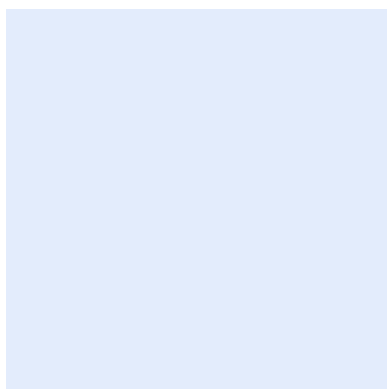


Référence SPIP/N/2017.003



GUIDE CONCEPTION DU PLUGIN NOIZETIER



FICHE D'IDENTIFICATION

Rédacteur	Eric Lupinacci
Projet	SPIP
Étude	Conception du plugin noiZetier
Nature du document	Guide
Date	13/05/2018
Nom du fichier	Guide N - Le plugin noiZetier
Référence	SPIP/N/2017.003
Dernière mise à jour	27/05/2018 18:28:20
Langue du document	Français
Nombre de pages	27

TABLE DES MATIERES

1.	INTRODUCTION	5
2.	CONCEPTS	5
2.1	PAGES SPIP, COMPOSITIONS ET BLOCS Z	5
2.1.1	LES PAGES SPIP	5
2.1.2	LES COMPOSITIONS	5
2.1.3	LES BLOCS Z	6
2.2	LES PAGES DU NOIZETIER	6
2.2.1	LES PAGES EXPLICITES	6
2.2.2	LES COMPOSITIONS VIRTUELLES	6
2.3	LES BLOCS DU NOIZETIER	7
2.4	UTILISATION DES CONCEPTS N-CORE DANS LE NOIZETIER	7
3.	PERIMETRE DU NOIZETIER	7
3.1	L'API DE GESTION DES PAGES	8
3.2	L'API DE GESTION DES BLOCS	9
3.3	L'API DE GESTION DES OBJETS	10
3.4	COMPLEMENT DE L'API DE GESTION DES CONTENEURS	11
4.	FONCTIONNEMENT DU NOIZETIER	11
4.1	L'INTERFACE DE GESTION DES PAGES	11
4.1.1	CHARGEMENT DES PAGES	11
4.1.2	AFFICHAGE DES PAGES	12
4.1.3	EDITION D'UNE PAGE	13
4.1.4	CREATION D'UNE COMPOSITION VIRTUELLE	13
4.1.5	ACTIVATION DES COMPOSITIONS SUR UN TYPE D'OBJET	13
4.2	L'INTERFACE DE GESTION DES NOISETTES	13
4.2.1	AFFICHAGE DES NOISETTES D'UNE PAGE	13
4.2.2	AJOUTER UNE NOISETTE	15
4.2.3	MODIFIER UNE NOISETTE	15
4.2.4	SUPPRIMER UNE NOISETTE	15
4.2.5	DEPLACER UNE NOISETTE	15
4.2.6	VIDER UN BLOC OU UNE PAGE	15
4.3	LES CONTENEURS	15
4.4	LA COMPILATION DES BLOCS	15

5.	DONNEES DU NOIZETIER	16
5.1	LA STRUCTURE DES DONNEES	16
5.1.1	LES PAGES	16
5.1.2	LES BLOCS	17
5.2	LES ESPACES DE STOCKAGE DU NOIZETIER	17
5.2.1	LES PAGES	17
5.2.2	LES TYPES DE NOISETTES ET NOISETTES	18
6.	MISE EN OEUVRE DE N-CORE	18
6.1	LA GESTION DES TYPES DE NOISETTE	18
6.1.1	CHARGER LES TYPES DE NOISETTES	18
6.1.2	LIRE UN TYPE DE NOISETTE	20
6.1.3	CONCLUSION	20
6.2	LA GESTION DES CONTENEURS	21
6.2.1	IDENTIFIER UN CONTENEUR	21
6.2.2	VIDER UN CONTENEUR	21
6.2.3	CONCLUSION	22
6.3	LA GESTION DES NOISETTES	22
6.3.1	AJOUTER UNE NOISETTE A UN CONTENEUR	22
6.3.2	SUPPRIMER UNE NOISETTE D'UN CONTENEUR	23
6.3.3	DEPLACER UNE NOISETTE DANS UN CONTENEUR	23
6.3.4	CONCLUSION	23
6.4	LA COMPILATION DES NOISETTES	24
6.5	CONCLUSION	24
7.	REGLES DE CODAGE	24
7.1	NOMMAGE DES FONCTIONS	24
7.2	ARGUMENTS STANDARDISES	25
8.	LES FICHIERS YAML / XML	26
8.1	LES TYPES DE PAGE EXPLICITES	26
8.1.1	LE NOUVEAU FICHIER YAML	26
8.1.2	LE FICHIER XML (DEPRECIE)	26
8.2	LES COMPOSITIONS EXPLICITES	27
8.3	LES BLOCS Z	27

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>` incluse contenant les meta, CSS et autres inclusions JS,
- une balise `<body>` incluse 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`.

