.

## 2.3 Les blocs du noiZetier

Le noiZetier s'appuie sur Z pour lister les blocs par défaut du layout unique utilisé (liste prédéfinie en v1 et variable globale `$GLOBALS['z_blocs']` en v2). Il est possible de personnaliser la liste des blocs par défaut pouvant accueillir des noisettes :

- globalement pour toutes les pages en utilisant le pipeline `noizetier_blocs_defaut()` ;
- et page par page, en définissant une liste de blocs exclus de l'ajout de noisettes via un formulaire dédié. La configuration, pour les pages explicites, d'une liste de blocs autorisés dans le fichier YAML ou XML a été dépréciée dans cette version.

En outre, cette nouvelle branche v3 du noiZetier permet de décrire plus précisément les blocs Z en leur associant un fichier YAML à l'instar des pages explicites. Ce fichier YAML est nommé `bloc.yaml` et est localisé dans le dossier matérialisant le bloc Z (par exemple, `content/bloc.yaml`).

## 2.4 Utilisation des concepts N-Core dans le noiZetier

Le noiZetier utilise sans restriction ni modification les concepts de N-Core. Cependant, pour les types de noisette et les noisettes, il utilise un espace de stockage propre et complète quelque peu la structure de données comme cela est prévu dans l'API de N-Core (voir le chapitre 0).

Pour les conteneurs, la mise en œuvre est la suivante :

- la base d'un conteneur est toujours un bloc Z ;
- le bloc Z appartient toujours à une page explicite ou à une composition virtuelle ;
- les objets SPIP dont le type est autorisé par configuration peuvent accueillir des noisettes. De fait, chaque bloc du type de page associé à cet objet peut devenir un conteneur.

Par exemple, les blocs `content/article`, `nav/article-forum` et `aside/article` pour l'article 12 peuvent être des conteneurs. L'application de la fonction de calcul de l'identifiant aux conteneurs précités donne les identifiants `content/article`, `nav/article-forum` et `aside/article|article|12`.

Le noiZetier fournit toutes les fonctions de service concernant ses conteneurs spécifiques, pour le calcul de l'identifiant - `conteneur_noizetier_identifier()`, pour la vérification du tableau conteneur - `conteneur_noizetier_verifier()` - et pour la reconstruction du tableau conteneur à partir de l'identifiant - `conteneur_noizetier_construire()`.

Le tableau canonique d'un conteneur du noiZetier possède toujours les mêmes index, à savoir, l'index `squelette` qui est toujours présent (par exemple, `content/article`) et éventuellement les index `objet` et `id_objet` si le conteneur se rapporte à un objet précis.



### 3. PERIMETRE DU NOIZETIER

Le noiZetier propose plusieurs API propres :

- La gestion des pages explicites, à savoir, le chargement des fichiers XML ou YAML, leur stockage, la lecture des informations stockées et la gestion des compositions virtuelles créées à partir d'une page explicite ;
- La gestion des blocs, à savoir, le chargement des fichiers YAML, la lecture des informations et l'autorisation des blocs ;
- La gestion des objets pouvant recevoir des noisettes ;

Le noiZetier reprend aussi les API de N-Core (voir la mise en œuvre des fonctions de service chapitre 6) et les complète par quelques fonctions propres pour la gestion des conteneurs. Il utilise la balise `#NOISETTE_COMPILER` fournie par N-Core lors de la compilation d'un bloc Z (voir le paragraphe 7.4) et surcharge le squelette `conteneur_compiler.html` pour des raisons de performance.

Enfin, le noiZetier fournit toute l'interface utilisateur de manipulation des pages, objets, types de noisettes et noisettes décrite dans le chapitre 4.

#### 3.1 L'API de gestion des pages

La gestion des pages et compositions consiste à stocker les descriptions dans un espace à accès rapide et à permettre leur lecture et leur mise à jour. Elle concerne les pages explicites et les compositions virtuelles.

Elle est composée en premier lieu d'une API fonctionnelle.

#### API PAGES : INC/NOIZETIER\_PAGE.PHP

<b>page_noizetier_charger</b>	Charge ou recharge les descriptions des pages explicites à partir des fichiers YAML ou XML. Les pages sont recherchées dans un répertoire relatif configurable. La fonction optimise le chargement en effectuant uniquement les traitements nécessaires en fonction des modifications, ajouts et suppressions des pages identifiées en comparant les md5 des fichiers YAML ou XML.
<b>page_noizetier_compter_noisettes</b>	Renvoie, pour une page donnée, la liste des blocs ayant des noisettes ajoutées et le nombre de noisettes pour chacun des blocs. Le tableau de sortie est de la forme [bloc] = nombre de noisettes. Les blocs vides ne sont pas fournis.
<b>page_noizetier_extraire_composition</b>	Extrait l'identifiant de la composition ou renvoie une chaîne vide.

<b>page_noizetier_extraire_type</b>	Extrait l'identifiant du type de page. Si la page n'est pas une composition l'identifiant du type de page coïncide avec celui de la page.
<b>page_noizetier_initialiser_dossier</b>	Retourne le dossier relatif dans lequel chercher les pages explicites.
<b>page_noizetier_lire</b>	Retourne la description complète de la page explicite ou de la composition virtuelle demandée. Les champs textuels peuvent être fournis bruts ou avec un traitement typo.
<b>page_noizetier_lister_blocs</b>	Renvoie la liste des blocs de la page pour lequel l'ajout de noisettes est autorisé.
<b>page_noizetier_composition_activee</b>	Détermine si les compositions sont activées (configuration du plugin Compositions) sur un type de page.
<b>page_noizetier_repertorier</b>	Renvoie une liste de pages éventuellement filtrées sur certains champs. La description renvoyée peut-être un champ, une liste de champ ou tous les champs. Les données sont renvoyées brutes sous forme d'un tableau indexé par l'identifiant de la page ou par un entier de 0 à n.

L'API fonctionnelle est complétée par une balise **#PAGE\_NOIZETIER\_INFOS** permettant de fournir toutes les informations utiles sur une page donnée, explicite ou virtuelle. La signature de la balise est `#PAGE_NOIZETIER_INFOS{page[, information]}`.

En premier lieu, cette balise fournit un champ ou tous les champs descriptifs d'une page explicite ou d'une composition virtuelle à partir de l'espace de stockage dédié (table `spip_noizetier_pages`).

La balise peut aussi renvoyer d'autres informations calculées cette fois, à savoir :

- `est_modifiee` qui indique si la configuration du fichier YAML ou XML de la page a été modifiée ou pas ;
- `compteurs_type_noisette` qui donne le nombre de types de noisettes disponibles pour la page donnée en distinguant les types de noisettes communs à toutes les pages, les types de noisettes spécifiques à un type de page et les types de noisettes spécifiques à une composition ;
- `compteurs_noisette` qui donne le nombre de noisettes incluses dans chaque bloc non vide de la page.

## 3.2 L'API de gestion des blocs

L'API de gestion des blocs fournit une interface fonctionnelle aujourd'hui limitée à la liste des blocs autorisés à accueillir des noisettes et à la description d'un bloc donné.

### API BLOCS : INC/NOIZETIER\_BLOC.PHP

<b>bloc_z_lister_defaut</b>	Renvoie la liste par défaut des identifiants de bloc d'une page. Cette liste peut être modifiée via le pipeline <code>noizetier_blocs_defaut</code> , en particulier pour supprimer certains blocs pour l'ensemble des pages.
<b>bloc_z_lire</b>	Retourne la description complète ou une information particulière d'un bloc donné. La description complète du bloc est inscrite dans un fichier YAML nommé <code>bloc.yaml</code> et stocké dans le dossier du bloc concerné.

L'API fonctionnelle est complétée par une balise `#BLOCS_Z_INFOS` permettant de fournir toutes les informations utiles sur un bloc donné. La signature de la balise est `#BLOC_Z_INFOS{bloc[, information]}`.

Cette balise fournit un champ ou tous les champs descriptifs d'un bloc à partir de la lecture de son fichier YAML si il existe.

## 3.3 L'API de gestion des objets

La possibilité d'associer des noisettes à un objet précis est une fonctionnalité un peu particulière apparue dans la version 2 du noiZetier. Elle permet de personnaliser un type de page (celui du type d'objet comme article par exemple) non pas avec une composition explicite ou virtuelle mais en associant directement les noisettes au conteneur formé par l'objet, le type d'objet et le bloc.

Pour manipuler ces objets, le noiZetier propose une API fonctionnelle et des balises décrites ci-après.

## API OBJETS : INC/NOIZETIER\_OBJET.PHP

<b>objet_noizetier_compter_noisettes</b>	Renvoie, pour un objet donné, la liste des blocs ayant des noisettes ajoutées et le nombre de noisettes pour chacun des blocs. Le tableau de sortie est de la forme [bloc] = nombre de noisettes. Les blocs vides ne sont pas fournis.
<b>objet_noizetier_lire</b>	Renvoie la description complète ou uniquement une information précise pour un objet donné. Étant donné que chaque objet possède une liste spécifique de champs, seuls les champs communément associés à un objet (titre, logo) ainsi que le nombre de noisettes ajoutées et la liste des blocs autorisés (ceux de la page identifiée par le type d'objet) sont fournis.
<b>objet_noizetier_repertorier</b>	Lister les objets ayant des noisettes spécifiquement ajoutées. La description complète de l'objet - au sens de la fonction <code>objet_noizetier_lire()</code> - est renvoyée. La liste des objets peut être filtrée sur un ou plusieurs champs de l'objet.
<b>objet_noizetier_type_active</b>	Détermine si un type d'objet est activé dans la configuration du noiZetier. Si oui, ses objets peuvent recevoir des noisettes spécifiques.

