SPIP

SPIP **08/04/2018**

Référence SPIP/N/2017.002

GUIDE
CONCEPTION DU PLUGIN N-CORE

SPIP

FICHE D'IDENTIFICATION	
Rédacteur	Eric Lupinacci
Projet	SPIP
Étude	Conception du plugin N-Core
Nature du document	Guide
Date	08/04/2018
Nom du fichier	Guide N - Le plugin N-Core.docx
Référence	SPIP/N/2017.002
Dernière mise à jour	15/04/2018 19:24:08
Langue du document	Français
Nombre de pages	26

SPIP

TABLE DES MATIERES

1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception du plugin N-Core dont l'objectif est de fournir des API génériques de gestion des noisettes et de leur compilation.

N-Core ne fournit aucune interface utilisateur. Le noiZetier, dans sa version 3, utilise N-Core et offre une interface d'administration permettant de configurer et d'insérer au choix des noisettes dans les diverses pages publiques du site.

D'autres plugins pourront ainsi être développés à partir de N-Core en particulier pour fournir des interfaces utilisateur pour associer des noisettes et des squelettes ou d'autres objets.

2. CONCEPTS

2.1 Les squelettes

La mise en page d'un site SPIP est effectuée au moyen de **gabarits au format HTML** nommés **squelettes**, contenant des instructions simplifiées permettant d'indiquer où et comment se placent les informations tirées de la base de données dans la page web.

Un squelette est donc un fichier HTML installé dans un site SPIP qui affiche une page comme `article.html` ou une partie d'une page web comme `content/article.html`.

L'identifiant d'un squelette est son chemin relatif sans extension (par exemple `content/article`).

2.2 Les conteneurs

Dans le noiZetier, par défaut, les noisettes sont associées à un squelette. N-Core, lui étend cette notion en permettant d'associer dans un ordre précis des noisettes à un **conteneur** qui peut être tout autre chose qu'un squelette.

Par exemple, le plugin noiZetier permet d'insérer des noisettes dans un bloc de page et ce pour un contenu précis (i.e. l'article 12). Ce faisant, il met en relation des noisettes, un squelette comme `content/article.html` et l'article concerné. Le couple (squelette, article) est en fait un conteneur.

Le conteneur peut aussi servir à choisir une liste de noisettes pour chaque utilisateur affichant une page ainsi composée. Dans ce cas, le conteneur est l'auteur concerné. Un type de noisette étant aussi un squelette, une noisette peut aussi être un conteneur et ainsi inclure d'autres noisettes.

Un conteneur est matérialisé par un **tableau associatif**. Les index sont libres à l'exception de l'index `squelette` qui permet d'utiliser un squelette comme conteneur ou comme élément d'un conteneur. Un conteneur possède aussi un identifiant unique de type chaine, défini spécifiquement par le plugin utilisateur (par exemple, `content/article12`).

2.3 Les types de noisettes

Les types de noisette sont les composants de base de N-Core. Un **type de noisette est un squelette HTML** autonome, suffisamment **générique** pour être **réutilisable** dans différentes pages ou sites et pouvant être **configurable**. Il est toujours associé à un fichier YAML qui décrit l'ensemble de ses caractéristiques.

L'identifiant d'un type de noisette est le nom du fichier associé sans extension (par exemple, article-cartouche).

2.4 Les noisettes

Une noisette est une **« instance paramétrée d'un type de noisette » incluse dans un conteneur** donné.

Outre son type, la noisette se distingue par son paramétrage (valeur de chaque paramètre attaché au type de noisette) et par un ensemble d'informations de contexte qui permettent de la compiler lors de son inclusion.

Il est possible d'inclure plusieurs noisettes à la suite dans un conteneur. Il est donc nécessaire de gérer un rang pour chaque noisette incluse dans un conteneur.