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 le fichier `dist.html` par défaut 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.

Ces pages seront appelées **pages explicites**. Un page explicite peut donc être une **composition explicite** ou pas et donc coïncider avec le **type de page**.

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 mais juste une liste de noisettes propres qui seront insérées soit dans une page de type d'objet comme `article.html` soit dans une page par défaut, `dist.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, soit en configurant pour les pages explicites, une liste de blocs autorisés dans le fichier YAML ou XML (solution dépréciée mais maintenue par compatibilité), soit en définissant pour toutes pages et compositions, une liste de blocs exclus de l'ajout de noisettes via un formulaire dédié.

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 6).

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. Le noiZetier fournit la fonction de service de calcul de leur identifiant - `noizetier_conteneur_identifier()`. L'application de cette fonction de service aux conteneurs précités donne les identifiants `content/article`, `nav/article-forum` et `aside/article|article|12`.

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 noisettes et surtout des conteneurs. Il utilise la balise `#NOISETTE_COMPILER` fournie par N-Core lors de la compilation d'un bloc Z (voir le paragraphe 4.4).

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

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

noizetier_page_charger	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.
noizetier_page_compter_noisettes	Renvoie, pour une page donnée, la liste des blocs ayant des noisettes ajoutées et renvoie leur nombre pour chacun des blocs. Le tableau de sortie est de la forme [bloc] = nombre de noisettes. Les blocs vides ne sont pas fournis.
noizetier_page_extraire_composition	Extrait l'identifiant de la composition ou renvoie une chaîne vide.
noizetier_page_extraire_type	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.
noizetier_page_initialiser_dossier	Retourne le dossier relatif dans lequel chercher les pages explicites.

noizetier_page_lire	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.
noizetier_page_lister_blocs	Renvoie la liste des blocs de la page pour lequel l'ajout de noisettes est autorisé.
noizetier_page_composition_activee	Détermine si les compositions sont activées (configuration du plugin Compositions) sur le type de page passé en argument.

L'API fonctionnelle est complétée par une balise `#NOIZETIER_PAGE_INFOS` permettant de fournir toutes les informations utiles sur une page donnée, explicite ou virtuelle. La signature de la balise est `#NOIZETIER_PAGE_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 de la page en contenant.

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

noizetier_bloc_lister_defaut	Renvoie la liste par défaut des identifiants de blocs 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.
noizetier_bloc_lire	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 `#NOIZETIER_BLOCS_INFOS` permettant de fournir toutes les informations utiles sur un bloc donné. La signature de la balise est `#NOIZETIER_BLOC_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 de page 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

noizetier_objet_compter_noisettes	Renvoie, pour un objet donné, la liste des blocs ayant des noisettes ajoutées et renvoie leur nombre pour chacun des blocs. Le tableau de sortie est de la forme [bloc] = nombre de noisettes. Les blocs vides ne sont pas fournis.
noizetier_objet_lire	Renvoie la description complète ou uniquement une information précise pour un objet donné. Étant donné que chaque objet possède une liste spécifique de champs, on ne retourne que 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).
noizetier_objet_repertorier	Lister les objets ayant des noisettes spécifiquement ajoutées. La description complète de l'objet - au sens de la fonction <code>noizetier_objet_lire()</code> - est renvoyée. La liste des objets peut être filtrée sur un ou plusieurs champs de l'objet.
noizetier_objet_type_active	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 `noizetier_objet_lire()` et `noizetier_objet_repertorier()`, à savoir :

- `#NOIZETIER_OBJET_INFOS{type_objet, id_objet[, information]}`;
- et `#NOIZETIER_OBJET_LISTE`.

Pour la balise `#NOIZETIER_OBJET_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.

C'est l'objet du complément fonctionnel de l'API des conteneurs présenté ci-dessous.

API CONTENEURS : INC/NOIZETIER_CONTENEUR.PHP

noizetier_conteneur_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 de service

`noizetier_conteneur_identifier()`.

noizetier_conteneur_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.

4. FONCTIONNEMENT DU NOIZETIER

4.1 L'interface de gestion des pages

4.1.1 Chargement des 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 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 de ces pages.