Outre cette API fonctionnelle, le noiZetier fournit deux balises qui sont le reflet des fonctions `objet_noizetier_lire()` et `objet_noizetier_repertorier()`, à savoir :

- `#OBJET_NOIZETIER_INFOS{type_objet, id_objet[, information]}` ;
- et `#OBJET_NOIZETIER_LISTE`.

Pour la balise `#OBJET_NOIZETIER_INFOS`, outre chacun des champs de la description de l'objet, il est possible de récupérer l'information calculée `compteurs_noisette` qui donne le nombre de noisettes incluses dans chaque bloc en contenant.

### 3.4 Complément de l'API de gestion des conteneurs

Pour le noiZetier, l'utilisation du framework N-Core a permis de simplifier la manipulation des noisettes. L'une des difficultés majeures qui émerge de fait, est la gestion des conteneurs. Étant donné que le noiZetier manipule des pages explicites ou des compositions virtuelles mais aussi des objets précis, le besoin récurrent est, d'une part, de définir le conteneur à partir de ces éléments et, d'autre part, de retrouver ces éléments à partir d'un conteneur donné.

Il existe déjà les fonctions de service `conteneur_identifier()` et `conteneur_construire()` qui manipulent le conteneur sous sa forme tabulaire canonique avec les index squelette, objet et `id_objet`. Mais l'interface du noiZetier amène parfois à manipuler des concepts plus atomique comme le type de page, la composition et le bloc. Le noiZetier est donc parfois amené à utiliser des

tableaux plus complexes où les index relatifs au bloc, type de page et composition doivent être présents.

C'est l'objet du complément fonctionnel de l'API des conteneurs présenté ci-dessous qui s'appuie sur les fonctions de service des conteneurs.

#### API CONTENEURS : INC/NOIZETIER\_CONTENEUR.PHP

##### **conteneur\_noizetier\_composer**

Détermine l'identifiant du conteneur à partir des éléments d'une page, d'un objet ou d'une noisette conteneur.

Cette fonction est en fait une encapsulation de la fonction d'API `conteneur_identifier()`.

##### **conteneur\_noizetier\_decomposer**

Détermine à partir de l'identifiant du conteneur les éléments propres au noizetier, à savoir, la page (type, composition), l'objet (type, identifiant) ou la noisette conteneur concernée (type, identifiant). Pour une noisette conteneur, la fonction renvoie aussi la page ou l'objet ascendant de plus haut niveau.

Cette fonction est en fait une extension de la fonction d'API `conteneur_construire()`.

## 4. FONCTIONNEMENT DU NOIZETIER

### 4.1 L'interface privée de gestion des pages

#### 4.1.1 Charger les pages

La première fonctionnalité du noiZetier est de recenser, afficher et éditer les pages explicites disponibles et de créer, afficher et éditer des compositions virtuelles.

En premier lieu, le noiZetier charge les pages explicites disponibles, à savoir :

- Les pages Z possédant un fichier YAML de description ;
- Les pages Z possédant un fichier XML de description (déprécié mais toujours possible pour assurer une compatibilité ascendante) ;
- Les pages Z ne possédant ni l'un ni l'autre mais uniquement repérées par leur fichier HTML. Cette possibilité, activée par défaut, est débrayable en positionnant la constante `_NOIZETIER_LISTER_PAGES_SANS_XML` à `false`.

Ce chargement est effectué à l'activation du plugin, lors de chaque affichage de la page d'administration des plugins afin de prendre en compte d'éventuelles nouvelles pages et sur demande en utilisant le bouton de raccourci « Recharger les pages » présent dans la page de l'espace privé dédiée à l'affichage des pages du noiZetier.

#### 4.1.2 Afficher les pages

Les pages sont affichées dans trois listes distinctes :

- la liste des pages dont le type coïncide avec celui d'un type d'objet (comme article),
- la liste des pages dont le type ne coïncide pas avec celui d'un type d'objet (comme sommaire ou plan),
- la liste de toutes les pages qui est affichée par défaut.

Ces listes correspondent à des onglets d'une même page de l'espace privé `noizetier_pages`, distingués par le paramètre d'url `est_page_objet` qui correspond au champ homonyme de la structure d'une page. Pour afficher toutes les pages il suffit d'omettre le paramètre `est_page_objet`.

Dans chaque liste les compositions explicites ou virtuelles sont présentées sous la page origine (type de page) et sont légèrement indentées. Une composition virtuelle est repérée par un fond distinct.

**Pages liées à un type de contenu**

---

Pages non liées à un type de contenu

---

Contenus possédant une configuration de noisettes

---

Configuration du noiZetier

---

Recharger les pages








Recharger les types de noisette

Exporter la configuration


Importer une config.

## noiZetier

Sélectionnez la page dont vous souhaitez configurer les noisettes.

 <b>Page Article (<i>article</i>)</b> Page par défaut des objets article	 
 <b>Essai de composition virtuelle (<i>article-essai</i>)</b>	 
 <b>Forum (<i>article-forum</i>)</b> Page de présentation d'un forum	 
 <b>Formulaire recommander (<i>article-recommander</i>)</b> Page de présentation du formulaire de recommandation d'un article	 
 <b>Page Auteur (<i>auteur</i>)</b>	 
 <b>zgala:page_titre_document (<i>document</i>)</b>	 
 <b>Page Rubrique (<i>rubrique</i>)</b> Page par défaut des objets rubrique	 
<b>Forums (<i>rubrique-forums</i>)</b> Cette rubrique contient des Forums	 
<b>Rubrique + du BIJ (<i>rubrique-plus</i>)</b> Composition adaptée pour les rubriques + du BIJ.	 

### 4.1.3 Editer une page


 À partir des listes précitées, il est possible pour chaque page d'éditer certaines de ses caractéristiques via la page de l'espace privé `noizetier_page_edit` :

- pour les pages explicites seule la liste des blocs exclus est modifiable ;
- pour les compositions virtuelles, toutes les caractéristiques d'une page sont modifiables à l'exception du type qui a été choisi une fois pour toute et de la composition une fois que celle-ci a été définie à la création.

Cette fonction est assurée par le formulaire `editer_page` d'édition d'une page (page de l'espace privé `noizetier_page_edit`).

### 4.1.4 Créer une composition virtuelle

Une des fonctionnalités remarquables du noiZetier est la création de composition virtuelle, c'est-à-dire non associée à un fichier XML, YAML ou HTML portant le même nom. Deux actions sont possibles à partir des listes de pages :

-  ○ **créer une composition** à partir d'un type de page en choisissant un nom de composition non déjà attribué. La composition virtuelle est initialisée avec l'identifiant et la liste des blocs exclus du type de page. Il est également possible de choisir de copier les noisettes du type de page source.



- **dupliquer une composition existante**, explicite ou virtuelle, en choisissant un nouveau nom de composition non déjà attribué. La composition virtuelle est initialisée avec l'identifiant du type de page et avec la liste des blocs exclus de la composition source. Les noisettes de la composition source sont automatiquement copiées.

Les deux actions sont réalisées dans le formulaire `editer_page` d'édition d'une page (page de l'espace privé `noizetier_page_edit`).

#### 4.1.5 Activer les compositions sur un type d'objet



Afin d'éviter de se rendre dans le formulaire de configuration du plugin Compositions, la liste des pages permet **d'activer les compositions** pour un type d'objet donné (correspondant à un type de page).

## 4.2 L'interface privée de gestion des noisettes

### 4.2.1 Afficher les noisettes d'une page

La manipulation des noisettes d'une page Z donnée se fait dans la page de l'espace privé nommée `noizetier_page`. Cette page de l'espace privé propose un layout classique qui rappelle celui des objets SPIP avec des informations de base sur la page Z concernée, une liste des types de noisettes utilisables et un affichage central visualisant les noisettes déjà affectées.

Étant donné que les noisettes sont affectées in fine à un bloc Z d'une page, l'affichage central permet de distinguer les blocs et de visualiser clairement les noisettes qui y sont associées. Il est possible de passer simplement d'un bloc à un autre.

La liste des noisettes d'un bloc est arborescente, le niveau maximal d'imbrication étant paramétrable (illimité, par défaut). Les noisettes incluses dans une noisette conteneur sont visualisées avec une légère indentation pour montrer l'imbrication.


A un instant donnée, un seul bloc est affiché in extenso avec sa liste de noisettes. Les autres sont repérés par des onglets à leur nom avec le nombre de noisettes insérés entre parenthèses. Il est possible de passer d'un bloc à l'autre en cliquant sur l'onglet souhaité.





Pages non liées à un type de contenu > Page Sommaire

IDENTIFIANT :  
**SOMMAIRE**

Page autonome non liée à un type de contenu

 Supprimer toutes les noisettes

 Page Sommaire

 Modifier cette page





















Page d'accueil du site


**Blocs configurables**


Aside (1) Breadcrumb (1) **Contenu (10)** Extra (0) Footer (0) Nav (1)

**Contenu**

Bloc principal de chaque page

	Bloc conteneur Balise englobante : section		Modifier cette noisette Supprimer cette noisette
	Bloc de texte libre Texte 1		Déplacer vers le haut Déplacer vers le bas
	Bloc de texte libre Texte 2		Ajouter une noisette au conteneur Supprimer les noisettes du conteneur
	Bloc conteneur Balise englobante : div		
	Bloc de texte libre Texte 3		
	Bloc de texte libre Texte 4		
	Articles récents		
	Conditions météorologiques Weather Underground – Paris,fr		
	Prévisions météorologiques 24h APIXU – Londres,UK Jours de prévisions : 5 – Premier jour : 1.		
	Bloc de texte libre Texte 5		















 Supprimer les noisettes du bloc

 Ajouter une noisette au bloc

**Types de noisette disponibles**

Les types de noisettes pouvant être ajoutés aux blocs de la page sont listés ci-dessous.