4.1.2 Affichage des pages

Les pages sont affichées dans deux listes distinctes, celle dont le type coïncide avec celui d'un type d'objet (comme article) et les autres pages (comme sommaire ou plan).

Ces listes correspondent à deux 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.

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 noisettes

noiZetier

Sélectionnez la page dont vous souhaitez configurer les noisettes.

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

4.1.3 Edition d'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`.

4.1.4 Création d'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 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 traitées dans le formulaire d'édition d'une page, `editer_page`.

4.1.5 Activation des 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 de gestion des noisettes

4.2.1 Affichage des noisettes d'une page

La manipulation des noisettes d'une page Z donnée se fait dans la page de l'espace privée 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. Les noisettes incluses dans une noisette conteneur sont visualisées avec une légère indentation pour montrer l'imbrication.


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
 -  Bloc de texte libre
Texte 1
 -  Bloc de texte libre
Texte 2
 -  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

un bouton « Ajouter une noisette » qui renvoie vers un formulaire permettant de choisir la noisette à ajouter au bloc concerné ;

- un mécanisme de drag'n drop d'un item de la liste des noisettes disponibles vers un bloc donné de la page en cours avec le choix de la position de la noisette dans la liste des noisettes déjà insérées.

4.2.3 Modifier une noisette

4.2.4 Supprimer une noisette

4.2.5 Déplacer une noisette

4.2.6 Vider un bloc ou une page

4.3 Les conteneurs

4.4 La compilation des blocs

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>	
page	Identifiant de la page. Correspond au nom du fichier YAML, XML ou HTML sans extension.
type	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.
composition	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.
nom	Titre de la page sous forme textuelle ou d'un item de langue. Par défaut coïncide avec l'identifiant de la page.
description	Texte ou item de langue décrivant le rôle de la page. Peut-être vide.
icon	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.
neccesite	Liste des plugins – préfixes – nécessairement actifs pour utiliser la page. Ce champ est un tableau, éventuellement vide, de format « [] = préfixe ».
branche	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 <code>[type_objet] = composition</code> .
<i>Données complémentaires</i>	
blocs_exclus	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.

est_active	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 ».
est_virtuelle	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.
est_page_objet	Prend la valeur « oui » pour une page de type objet et « non » sinon. Ce champ est calculé à partir du champ type.
signature	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.

DESCRIPTION D'UN BLOC	
<i>Données issues du fichier YAML ou HTML</i>	
nom	Identifiant du bloc, c'est-à-dire le nom du dossier.
description	Texte ou item de langue décrivant le rôle du bloc. Peut-être vide.
icon	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.

5.2 Les espaces de stockage du noiZetier

5.2.1 Les pages

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`.

L'identifiant unique de chaque page est son identifiant, à savoir, le champ `page` de la structure.

5.2.2 Les types de noisettes et noisettes

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.

Pour les types de noisette, le noiZetier, par l'intermédiaire de la fonction de service `noizetier_type_noisette_completer()`, ajoute les champs `type` et `composition` déduits de l'identifiant du type de noisette.

Pour les noisettes, le noiZetier, par l'intermédiaire de la fonction de service `noizetier_noisette_completer()`, ajoute les champs `type`, `composition`, `bloc`, `objet` et `id_objet` qu'il déduit du champ `conteneur` passé par N-Core et qui, lui, contient le squelette et éventuellement l'objet précis ou la noisette concernée. Puisque le noiZetier stocke les éléments qui constitue le conteneur d'une noisette (en plus de l'`id_conteneur`), il ne stocke pas le tableau associatif du conteneur fourni par N-Core.

6. MISE EN OEUVRE DE N-CORE

Outre l'utilisation des fonctions d'API de N-Core, la mise en œuvre de N-Core par le noiZetier consiste principalement à coder les fonctions de service nécessaire dans le fichier `ncore/noizetier.php`.

6.1 La gestion des types de noisette

6.1.1 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.

Pour que cette API N-Core fonctionne avec l'espace de stockage du noiZetier, celui-ci doit proposer les 3 fonctions de services suivantes car il est inutile de surcharger le service `type_noisette_initialiser_dossier` étant donné que le noiZetier, comme N-Core, utilise le dossier relatif `noisettes/` pour rechercher les types de noisette disponibles :

- `noizetier_type_noisette_completer()`, qui va ajouter les champs `type` et `composition` ;
- `noizetier_type_noisette_lister()`, qui renvoie la liste des types de noisettes enregistrés dans la table `spip_types_noisettes` ;
- `noizetier_type_noisette_stocker()`, qui enregistre les types de noisettes dans la table `spip_types_noisettes` en distinguant les types de noisette ajoutés, modifiés et supprimés, fournis par N-Core.

Le code de ces fonctions de service est fourni ci-après.

```
function noizetier_type_noisette_completer($plugin, $description) {
    $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;
}
```

```
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';

    if (sql_preferer_transaction()) {
        sql_demarrer_transaction();
    }
    $where = array('plugin=' . sql_quote($plugin));
    // -- Suppression des noisettes obsolètes ou de toute les noisettes d'un coup si
    // on est en mode rechargement forcé.
    if ($recharger) {
        sql_delete($from, $where);
    } elseif (!empty($types_noisettes['a effacer'])) {
        $where[] = sql_in('type_noisette', $types_noisettes['a effacer']);
        sql_delete($from, $where);
    }
    // -- Update des pages modifiées
    if (!empty($types_noisettes['a changer'])) {
        sql_replace_multi($from, $types_noisettes['a changer']);
    }
    // -- Insertion des nouvelles pages
    if (!empty($types_noisettes['a ajouter'])) {
        sql_insertq_multi($from, $types_noisettes['a ajouter']);
    }
    if (sql_preferer_transaction()) {
        sql_terminer_transaction();
    }

    return $retour;
}
```

```
function noizetier_type_noisette_lister($plugin, $information = '') {
    $where = array('plugin=' . sql_quote($plugin));
    $select = $information ? array('type_noisette', $information) : '*';
    if ($info_noisettes = sql_allfetsel($select, 'spip_types_noisettes', $where)) {
        if ($information) {
            $info_noisettes = array_column($info_noisettes, $information, 'type_noisette');
        } else {
            $info_noisettes = array_column($info_noisettes, null, 'type_noisette');
        }
    }
    return $info_noisettes;
}
```

On peut remarquer que le code est très simple car toute la complexité des traitements est inscrite une fois pour toute dans les fonctions d'API de N-Core.

6.1.2 Lire un type de noisette

Pour vérifier la saisie correcte des paramètres d'une noisette, le noiZetier fait appel à l'API `type_noisette_lire()` en limitant l'information requise au champ paramètre. L'appel est de la forme :

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

Pour utiliser cette API, le noiZetier doit au préalable définir une quatrième fonction de service des types de noisettes, à savoir, `noizetier_type_noisette_decrire()` qui renvoie la description complète du type de noisette. Le code de cette fonction est trivial :

```
function noizetier_type_noisette_decrire($plugin, $type_noisette) {
    // Chargement de toute la configuration de la noisette en base de données.
    // Les données sont renvoyées brutes sans traitement sur les textes
    // ni sur les tableaux sérialisés.
    $where = array('plugin=' . sql_quote($plugin), 'type_noisette=' .
    sql_quote($type_noisette));
    $description = sql_fetsel('*', 'spip_types_noisettes', $where);
    return $description;
}
```

6.1.3 Conclusion

Les 4 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 4 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

6.2.1 Identifier un conteneur

Pour le noiZetier, un conteneur est un bloc d'une page ou un bloc d'un objet SPIP donné. En conséquence, un conteneur est donc 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 (sans oublier la noisette conteneur fournie par N-Core).

Comme on l'a vu au paragraphe 5.2.2, la structure de données des noisettes contient tous les éléments concourant à l'identification du conteneur d'une noisette : type, composition, objet, id_objet, bloc. Mais cet espace contient aussi l'identifiant du conteneur sous une forme spécifique au plugin noiZetier, résultat de la fonction de service `noizetier_conteneur_identifier()`.

Le code de cette fonction est le suivant :

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

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

    if ($conteneur) {
        // Cas d'une noisette conteneur.
        if (!empty($conteneur['type_noisette']) and intval($conteneur['id_noisette'])) {
            $identifiant = $conteneur['type_noisette'] . '|noisette|' .
            $conteneur['id_noisette'];
        } else {
            // Le nom du squelette en premier si il existe (normalement toujours).
            if (!empty($conteneur['squelette'])) {
                $identifiant .= $conteneur['squelette'];
            }
            // L'objet et son id si on est en présence d'un objet.
            if (!empty($conteneur['objet']) and !empty($conteneur['id_objet']) and
            intval($conteneur['id_objet'])) {
                $identifiant .= ($identifiant ? '|' : '') .
                "{$conteneur['objet']}|{$conteneur['id_objet']}";
            }
        }
    }

    return $identifiant;
}
```

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`. La noisette conteneur aura elle un identifiant de la forme `conteneur|noisette|8`.

6.2.2 Vider un conteneur

Le noiZetier propose un bouton « Supprimer les noisettes du bloc » dans chaque formulaire de configuration d'un bloc de page pour vider l'ensemble de ses noisettes. Pour ce faire, le noiZetier utilise l'API de N-Core `conteneur_vider()` en passant comme argument l'identifiant du conteneur

concerné. Pour fonctionner cette API a besoin d'une fonction de service nommée `noizetier_conteneur_destocker()` dont le code est fourni ci-dessous.