Types de noisette communs à toutes les pages :

-  Articles récents
-  Bienvenue
-  Bloc de texte libre
-  Fil d'Ariane
-  Code Spip libre
-  Bloc conteneur
-  Documents joints
-  Commentaires récents
-  Menu
-  Conditions météorologiques
-  Prévisions météorologiques 24h
-  Barre de navigation principale
-  Portfolio d'images
-  Articles en vedette


#### 4.2.2 Ajouter une noisette

Plusieurs méthodes sont à disposition de l'utilisateur pour ajouter une noisette au bloc en cours d'affichage :


- un bouton « **Ajouter une noisette** » est visible en bas du bloc en cours d’affichage et renvoie vers un formulaire permettant de choisir la ou les noisettes à ajouter en fin du bloc concerné ;
- une entrée du menu contextuel d’une noisette conteneur « **Ajouter une noisette au conteneur** » renvoie vers un formulaire permettant de choisir la ou les types de noisette et permet d’insérer une ou plusieurs noisettes en fin du conteneur concerné ;
- un mécanisme de **drag’n drop** d’un item de la liste des types de noisette disponibles à un endroit précis dans l’arborescence des noisettes du bloc en cours d’affichage permet d’ajouter cette noisette exactement à l’endroit désigné.

Si seule une noisette est ajoutée par la méthode 1 ou 2, alors le traitement renvoie vers le formulaire d’édition de la noisette, `editer_noisette`, pour finaliser son paramétrage. Si plusieurs noisettes sont ajoutées en même temps ou si le mécanisme de drag’n drop est utilisé, alors la ou les noisettes sont ajoutées avec les paramètres par défaut. Il convient ensuite d’éditer les noisettes pour affiner le paramétrage (paramètres d’affichage, encapsulation et styles).

### 4.2.3 Modifier le paramétrage d’une noisette


 A tout moment il est possible d’éditer le paramétrage d’une noisette déjà insérée dans un bloc. Cela se fait via l’icône crayon en regard de chaque noisette qui ouvre le formulaire d’édition dans une popup. Cette fonction est assurée par le formulaire `editer_noisette`.

### 4.2.4 Supprimer une noisette

 Le menu contextuel de chaque noisette insérée dans le bloc affiché possède une entrée « **Supprimer cette noisette** » qui lance la suppression de la noisette concernée sans préalable. Il est donc essentiel de n’activer cette option qu’avec précaution. Si la noisette est un conteneur toute l’arborescence est supprimée


### 4.2.5 Déplacer une noisette

Pour déplacer une noisette au sein d’un même conteneur ou d’un même bloc il existe deux méthodes :

- un mécanisme de **drag’n drop** d’une noisette d’un bloc donné à un autre endroit du même bloc. La position de destination est visualisée par un rectangle coloré tant que la noisette n’a pas été lâchée ;
-  ○ et deux entrées du menu contextuel d’une noisette, « **Déplacer vers le haut** » ou « **Déplacer vers le bas** », qui déplacent d’un rang vers le haut ou le bas la noisette au sein de son conteneur. Les déplacements sont circulaires au sein du conteneur.

Si la noisette déplacée est une noisette conteneur, toute son arborescence est déplacée.

### 4.2.6 Dupliquer une noisette au sein de son conteneur

 L’entrée du menu contextuel d’une noisette « **Dupliquer cette noisette** » crée une copie de la noisette concernée à la position suivante dans son conteneur en copiant également l’ensemble de ses paramètres d’affichage, d’encapsulation et de styles.

Si la noisette dupliquée est une noisette conteneur, toute son arborescence est dupliquée.

#### 4.2.7 Propager une noisette dans d'autres pages



L'entrée du menu contextuel d'une noisette « **Copier dans le même bloc d'autres pages** » permet de copier une noisette donnée (de profondeur nulle) dans le même bloc d'autres pages. L'action renvoie sur un formulaire où il est possible de choisir les pages destination et aussi si l'on veut copier le paramétrage de la noisette source (paramètres d'affichage, encapsulation et styles). La noisette est copiée en fin du bloc destination.

Si la noisette copiée est une noisette conteneur, toute son arborescence est aussi copiée.

#### 4.2.8 Vider un bloc ou une page

Un premier bouton situé au bas du bloc affiché et nommé « Supprimer les noisettes du bloc » permet de vider complètement l'ensemble des noisettes du bloc.

De même, un bouton situé dans la boîte d'informations de la page en cours de configuration et nommé « Supprimer toutes les noisettes » permet de vider entièrement les noisettes de tous les blocs de la page.

Aucun avertissement préalable n'est envoyé avant le vidage, il faut donc utiliser ces actions avec précaution.

### 4.3 L'interface privée de gestion des contenus

Etant donné qu'il est possible d'affecter des noisettes à un contenu précis, la page privée de ce contenu permet d'identifier si des noisettes sont effectivement affectées et fournit un lien vers la page idoine du noiZetier. Cette information est incluse dans le corps du contenu via le pipeline `affiche_milieu`.



La page de configuration des noisettes d'un contenu est similaire à celle d'une page explicite ou virtuelle.

### 4.4 La configuration du noiZetier

#### 4.4.1 Configurer le noiZetier

Le plugin noiZetier possède une configuration (formulaire `configurer_noizetier`) limitée aux variables suivantes :

## META : NOIZETIER

<b>encapsulation_noisette</b>	Détermine si, par défaut, il faut inclure chaque noisette dans une capsule (on) ou pas (chaîne vide).	on
<b>ajax_noisette</b>	Détermine si, par défaut, il faut inclure les noisettes en Ajax (on) ou pas (chaîne vide).	on
<b>inclusion_dynamique_noisette</b>	Détermine si, par défaut, il faut inclure les noisettes en Ajax (on) ou pas (chaîne vide).	Chaîne vide
<b>profondeur_max</b>	Définit la valeur maximale de la profondeur d'imbrication des noisettes dans un bloc de page. Il est possible de ne positionner aucune limite.	Chaîne vide (pas de limite)
<b>objets_noisettes</b>	Liste des objets SPIP pouvant accepter une personnalisation par contenu. Cette variable est un tableau de tables SPIP (par exemple, spip_articles).	Tableau vide

La configuration est intégrée au schéma de données du plugin et gérée conjointement avec les tables de la base de données. Les valeurs par défaut (colonne de droite du tableau précédent) sont affectées à la première activation du plugin.

Outre le lien disponible dans la liste des plugins de la page `admin_plugin`, la page d'accueil du noiZetier, `noizetier_pages`, propose un raccourci « **Configurer le plugin noiZetier** » pour se rendre dans la configuration du plugin.

### 4.4.2 Exporter ou importer des données

Le noiZetier peut exporter ou importer sa configuration et ses données de production en utilisant les services du plugin IEConfig. Le noiZetier ne nécessite pas IEConfig mais si celui-ci est activé la page d'accueil du noiZetier propose deux raccourcis supplémentaires en dessous du raccourci « Configurer le plugin noiZetier » qui sont : « **Exporter des données** » et « **Importer des données** ».

Le bouton « Exporter les données » renvoie vers le formulaire d'exportation compilé par IEConfig à partir des saisies fournies par les plugins utilisant la fonctionnalité au travers du pipeline `ieconfig`. Pour le noiZetier, il est uniquement demandé si les données du plugin doivent être ou pas incluses dans l'export sachant que IEConfig permet de sauvegarder dans un fichier les données de tous les plugins concernés.

L'export du noiZetier contient donc systématiquement :

- les variables de configuration décrites au paragraphe 4.4.1 ;
- le paramétrage des blocs exclus éventuellement des pages du noiZetier ;

- la description complète des compositions virtuelles créées par les utilisateurs ;
- et la description complète des noisettes affectées aux pages du noiZetier.

Le fichier d'export contient également pour le noiZetier, la version du plugin et celle du schéma de données de façon à gérer une incompatibilité due à une évolution du plugin.

Le bouton « Importer les données » renvoie vers le formulaire d'importation compilé par IEConfig à partir des saisies fournies par les plugins utilisant la fonctionnalité au travers du pipeline `ieconfig` et du contenu du fichier lui-même. Si le fichier d'import choisi contient des données du noiZetier, un formulaire est proposé pour choisir les données à importer et le mode d'importation de ces données. Les choix proposés dépendent du contenu du fichier (absence de certaines données, présence de doublons avec le contenu du site...).

La partie du formulaire concernant le noiZetier est présenté ci-dessous.

noiZetier

Le fichier à importer a été construit avec le noiZetier en version 3.0.63, schéma de données 0.6.0.

La configuration du plugin

La version 3.0.63 du plugin noiZetier actif sur ce site possède le même schéma 0.6.0 que celui du fichier d'import.

☐ Remplacer la configuration actuelle du noiZetier par celle du fichier d'import

Les blocs exclus des pages explicites

Il existe des pages et compositions explicites communes entre le site et le fichier d'import.

☐ Remplacer les blocs exclus des pages explicites du site par ceux du fichier d'import

Les compositions virtuelles

Il existe des compositions virtuelles dans le site et dans le fichier d'import.

☐ Ajouter les compositions virtuelles du fichier d'import. Les compositions virtuelles disponibles sur le site ne seront pas modifiées.
☐ Remplacer les compositions virtuelles disponibles sur le site par celles du fichier d'import
☐ Ajouter les compositions virtuelles du fichier d'import et remplacer les compositions virtuelles du site aussi disponibles dans le fichier d'import.