```
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)) {
        $id_conteneur = noizetier_conteneur_identifier($plugin, $conteneur);
    } else {
        $id_conteneur = $conteneur;
    }

    if ($id_conteneur) {
        // Suppression de toutes les noisettes du conteneur.
        $where = array('id_conteneur=' . sql_quote($id_conteneur));
        if (sql_delete('spip_noisettes', $where)) {
            $retour = true;
        }
    }

    return $retour;
}
```

Cette fonction prend en compte les deux façons d'identifier un conteneur, par son id ou par son tableau descriptif. Ceci est nécessaire si l'on veut rendre l'espace de stockage du noiZetier réutilisable comme une librairie par d'autres plugins.

6.2.3 Conclusion

Encore une fois, le code de ces 2 fonctions de service est trivial et l'utilisation des API N-Core reste simple. Il est à noter que la **fonction d'identification du conteneur doit toujours être définie** par le plugin utilisateur.

6.3 La gestion des noisettes

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

6.3.1 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)
```

Pour que cette API N-Core fonctionne avec l'espace de stockage du noiZetier, celui-ci doit proposer les 4 fonctions de services suivantes :

- `noizetier_noisette_completer()`, qui va ajouter les champs `type`, `composition`, `objet`, `id_objet` et `bloc` à la description de la noisette ;
- `noizetier_noisette_lister()`, qui renvoie la liste des noisettes déjà présentes dans le conteneur et enregistrées dans la table `spip_noisettes` ;
- `noizetier_noisette_ranger()`, qui positionne le rang de la noisette dans le conteneur ;
- `noizetier_noisette_stocker()`, qui enregistre la noisette dans la table `spip_noisettes`.

Comme pour les autres fonctions de service, le code est relativement simple. La seule complexité réside pour certaines fonctions dans le traitement des deux cas d'identification d'un conteneur, par son identifiant ou par sa description tabulaire. Comme précisé plus tôt dans ce document, cette prise en compte est utile quand on veut exposer une librairie de stockage pour d'autres plugins ce qui est le cas pour le noiZetier.

6.3.2 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, et ce de façon récursive.

Pour utiliser cette fonction, il est nécessaire de compléter le noiZetier par 2 nouvelles fonctions de service, à savoir :

- `noizetier_noisette_decrire()`, qui renvoie la description complète d'une noisette ;
- `noizetier_noisette_destocker()`, qui supprime la noisette de la table `spip_noisettes`.

Le code de ces fonctions est simplissime car il se contente de faire appel à un `sql_fetch()` pour la première et à un `sql_delete()` pour la seconde.

6.3.3 Déplacer 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`. Toutes les fonctions de service nécessaires ont déjà été codées, à savoir, `noizetier_noisette_decrire()`, `noizetier_noisette_lister()` et `noizetier_noisette_ranger()`.

6.3.4 Conclusion

L'utilisation de l'API de gestion des noisettes de N-Core par le noiZetier est somme toute assez aisée. Comme pour la gestion des types de noisettes, le code des fonctions de services reste simple, la seule complexité étant de gérer les deux façons d'identifier une noisette, l'id de la table `spip_noisettes` ou le couple (`id_conteneur`, `rang`).

6.4 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 de 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 un paramètre de configuration du plugin.

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 (voir le paragraphe 4.4), celui-ci n'utilise pas l'inclusion `compiler_conteneur.html` voire `compiler_noisettes.html` mais définit sa propre inclusion nommée `bloc_compiler.html` avec une boucle SPIP sur la table `spip_noisettes`.

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

6.5 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 définies spécifiquement pour le noiZetier dans l'API des conteneurs (fichier `inc/noizetier_conteneur`, voir paragraphe 3.4).

7. REGLES DE CODAGE

7.1 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()`.

7.2 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.

8. LES FICHIERS YAML / XML

8.1 Les types de page explicites

8.1.1 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 dde 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).

8.1.2 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"> <!-- chaine 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
>
```

8.2 Les compositions explicites

Le fichier XML pour décrire une 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
>
```

8.3 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'
```