Les noisettes

Il existe des pages ou objets communs dans le site et dans le fichier d'import.

☐ Remplacer, pour les pages ou objets concernés, les noisettes actuellement configurées pour le site par les noisettes du fichier d'import
☐ Ajouter les noisettes du fichier d'import dans les pages ou objets concernés. Les noisettes actuellement configurées sur le site ne seront pas modifiées

## 4.5 L'interface publique de gestion des noisettes

## 5. DONNEES DU NOIZETIER

### 5.1 La structure des données

#### 5.1.1 Les pages

La description d'une page du noiZetier est structurée dans un tableau associatif dont tous les champs possibles sont initialisés.

DESCRIPTION D'UNE PAGE	
<i>Données de base (issues du fichier YAML, XML, HTML ou saisies)</i>	
<b>page</b>	Identifiant de la page. Correspond au nom du fichier YAML, XML ou HTML sans extension.
<b>type</b>	Identifiant du type de page. Pour une composition correspond à la chaîne qui précède le tiret dans l'identifiant de la page, sinon coïncide avec l'identifiant de la page.
<b>composition</b>	Identifiant de la composition correspondant au suffixe qui suit le tiret dans l'identifiant de la page si il existe ou à la chaîne vide sinon.
<b>nom</b>	Titre de la page sous forme textuelle ou d'un item de langue. Par défaut coïncide avec l'identifiant de la page.
<b>description</b>	Texte ou item de langue décrivant le rôle de la page. Peut-être vide.
<b>icon</b>	Nom du fichier d'icône représentant la page (sans chemin). Par défaut, prend la valeur <code>page-24.png</code> et <code>page-noxml-24.png</code> si la page explicite ne possède pas de fichier XML ou YAML.
<b>neccesite</b>	Liste des plugins – préfixes – nécessairement actifs pour utiliser la page. Ce champ est un tableau, éventuellement vide, de format « [] = préfixe ».
<b>branche</b>	Pour une composition sur un objet possédant des branches (comme l'objet rubrique), liste les compositions applicables sur les types d'objets pouvant être liés aux branches de l'objet parent. Le tableau est présenté sous la forme [type_objet] = composition.
<i>Données complémentaires</i>	
<b>blocs_exclus</b>	Liste des identifiants de blocs non autorisés à accueillir des noisettes. Cette liste est, d'une part, calculée à partir des blocs autorisés dans le fichier XML si ils existent (fonctionnalité dépréciée à éviter) et d'autre part, mis à jour manuellement via le formulaire d'édition d'une page.
<b>est_active</b>	Indicateur précisant si les plugins nécessités par la page sont tous activés ou pas. Prend les valeurs « oui » (par défaut) ou « non ». Si aucun plugin n'est nécessité, l'indicateur vaut toujours « oui ».

<b>est_virtuelle</b>	Prend la valeur « oui » pour une composition virtuelle et « non » pour une page explicite c'est-à-dire possédant un fichier YAML, XML ou HTML.
<b>est_page_objet</b>	Prend la valeur « oui » pour une page de type objet et « non » sinon. Ce champ est calculé à partir du champ type.
<b>signature</b>	md5 du fichier YAML ou XML calculé lors de son chargement. Si la page ne possède pas de fichier YAML ou XML le md5 est fixé à une valeur arbitraire qui ne change jamais.

Suivant que la page est explicite ou virtuelle, la plupart des champs ou une partie seulement est modifiable. Une composition virtuelle est presque entièrement modifiable alors qu'une page explicite n'a que la liste des blocs exclus de modifiable.

### 5.1.2 Les blocs

La description d'un bloc est structurée dans un tableau associatif dont tous les champs possibles sont initialisés. Le tableau est indexé par l'identifiant du bloc, c'est-à-dire le nom du dossier, ce qui explique que cet identifiant ne soit pas présent dans la structure.

DESCRIPTION D'UN BLOC	
<i>Données issues du fichier YAML ou HTML</i>	
<b>nom</b>	Titre du bloc sous forme textuelle ou d'un item de langue. Par défaut coïncide avec l'identifiant du bloc.
<b>description</b>	Texte ou item de langue décrivant le rôle du bloc. Peut-être vide.
<b>icon</b>	Nom du fichier d'icône représentant la page (sans chemin). Par défaut, prend la valeur <code>bloc-24.png</code> .

Les données relatives aux blocs proviennent soit du fichier YAML si il existe soit uniquement du nom du dossier. Cette structure n'est jamais conservée dans un espace de stockage à accès rapide mais est reconstruite à chaque requête de lecture.

### 5.1.3 Les types de noisettes

Les données des types de noisette sont constituées de la structure de base issue des fichiers YAML et fournie par N-Core et de deux champs supplémentaires nécessités pour le traitement du noiZetier, à savoir, le type de page et la composition éventuelle. Ces données sont extraites de l'identifiant du type de noisette.

## DESCRIPTION D'UN TYPE DE NOISETTE

### Données complémentaires du noiZetier

<b>type</b>	Identifiant du type de page. Pour une composition correspond à la chaîne qui précède le tiret dans l'identifiant de la page, sinon coïncide avec l'identifiant de la page.
<b>composition</b>	Identifiant de la composition ou vide sinon.

#### 5.1.4 Les noisettes

Pour ces traitements le noiZetier a besoin d'information plus atomique sur le conteneur (le bloc Z) qui accueille des noisettes. L'identifiant tabulaire ou chaîne n'est pas suffisant et pour les noisettes imbriquées il est nécessaire de toujours caractériser le conteneur bloc de plus haut niveau.

## DESCRIPTION D'UNE NOISETTE

### Données complémentaires du noiZetier

<b>type</b>	Identifiant du type de page. Pour une composition correspond à la chaîne qui précède le tiret dans l'identifiant de la page, sinon coïncide avec l'identifiant de la page.
<b>composition</b>	Identifiant de la composition ou vide sinon.
<b>bloc</b>	Identifiant du bloc Z qui correspond au nom du dossier.
<b>objet</b>	Type d'objet comme « article » et « rubrique » ou vide sinon.
<b>id_objet</b>	Identifiant de l'objet précis recevant la noisette ou 0 sinon.

## 5.2 Les espaces de stockage du noiZetier

Le noiZetier utilise la base de données SPIP comme espace de stockage privilégié. Ainsi, les pages explicites et les compositions virtuelles sont stockées dans une table SPIP nommée `spip_noizetier_pages`.

Les blocs n'ont pas d'espace de stockage, le noiZetier, quand il en a besoin, lit le fichier YAML correspondant ou si il n'existe pas récupère uniquement l'identifiant du bloc.

Le noiZetier n'utilise pas les espaces de stockage de N-Core pour les types de noisette et les noisettes. Il définit ses propres espaces, à savoir, la table `spip_types_noisettes` pour les types de noisette et la table `spip_noisettes` pour l'affectation des noisettes. La structure des tables est composée des champs imposés par N-Core et de champs complémentaires nécessaires au plugin noiZetier (voir les paragraphes 5.1.3 et 5.1.4).



## 6. MISE EN OEUVRE DES SERVICES N-CORE

La mise en œuvre du framework N-Core par le noiZetier consiste principalement à coder les fonctions de service nécessaires dans le fichier `ncore/noizetier.php`. Les paragraphes suivant donnent ainsi un éclairage sur la manière dont un plugin utilisateur doit réaliser les services requis par N-Core.

Dans la suite, les services sont désignés sans leur préfixe (« `noizetier_` »).

### 6.1 La gestion des types de noisette

Etant donné que le noiZetier définit son propre espace de stockage pour les types de noisette, il lui faut fournir la plupart des services requis par N-Core pour manipuler les types de noisette.

#### 6.1.1 Le service `type_noisette_completer()`

N-Core donne la possibilité au plugin appelant de compléter la description de base d'un type de noisette issue du fichier YAML.

Le noiZetier a besoin du type de page et de la composition éventuellement inscrits dans l'identifiant du type de noisette : `[<type_page>-]<composition>-nom_type_noisette`. Il extrait donc « `type_page` » et « `composition` » (éventuellement vides), les rajoute à la description fournie en entrée et renvoie la description modifiée en retour de la fonction.

```
function noizetier_type_noisette_completer($plugin, $description) {
    // - type_page-type_noisette si le type de noisette est dédié uniquement à une page
    // - type_page-composition-type_noisette si le type de noisette est dédié uniquement à une
    // composition
    // - type_noisette sinon
    $description['type'] = '';
    $description['composition'] = '';
    $identifiants = explode('-', $description['type_noisette']);
    if (isset($identifiants[1])) {
        $description['type'] = $identifiants[0];
    }
    if (isset($identifiants[2])) {
        $description['composition'] = $identifiants[1];
    }
    return $description;
}
```

#### 6.1.2 Le service `type_noisette_stocker()`

Pour optimiser les traitements N-Core fournit au plugin appelant les descriptions des types de noisette en les rangeant par traitement : ceux à ajouter, ceux à supprimer et ceux modifier. En outre, N-Core fournit un indicateur pour forcer le rechargement complet et dans ce cas le plugin appelant doit supprimer tous les types de noisette stockés et ajouter les types de noisette fournis (les autres listes sont vides).

Le noiZetier doit donc prendre en compte ces deux comportements avec et sans forçage et réalise les traitements demandés via l'API SQL, les tableaux étant formatés pour faciliter les appels.

```
function noizetier_type_noisette_stocker($plugin, $types_noisettes, $recharger) {

    $retour = true;

    // Mise à jour de la table des noisettes 'spip types noisettes'.
    $from = 'spip types noisettes';

    // -- Suppression des types de noisette obsolètes ou de tous les types de noisette d'un
    coup si on est en mode
    //   rechargement forcé.
    if (sql_preferer_transaction()) {
        sql_demarrer_transaction();
    }
    $where = array('plugin=' . sql_quote($plugin));
    if ($recharger) {
        sql_delete($from, $where);
    } elseif (!empty($types_noisettes['a_effacer'])) {
        $where[] = sql_in('type_noisette', $types_noisettes['a_effacer']);
        sql_delete($from, $where);
    }
    // -- Update des types de noisettes modifiés
    if (!empty($types_noisettes['a_changer'])) {
        sql_replace_multi($from, $types_noisettes['a_changer']);
    }
    // -- Insertion des nouveaux types de noisette
    if (!empty($types_noisettes['a_ajouter'])) {
        sql_insertq_multi($from, $types_noisettes['a_ajouter']);
    }
    if (sql_preferer_transaction()) {
        sql_terminer_transaction();
    }

    return $retour;
}
```

### 6.1.3 Le service `type_noisette_decrire()`

Ce service lit les données d'un type de noisette et retourne in extenso la description complète stockée. Pour le noiZetier cela revient à un appel à l'API SQL `sql_fetsel` avec l'ensemble des champs de la table à l'exception du timestamp `maj`.

```
function noizetier_type_noisette_decrire($plugin, $type_noisette) {

    // Table des types de noisette.
    $from = 'spip types noisettes';

    // Chargement de la noisette en base de données sauf le timestamp 'maj'. Les données
    // sont renvoyées brutes sans traitement sur les textes ni les tableaux sérialisés.
    $trouver_table = charger_fonction('trouver_table', 'base');
    $table = $trouver_table($from);
    $select = array_diff(array_keys($table['field']), array('maj'));

    $where = array('plugin=' . sql_quote($plugin), 'type_noisette=' .
    sql_quote($type_noisette));
    $description = sql_fetsel($select, $from, $where);

    return $description;
}
```

### 6.1.4 Le service `type_noisette_lister()`

Ce service permet de récupérer la description complète ou juste un champ de l'ensemble des types de noisettes disponibles. Le service renvoie donc un tableau indexé par identifiant de type de

noisette et contenant soit une valeur unique, celle de l'information demandée, soit un tableau associatif de l'ensemble des champs. Dans le cas où l'information demandée est invalide le service renvoie un tableau vide : cela requiert du plugin appelant de faire une vérification sur la liste des champs autorisés.

Dans le cas du noiZetier, la table `spip_types_noisettes` contient un timestamp en plus des données standard. Il convient donc de retirer ce champ `maj` de la liste des champs lus. La lecture se fait simplement au travers d'un `sql_allfetsel()` de l'API SQL.

```
function noizetier_type_noisette_lister($plugin, $information = '') {
    // Initialiser le tableau de sortie en cas d'erreur
    $types_noisettes = array();

    $from = 'spip_types_noisettes';
    $trouver_table = charger_fonction('trouver_table', 'base');
    $table = $trouver_table($from);
    $champs = array_keys($table['field']);
    if ($information) {
        // Si une information précise est demandée on vérifie sa validité
        $information_valide = in_array($information, $champs);
        $select = array('type_noisette', $information);
    } else {
        // Tous les champs sauf le timestamp 'maj' sont renvoyés.
        $select = array_diff($champs, array('maj'));
    }
    $where = array('plugin=' . sql_quote($plugin));

    if ((!$information or ($information and $information_valide))
        and ($types_noisettes = sql_allfetsel($select, $from, $where))) {
        if ($information) {
            $types_noisettes = array_column($types_noisettes, $information, 'type_noisette');
        } else {
            $types_noisettes = array_column($types_noisettes, null, 'type_noisette');
        }
    }

    return $types_noisettes;
}
```

### 6.1.5 Le service `type_noisette_initialiser_ajax()`

Pour construire son cache des indicateurs Ajax, N-Core, quand un type de noisette est positionné à défaut dans son YAML fait appel à l'indicateur défini par défaut. N-Core initialise cet indicateur à oui (toutes les noisettes sont incluses en Ajax par défaut) mais permet à chaque plugin utilisateur de définir son propre défaut.

Dans le cas du noiZetier, cette valeur par défaut est une variable de configuration du plugin (voir le paragraphe 4.4.1). Le service du noiZetier se contente donc de lire et renvoyer la valeur courante de cette variable de configuration.

```
function noizetier_type_noisette_initialiser_ajax($plugin) {
    // La valeur Ajax par défaut est inscrite dans la configuration du plugin.
    include_spip('inc/config');
    $default_ajax = lire_config("${plugin}/ajax_noisette");

    return $default_ajax;
}
```

### 6.1.6 Le service `type_noisette_initialiser_inclusion()`

Pour construire son cache des indicateurs d'inclusion dynamique, N-Core, quand un type de noisette est positionné à défaut dans son YAML fait appel à l'indicateur défini par défaut. N-Core initialise cet indicateur à statique (toutes les noisettes sont incluses en statique par défaut) mais permet à chaque plugin utilisateur de définir son propre défaut.

Dans le cas du noiZetier, cette valeur par défaut est une variable de configuration du plugin (voir le paragraphe 4.4.1). Le service du noiZetier se contente donc de lire et renvoyer la valeur courante de cette variable de configuration.

```
function noizetier_type_noisette_initialiser_inclusion($plugin) {
    // La valeur d'inclusion dynamique par défaut est inscrite dans la configuration du plugin.
    include spip('inc/config');
    $default_inclusion = lire_config("${plugin}/inclusion_dynamique_noisette");

    return $default_inclusion;
}
```

### 6.1.7 Le service `type_noisette_initialiser_dossier()`

Ce service définit la localisation relative des types de noisettes. Par défaut, N-Core définit cette valeur à `noisettes/`.

Le noiZetier utilise aussi ce dossier relatif comme localisation de ses types de noisette. **Il ne fournit donc pas de service propre.**

### 6.1.8 Conclusion

Les fonctions de services nécessaires pour faire fonctionner l'espace de stockage des types de noisette propre au noiZetier ont un code limité et plutôt trivial.

Le choix du stockage en base de données des types de noisette procède plus d'une logique de regrouper l'ensemble des concepts du noiZetier en un même espace simple d'utilisation que d'une optimisation des performances. Néanmoins, si l'on voulait à terme changer l'espace de stockage il suffirait de recoder uniquement ces fonctions de service contenues dans le fichier `ncore/noizetier.php` ou de les supprimer pour utiliser directement le stockage proposé par N-Core (fichiers cache sécurisés) : **le code du noiZetier ne subirait aucune autre modification.**

## 6.2 La gestion des conteneurs

Le conteneur, à l'exception des noisettes conteneur, est toujours un concept spécifique au plugin utilisateur. Dans le cas du noiZetier on a vu qu'il correspond toujours à un bloc Z d'une page ou d'un contenu prévus. L'identification d'un tel conteneur, quel que soit sa forme, est donc un service que le plugin utilisateur devra toujours fournir.

Le noiZetier manipule aussi des informations connexes à celle de la forme canonique des conteneurs. C'est pourquoi il complète l'API conteneur de N-Core avec des fonctions spécifiques (voir le

paragraphe 3.4). Ces fonctions complémentaires utilisent aussi les API de N-Core, donc par translation, les services de N-Core.

### 6.2.1 Le service `conteneur_identifier()`

C'est le service de base qui calcule l'identifiant unique d'un conteneur spécifique du noizetier sous la forme d'une chaîne de caractères. C'est cette fonction qui produit les identifiants du type `content/article`, `nav/article-forum` et `aside/article|article|12` à partir du conteneur fourni sous forme tabulaire. Seuls les index de la forme canonique sont utilisés, à savoir, `squelette`, `objet` et `id_objet` si ils existent.

Le tableau conteneur fourni en argument de la fonction est réputé vérifié.

```
function noizetier_conteneur_identifier($plugin, $conteneur) {

    // On initialise l'identifiant à vide.
    $id_conteneur = '';

    // Les noisettes conteneur ont été identifiées par N-Core, inutile donc de s'en préoccuper.
    if ($conteneur) {
        if (!empty($conteneur['squelette'])) {
            $id_conteneur .= $conteneur['squelette'];
        }
        if (!empty($conteneur['objet']) and !empty($conteneur['id_objet']) and
intval($conteneur['id_objet'])) {
            $id_conteneur .= ($id_conteneur ? '|' : '') .
"{ $conteneur['objet'] | { $conteneur['id_objet'] }";
        }
    }

    return $id_conteneur;
}
```

### 6.2.2 Le service `conteneur_construire()`

C'est le service inverse du service `conteneur_identifier()` qui calcule l'identifiant unique d'un conteneur spécifique du noizetier sous sa forme canonique tabulaire à partir de l'identifiant chaîne. Seuls les index canoniques sont inclus dans le tableau de sortie.

```
function noizetier_conteneur_construire($plugin, $id_conteneur) {

    // N-Core s'occupe des noisettes conteneur
    $conteneur = array();
    if ($id_conteneur) {
        $elements = explode('|', $id_conteneur);
        if (count($elements) == 1) {
            // C'est une page ou une composition : seul l'index squelette est positionné.
            $conteneur['squelette'] = $id_conteneur;
        } elseif (count($elements) == 3) {
            // C'est un objet
            $conteneur['objet'] = $elements[1];
            $conteneur['id_objet'] = $elements[2];
            $conteneur['squelette'] = $elements[0];
        }
    }

    return $conteneur;
}
```

### 6.2.3 Le service `conteneur_verifier()`

Ce service vérifie les index d'un conteneur spécifique du noiZetier fourni sous forme de tableau et produit en sortie un conteneur au format tabulaire canonique comme le service `conteneur_construire()`. Ce service est particulièrement utile lors de la mise à jour de la description d'une noisette dans laquelle on stocke le conteneur sous sa forme tabulaire canonique.

Le noiZetier filtre les index et vérifie donc la présence obligatoire de l'index `squelette` et la présence facultative des index `objet` et `id_objet`.

```
function noizetier_conteneur_verifier($plugin, $conteneur) {

    // Liste des index autorisés.
    static $index_conteneur = array('squelette', 'objet', 'id_objet');

    // On vérifie que les index autorisés sont les seuls retournés.
    $conteneur_verifie = array();
    if ($conteneur) {
        // L'index squelette doit toujours être présent.
        if ((isset($conteneur['squelette']) and $conteneur['squelette'])) {
            $conteneur = array_intersect_key($conteneur, array_flip($index_conteneur));
            if ((count($conteneur) == 1)
                or ((count($conteneur) == 3)
                    and isset($conteneur['objet'], $conteneur['id_objet'])
                    and $conteneur['objet']
                    and intval($conteneur['id_objet'])))) {
                // Le conteneur coïncide avec un squelette de bloc générique ou d'un objet donné.
                $conteneur_verifie = $conteneur;
            }
        }
    }

    return $conteneur_verifie;
}
```

### 6.2.4 Le service `conteneur_destocker()`

L'objectif de ce service est de vider un conteneur donné de ses noisettes. Le plugin utilisateur n'a pas à se préoccuper de l'imbrication des noisettes car c'est N-Core qui gère l'éventuelle récursivité due aux noisettes conteneur.

De fait, pour le noiZetier, la fonction est limitée à un `sql_delete()` avec la condition sur l'identifiant chaîne du conteneur à vider.

```
function noizetier_conteneur_destocker($plugin, $conteneur) {

    // Initialisation de la sortie.
    $retour = false;

    // Calcul de l'id du conteneur en fonction du mode d'appel de la fonction.
    if (is_array($conteneur)) {
        include_spip('inc/ncore_conteneur');
        $id_conteneur = conteneur_identifier($plugin, $conteneur);
    } else {
        $id_conteneur = $conteneur;
    }

    if ($id_conteneur) {
        // Suppression de toutes les noisettes du conteneur.
        $where = array('id_conteneur=' . sql_quote($id_conteneur), 'plugin=' .
    }
```

```
sql_quote($plugin));
    if (sql_delete('spip_noisettes', $where)) {
        $retour = true;
    }
}

return $retour;
}
```

### 6.2.5 Conclusion

Comme pour les types de noisette, les services associés aux conteneurs restent d'un codage aisé. Les **services d'identification des conteneurs spécifiques sont toujours à fournir par le plugin utilisateur** qu'il utilise ou pas les espaces de stockage de N-Core. Seul le service `conteneur_destocker()` dépend de l'espace de stockage utilisé par le plugin utilisateur.

Enfin, on constate comme expliqué dans la documentation de N-Core, que le service `conteneur_est_noisette()` n'est jamais à coder par un plugin utilisateur car il ne concerne que les conteneurs noisette.

## 6.3 La gestion des noisettes

Etant donné que le noiZetier définit son propre espace de stockage pour les noisettes, il lui faut fournir la plupart des services requis par N-Core pour manipuler les noisettes.

### 6.3.1 Le service `noisette_completer()`

N-Core donne la possibilité au plugin appelant de compléter la description de base d'une noisette issue principalement du type de noisette, du conteneur d'accueil et de la position dans le conteneur.

Le noiZetier a besoin de connaître plus finement les éléments du conteneur de plus haut niveau, à savoir, le bloc Z, y compris quand la noisette n'est pas directement incluse dans ce bloc mais imbriquée dans un conteneur noisette. Ces informations complémentaires sont extraites de l'identifiant du conteneur représentant le bloc Z.

Le noiZetier extrait donc le type de page, la composition éventuelle, le nom du bloc Z et les éventuels informations sur le contenu, les rajoute à la description fournie en entrée et renvoie la description modifiée en retour de la fonction. Pour extraire ces informations complémentaires il utilise la fonction d'API `conteneur_noizetier_decomposer()`.

```
function noizetier_noisette_completer($plugin, $description) {
    if (!empty($description['conteneur'])) {
        //
        $complement = array(
            'type' => '',
            'composition' => '',
            'objet' => '',
            'id objet' => 0,
            'bloc' => ''
        );

        $id_conteneur = $description['id_conteneur'];
        $conteneur_etendu = noizetier_conteneur_decomposer($id_conteneur);
```

```

if ($conteneur_etendu) {
    // On ajoute les index manquants avec leur valeur par défaut
    $conteneur_etendu = array_merge($complement, $conteneur_etendu);
    // On filtre les index inutiles comme squelette par exemple.
    $complement = array_intersect_key($conteneur_etendu, $complement);
}

// Ajout du complément à la description.
$description = array_merge($description, $complement);
}

return $description;
}

```

### 6.3.2 Le service `noisette_ranger()`

Le service permet de changer le rang d'une noisette au sein de son conteneur. Le fait que la position finale soit occupée ou pas n'est pas à gérer par le service. De fait, pour le noizetier, ce service est trivial.

```

function noizetier_noisette_ranger($plugin, $description, $rang_destination) {

    // Initialisation de la sortie.
    $retour = false;

    if (isset($description['id_noisette']) and ($id = intval($description['id_noisette'])) {
        $where = array('id_noisette=' . $id, 'plugin=' . sql_quote($plugin));
        $update = array('rang_noisette' => $rang_destination);
        if (sql_updateq('spip_noisettes', $update, $where)) {
            $retour = true;
        }
    }

    return $retour;
}

```

### 6.3.3 Le service `noisette_changer_conteneur()`

Ce service permet de transférer une noisette d'un conteneur source, à un rang donnée dans un conteneur destination. Le fait que la position finale soit occupée ou pas n'est pas à gérer par le service.

Le service fournit la description complète de la noisette dans sa position source et attend que le service modifie cette description pour refléter la nouvelle position dans l'espace de stockage et qu'il renvoie la description modifiée.

Les champs mis à jour sont les identifiants chaîne et tableau du conteneur d'accueil, le rang et la profondeur. Pour faciliter le codage du service, la profondeur de la noisette dans le conteneur destination est fournie en argument du service.

```

function noizetier_noisette_changer_conteneur($plugin, $description, $id_conteneur, $rang,
$profondeur) {

    // On rajoute la description à son emplacement destination en prenant soin de modifier les
    index id_conteneur,
    // conteneur et rang_noisette qui doivent représenter le conteneur destination.
    include_spip('inc/noisette_conteneur');
    $description['conteneur'] = serialize(conteneur_construire($plugin, $id_conteneur));
    $description['id_conteneur'] = $id_conteneur;
}

```



```

    $description['rang_noisette'] = $rang;
    $description['profondeur'] = $profondeur;

    // On met à jour l'objet en base
    // On sauvegarde l'id de la noisette et on le retire de la description pour éviter une
    erreur à l'update.
    $id_noisette = intval($description['id_noisette']);
    unset($description['id_noisette']);

    // Mise à jour de la noisette.
    $where = array('id_noisette=' . $id_noisette, 'plugin=' . sql_quote($plugin));
    sql_updateq('spip_noisettes', $description, $where);

    // On remet l'id de la noisette pour renvoyer la description complète
    $description['id_noisette'] = $id_noisette;

    return $description;
}

```

### 6.3.4 Le service noisette\_stocker()

Le service permet, d'une part, de stocker une nouvelle noisette dans l'espace de stockage du plugin utilisateur ou de mettre à jour la description d'une noisette déjà stockée. La description complète de la noisette à stocker ou à mettre à jour est fournie en argument.

Si l'identifiant de la noisette n'est pas fourni dans la description, le service considère que la noisette est à stocker pour la première fois. Si, par contre, la description de la noisette contient son identifiant, le service ne fait qu'une mise à jour dans l'espace de stockage. Dans tous les cas, le service renvoie l'identifiant de la noisette ou 0 si une erreur s'est produite.

```

function noizetier_noisette_stocker($plugin, $description) {

    // Mise à jour en base de données.
    if (empty($description['id_noisette'])) {
        // On s'assure que la description contient bien le plugin et alors on insère la nouvelle
        noisette.
        $id_noisette = 0;
        if (isset($description['plugin']) and ($description['plugin'] == $plugin)) {
            // Insertion de la nouvelle noisette.
            $id_noisette = sql_insertq('spip_noisettes', $description);
        }
    } else {
        // On sauvegarde l'id de la noisette et on le retire de la description pour éviter une
        erreur à l'update.
        $id_noisette = intval($description['id_noisette']);
        unset($description['id_noisette']);

        // Mise à jour de la noisette.
        $where = array('id_noisette=' . $id_noisette, 'plugin=' . sql_quote($plugin));
        if (!sql_updateq('spip_noisettes', $description, $where)) {
            $id_noisette = 0;
        }
    }

    return $id_noisette;
}

```

### 6.3.5 Le service noisette\_destocker()

Ce service ne fait que retirer de l'espace de stockage une noisette définie par sa description complète. Aucune vérification n'est à faire par le service autre que l'existence de la noisette dans

l'espace de stockage. Les impacts sur le conteneur et les autres noisettes ne sont pas à gérer dans ce service.

Dans le cas du noiZetier, ce service correspond donc à un simple `sql_delete()` sur l'identifiant de la noisette.

```
function noizetier_noisette_destocker($plugin, $description) {

    // Initialisation de la sortie.
    $retour = false;

    // Calcul de la clause where à partir de l'id du conteneur.
    if (isset($description['id_noisette'])) {
        $where = array('id_noisette=' . intval($description['id_noisette']), 'plugin=' .
sql_quote($plugin));

        // Suppression de la noisette.
        if (sql_delete('spip_noisettes', $where)) {
            $retour = true;
        }
    }

    return $retour;
}
```

### 6.3.6 Le service noisette\_decrire()

Ce service lit les données d'une noisette et retourne in extenso la description complète stockée. Pour le noiZetier cela revient à un appel à l'API SQL `sql_fetsel()` avec l'ensemble des champs de la table. La seule difficulté pour le service est de prendre en compte les deux modes d'identification d'une noisette, l'identifiant unique ou le couple (id\_conteneur, rang).

```
function noizetier_noisette_decrire($plugin, $noisette) {

    $description = array();

    $where = array();
    if (!is_array($noisette)) {
        // L'identifiant est l'id unique de la noisette. Il faut donc parcourir le tableau pour
trouver la
        // noisette désirée
        // => C'est la méthode optimale pour le stockage noiZetier.
        $where[] = 'id_noisette=' . intval($noisette);
    } elseif (isset($noisette['id_conteneur']) and isset($noisette['rang_noisette'])) {
        // L'identifiant est un tableau associatif fournissant l'id du conteneur et le rang.
        $where[] = 'id_conteneur=' . sql_quote($noisette['id_conteneur']);
        $where[] = 'rang_noisette=' . intval($noisette['rang_noisette']);
    }

    if ($where) {
        $where[] = 'plugin=' . sql_quote($plugin);
        $description = sql_fetsel('*', 'spip_noisettes', $where);
    }

    return $description;
}
```

### 6.3.7 Le service `noisette_list()`

Ce service permet de récupérer la description complète ou juste un champ de l'ensemble des noisettes d'un conteneur ou de l'ensemble des noisettes disponibles. La complexité de ce service provient de sa flexibilité tant sur le filtrage des noisettes que sur la présentation du tableau de sortie.

Si un conteneur est spécifié en argument du service il peut être identifié soit par son identifiant chaîne, soit par son identifiant canonique tabulaire. Le service doit donc prendre en compte ces deux modes.

Ensuite, le service peut renvoyer une information précise ou l'ensemble de la description.

Enfin, le service doit renvoyer un tableau indexé soit par identifiant de noisette (`id_noisette`), soit par rang (`rang_noisette`) ce qui peut nécessiter, si la liste n'est pas réduite à un conteneur donné, d'utiliser une double indexation par identifiant de conteneur (`id_conteneur`) et par rang.

Dans le cas du `noiZetier`, la lecture se fait simplement au travers d'un `sql_allfetsel()` de l'API SQL avec un classement des résultats par rang dans un conteneur ou par conteneur et par rang.

```
function noisetier_noisette_list($plugin, $conteneur = array(), $information = '', $cle = 'rang_noisette') {

    // Initialisation du tableau de sortie.
    $noisettes = array();

    // Construction du where et du order by en fonction du conteneur qui est soit un squelette,
    // soit un squelette d'un objet donné, soit vide (on veut toutes les noisettes du plugin).
    $where = array('plugin=' . sql_quote($plugin));
    if ($conteneur) {
        // On sélectionne le contenant par son identifiant qui est stocké dans la table.
        if (is_array($conteneur)) {
            include spip('inc/ncore_conteneur');
            $id_conteneur = conteneur_identifier($plugin, $conteneur);
        } else {
            $id_conteneur = $conteneur;
        }
        $where[] = array('id_conteneur=' . sql_quote($id_conteneur));
        $order_by = array('rang_noisette');
    } else {
        // On veut toutes les noisettes, on ordonne par conteneur et par rang dans chaque
        // conteneur.
        $order_by = array('id_conteneur', 'rang_noisette');
    }

    // Construction du select en fonction des informations à retourner.
    $select = $information ? array_merge(array('id_conteneur', 'rang_noisette', 'id_noisette'),
    array($information)) : '*';

    if ($table_noisettes = sql_allfetsel($select, 'spip_noisettes', $where, '', $order_by)) {
        if ($cle == 'rang_noisette') {
            // On demande un rangement par rang.
            // Il faut tenir compte du fait que la liste est réduite à un conteneur ou pas.
            foreach ($table_noisettes as $_noisette) {
                if ($conteneur) {
                    $noisettes[$_noisette['rang_noisette']] = $information
                        ? array($information => $_noisette[$information])
                        : $_noisette;
                } else {
                    $noisettes[$_noisette['id_conteneur']][$_noisette['rang_noisette']] =
                    $information
                }
            }
        }
    }
}
```

```

        ? array($information => $_noisette[$information])
        : $_noisette;
    }
} else {
    // On demande un rangement par id noisette
    $noisettes = $information
        ? array_column($table_noisettes, $information, 'id_noisette')
        : array_column($table_noisettes, null, 'id_noisette');
}
}

return $noisettes;
}

```

### 6.3.8 Le service `noisette_initialiser_encapsulation()`

Pour déterminer le mode d'encapsulation d'une noisette, N-Core, quand le champ `encapsulation` d'une noisette est positionné à défaut, fait appel à l'indicateur défini par défaut. N-Core initialise cet indicateur à oui (toutes les noisettes sont encapsulées par défaut) mais permet à chaque plugin utilisateur de définir son propre défaut.

Dans le cas du `noiZetier`, cette valeur par défaut est une variable de configuration du plugin (voir le paragraphe 4.4.1). Le service du `noiZetier` se contente donc de lire la valeur courante de cette variable de configuration et de retourner un booléen.

```

function noizetier_noisette_initialiser_encapsulation($plugin) {
    // La capsule par défaut est inscrite dans la configuration du plugin.
    include_spip('inc/config');
    $default_capsule = lire_config("${plugin}/encapsulation noisette");

    return $default_capsule;
}

```

### 6.3.9 Conclusion

Comme pour les types de noisette, les services associés aux conteneurs restent d'un codage aisé. Les **services d'identification des conteneurs spécifiques sont toujours à fournir par le plugin utilisateur** qu'il utilise ou pas les espaces de stockage de N-Core. Seul le service `conteneur_destocker()` dépend de l'espace de stockage utilisé par le plugin utilisateur. Enfin, on constate comme expliqué dans la documentation de N-Core, que le service `conteneur_est_noisette()` n'est jamais à coder par un plugin utilisateur car il ne concerne que les conteneurs noisette.

## 7. MISE EN OEUVRE DE L'API N-CORE

### 7.1 La gestion des types de noisette

#### 7.1.1 Charger les types de noisettes

Le noiZetier charge les types de noisettes disponibles en base de données en appelant la fonction `type_noisette_charger('noizetier')` dans le pipeline `affiche_milieu` de la page d'administration des plugins et manuellement lors du clic sur le bouton « Recharger les types de noisettes » présent dans la liste des pages du noiZetier dans l'espace privé.

#### 7.1.2 Lire un type de noisette

Pour lire la description complète d'un type de noisette ou d'une information donnée, le noiZetier fait appel à l'API `type_noisette_lire()`. Pour lire uniquement le champ paramètre, l'appel est de la forme :

```
type_noisette_lire('noizetier', _request('type_noisette'), 'parametres', false);
```

Pour lire toute la description il suffit de passer la chaîne vide en troisième argument.

#### 7.1.3 Répertoire les types de noisette

Pour établir la collection des types de noisette disponibles, le noiZetier peut faire appel à l'API `type_noisette_repertoire()` en précisant d'éventuels filtres sur les champs du type de noisette.

### 7.2 La gestion des conteneurs

#### 7.2.1 Identifier un conteneur

Pour le noiZetier, un conteneur est un bloc d'une page ou un bloc d'un objet SPIP donné (sans oublier la noisette conteneur gérée par N-Core). En conséquence, un conteneur est toujours matérialisé par un squelette, soit générique comme `content/article` ou `content/sommaire`, soit instancié sur un objet précis comme `content/article` de l'article d'id 12. Sa forme canonique est celle d'un tableau dont l'index unique « squelette » pour les pages est complété par « objet » et « id\_objet » pour les contenus.

Néanmoins, les API de N-Core autorisent aussi l'utilisation d'un autre identifiant pour le conteneur sous forme d'une chaîne unique et qui est stocké dans la structure d'une noisette.

Le calcul de cet identifiant est confié à l'API `conteneur_identifier()` qui fait appel à deux fonctions de service que le noiZetier doit fournir, `noizetier_conteneur_identifier()` et `noizetier_conteneur_verifier()`, dont le but est de vérifier que la structure tabulaire du conteneur est conforme en supprimant éventuellement les index ne faisant pas partie de la forme canonique.

Il en résulte qu'un conteneur de type squelette générique aura un identifiant de la forme `content/article` et qu'un conteneur de « type objet » aura un identifiant de la forme `content/article|article|12`.

On constate que le noiZetier ne se préoccupe pas du calcul de l'identifiant d'une noisette conteneur. En effet, c'est N-Core qui le calcule et fournit un identifiant de la forme `conteneur|noisette|8`.

En outre, étant donné que les API de N-Core permettent de spécifier indifféremment un conteneur par sa forme tabulaire ou chaîne, il est nécessaire de proposer une fonction inverse du calcul de l'identifiant. Cette fonction d'API est nommée `conteneur_construire()`.

De même que pour le service de calcul de l'identifiant chaîne, le noiZetier ne se préoccupe pas du cas d'une noisette conteneur qui est entièrement gérée par N-Core.

### 7.2.2 Tester un conteneur

Les traitements diffèrent souvent entre un conteneur noisette et un autre type de conteneur spécifique au plugin utilisateur. C'est pourquoi N-Core propose l'API `conteneur_est_noisette()` qui renvoie vrai si le conteneur est une noisette.

### 7.2.3 Vider un conteneur

Le noiZetier propose différentes options pour vider l'ensemble des noisettes d'un conteneur quel qu'il soit. Pour ce faire, le noiZetier utilise l'API de N-Core `conteneur_vider()` en passant comme argument l'identifiant du conteneur concerné.

### 7.2.4 Les besoins spécifiques du noiZetier

Comme décrit dans le paragraphe 3.4 le noiZetier est parfois amené à utiliser des tableaux plus complexes que celui du conteneur où les index bloc, type de page et composition doivent être présents en plus des index canoniques. Pour ce faire le noiZetier a défini deux fonctions d'API qui font appel aux API d'identification et de construction des conteneurs fournies par N-Core.

## 7.3 La gestion des noisettes

L'API de gestion des noisettes est une sorte de « CRUD étendu ». Elle permet d'ajouter, de dupliquer ou de retirer une noisette d'un conteneur, de déplacer une noisette au sein d'un conteneur ou dans un autre conteneur et de lire les caractéristiques d'une ou plusieurs noisettes.

### 7.3.1 Ajouter une noisette à un conteneur

Le noiZetier propose dans la page de configuration des noisettes d'une page ou d'un objet différents moyens d'ajouter une noisette (voir paragraphe 4.2.2). Néanmoins, dans tous les cas le noiZetier utilise l'API de gestion des noisettes de la façon suivante :

```
noisette_ajouter('noizetier', $type_noisette, $conteneur, $rang)
```

### 7.3.2 Supprimer une noisette d'un conteneur

Le noiZetier fournit pour chaque noisette insérée dans un bloc de page, un moyen pour retirer la noisette du conteneur auquel elle appartient. Le noiZetier appelle ainsi la fonction d'API `noisette_supprimer()` qui va gérer elle-même le cas où la noisette à supprimer est une noisette conteneur en supprimant les noisettes imbriquées de façon récursive.

### 7.3.3 Déplacer une noisette dans un conteneur

Le noiZetier permet de déplacer les noisettes d'un bloc d'une position à une autre dans le même conteneur ou dans un conteneur différent. Le noiZetier appelle donc la fonction d'API `noisette_deplacer()` à partir de l'action nommée `deplacer_noisette` ou du mécanisme de drag'n drop javascript.

### 7.3.4 Dupliquer une noisette dans un conteneur

Le noiZetier permet de déplacer les noisettes d'un bloc d'une position à une autre. Le noiZetier appelle donc la fonction d'API `noisette_deplacer()` à partir de l'action nommée `deplacer_noisette`.

## 7.4 La compilation des noisettes

Le noiZetier utilise la balise `#NOISETTE_COMPILER` de N-Core sans restriction ni modification. Pour ce faire, il est nécessaire de compléter les services du noiZetier par la fonction de service `type_noisette_initialiser_ajax()` qui renvoie la configuration par défaut de l'Ajax à appliquer pour la compilation des noisettes. Cette configuration est une variable de configuration du plugin (voir paragraphe 4.4.1).

Normalement la balise nécessite aussi une autre fonction de service, à savoir, `type_noisette_initialiser_dossier()`, mais cette fonction est omise par le noiZetier car celui-ci utilise le même dossier que N-Core pour rechercher les types de noisette.

Par contre, pour la compilation d'un bloc Z qui est le but du noiZetier, celui-ci n'utilise pas l'inclusion `compiler_conteneur.html` fournie par N-Core mais la surcharge pour utiliser directement une boucle SPIP native sur la table `spip_noisettes` (N-Core utilise une boucle DATA après avoir répertorié les noisettes du conteneur).

## 7.5 Le mode voir\_noisettes

Le noiZetier complète aussi cette inclusion par le fichier `bloc_compiler_editor.html` qui permet d'afficher un bloc avec des enrichissements visuels pour repérer les noisettes et les manipuler comme dans l'espace privé.

**A compléter**

## 7.6 Conclusion

Finalement, la vraie complexité du noiZetier utilisant N-Core est la relation biunivoque entre conteneur et les concepts de squelette, page, composition, objet et bloc. Pour factoriser ces traitements laborieux et répétitifs deux fonctions complémentaires ont été définis spécifiquement pour le noiZetier dans l'API des conteneurs (fichier `inc/noizetier_conteneur`, voir paragraphe 3.4).



## 8. REGLES DE CODAGE

### 8.1 Nommage des fonctions

Le nommage des fonctions appartenant aux différentes API de N-Core suit des règles strictes qui simplifient l'identification de l'objet et de l'action appliquée. Le nom de chaque fonction est donc composée ainsi : `noizetier_<objet>_<verbe_infinitif>`. Par exemple, la fonction de lecture de la description d'une page se nomme `noizetier_page_lire()`.

En outre, la même action se traduit par le même verbe à l'infinitif quel que soit l'objet concerné. Par exemple, la fonction de lecture de la description d'un bloc se nomme `noizetier_bloc_lire()`.

### 8.2 Arguments standardisés

Toutes règles issues des API N-Core s'applique au `noiZetier`. Les variable `$plugin`, `$stockage`, `$type_noisette`, `$noisette`, `$conteneur`, `$information`, `$id_noisette` et `$id_conteneur` désignent les mêmes éléments que dans N-Core.

A ceux-ci le `noiZetier` rajoute les variables standard suivantes :

- `$page`, pour l'identifiant ou la description d'une page ;
- `$type` ou `$type_page` pour le préfixe d'une composition ;
- `$composition` pour le suffixe d'une composition ;
- `$bloc` pour l'identifiant du bloc ;
- `$objet` ou `$type_objet` pour le type d'un objet ;
- `$id_objet` pour l'identifiant unique d'un objet.

## 9. LES FICHIERS YAML / XML

### 9.1 Les types de page explicites

#### 9.1.1 Le nouveau fichier YAML

Le fichier YAML d'un type de page du noiZetier est de la forme suivante :

```
# Titre de la page
# - obligatoire
# - texte ou item de langue
nom: '<:noizetier:type_page_xxxx_nom:>'
# Description du rôle de la page
# - facultatif, vide si absent
# - texte ou item de langue
description: '<:noizetier:type_page_xxxx_description:>'
# Nom de l'icone représentant le type de noisette sans chemin
# - facultatif, 'page-24.png' si absent
icon: 'xxxx-24.png'
# Liste des plugins nécessités pour le fonctionnement de la page
# - facultatif, [] si absent
# - tableau des préfixes de plugin
necessite: ['prefixe1', 'prefixe2']
```

Ce fichier est complété par un schéma JSON qui permet de valider formellement tout fichier YAML de type de page (fonction non intégrée dans la noiZetier actuellement).

#### 9.1.2 Le fichier XML (déprécié)

Le fichier XML n'est plus recommandé pour décrire une page mais cette branche du noiZetier continue à le supporter. Il peut être décrit par une DTD approximative dont la partie spécifique est fournie ci-dessous :

```
<!ENTITY % NAME "CDATA"> <!-- identificateur (notamment nom de fonction) -->
<!ENTITY % ITEM "CDATA"> <!-- chaîne de langue -->
<!ENTITY % PATH "CDATA"> <!-- chemin d'accès relatif à un fichier -->

<!ENTITY % CONTENT "description|icon" >

<!ELEMENT page (nom, %CONTENT;*, necessite*) >

<!ELEMENT nom (%ITEM|#PCDATA) *>

<!ELEMENT description (%ITEM|#PCDATA) *>
<!ELEMENT icon (%PATH) *>

<!ELEMENT necessite EMPTY>
<!ATTLIST necessite
    id %NAME; #REQUIRED
>
```

### 9.2 Les compositions explicites

Le fichier XML pour décrire une page peut être décrit par une DTD approximative dont la partie spécifique est fournie ci-dessous :

```
<!--===== DTD originale =====>

<!ENTITY % NAME "CDATA" <!-- identificateur (type d'objet, composition) -->
<!ENTITY % ITEM "CDATA" <!-- chaine de langue -->
<!ENTITY % PATH "CDATA" <!-- chemin d'accès relatif a un fichier -->

<!ENTITY % CONTENT "(description|icon|image_exemple|class|configuration)*" >

<!ELEMENT composition (nom, %CONTENT;, branche*) >

<!ELEMENT nom (%ITEM|#PCDATA)>

<!ELEMENT description (%ITEM|#PCDATA)>
<!ELEMENT icon (%PATH)>
<!ELEMENT image_exemple (%PATH)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT configuration (#PCDATA)>

<!ELEMENT branche EMPTY>
<!ATTLIST branche
  type %NAME; #REQUIRED
  composition %NAME; #REQUIRED
>
```

## 9.3 Les blocs Z

Le fichier YAML d'un bloc Z du noizetier est de la forme suivante :

```
# Titre du bloc
# - obligatoire
# - texte ou item de langue
nom: '<:noizetier:bloc_xxxx_nom:>'
# Description du rôle du bloc
# - facultatif, vide si absent
# - texte ou item de langue
description: '<:noizetier:bloc_xxxx_description:>'
# Nom de l'icone représentant le bloc sans chemin relatif
# - facultatif, 'bloc-24.png' si absent
icon: 'xxx-24.png'
```