Toute noisette possède un identifiant unique qui se nomme `id_noisette` et qui peut être un entier ou une chaine de caractères pour autant qu'il soit unique pour une utilisation donnée. Il est aussi possible d'identifier de façon unique une noisette avec le conteneur auquel elle est associée et son rang. L'utilisation optimale de l'un ou l'autre identifiant dépend de la structure de stockage adoptée.

3. PERIMETRE DE N-CORE

N-Core propose plusieurs API :

- La gestion des types de noisettes, à savoir, le chargement des fichiers YAML, leur stockage et la lecture des informations stockées ;
- La gestion des noisettes, à savoir, un « CRUD étendu » avec un stockage dédié ;
- La gestion des conteneurs ;
- La compilation des noisettes, à savoir, la gestion du contexte et des paramètres de chaque noisette et leur affichage ;
- La gestion de fichiers caches pour le stockage à accès rapide des types de noisettes et de certains éléments de contexte.

3.1 L'API de gestion des types de noisette

La gestion des types de noisette consiste à stocker les descriptions dans un espace à accès rapide et à permettre leur lecture et leur mise à jour.

API TYPES DE NOISETTE : INC/NCORE_TYPE_NOISETTE.PHP	
type_noisette_charger	Charge ou recharge les descriptions des types de noisette à partir des fichiers YAML. Les types de noisette sont recherchés dans un répertoire relatif fourni en argument. La fonction optimise le chargement en effectuant uniquement les traitements nécessaires en fonction des modifications, ajouts et suppressions des types de noisette identifiés en comparant les md5 des fichiers YAML.
type_noisette_lire	Retourne, pour un type de noisette, la description complète ou seulement un champ précis. Les champs textuels peuvent être fournis bruts ou avec un traitement typo.
type_noisette_repertorier	Renvoie une liste de types de noisette éventuellement filtrée sur certains champs. Les données sont renvoyées brutes sous forme d'un tableau indexé par l'identifiant de chaque type de noisette

3.2 L'API de gestion des noisettes

L'API de gestion des noisettes fournit une interface de type « CRUD étendue » pour associer des instances de type de noisette à un conteneur. L'interface utilisateur permettant le choix du conteneur et des types de noisette ne fait pas partie de N-Core.

API NOISETTES : INC/NCORE_NOISETTE.PHP	
noisette_ajouter	Ajoute à un conteneur, à un rang donné ou en dernier rang, une noisette d'un type donné et met à jour le rang des autres noisettes du conteneur si nécessaire. Si le rang n'est pas précisé, la noisette est ajoutée en dernier rang. Si tout s'est bien passé la fonction renvoie l'identifiant unique de la noisette.
noisette_lire	Retourne, pour une noisette, la description complète ou seulement un champ précis. Les champs textuels peuvent subir un traitement typo si demandé.
noisette_parametrer	Met à jour les paramètres de configuration de la noisette destinés à son affichage.
noisette_supprimer	Supprime une noisette donnée et met à jour les rangs des autres noisettes du conteneur si nécessaire.
noisette_deplacer	Déplace une noisette de sa position au sein d'un conteneur et met à jour le rang des autres noisettes du conteneur si nécessaire.

noisette_repertorier	Renvoie une liste de noisettes, appartenant ou pas à un conteneur et éventuellement filtrée sur certains champs. Les données sont renvoyées brutes sous forme d'un tableau indexé soit par l'identifiant de chaque noisette soit par le couple (identifiant de conteneur, rang).

3.3 L'API de gestion des conteneurs

L'API de gestion des conteneurs fournit une interface aujourd'hui limitée au vidage des noisettes d'un conteneur.

API CONTENEURS : INC/NCORE_CONTENEUR.PHP	
conteneur_vider	Supprime toutes les noisettes rangées dans un conteneur.

3.4 L'API de compilation des noisettes

La véritable API de compilation des noisettes est la balise `#NOISETTE_COMPILER` qui construit l'affichage de la noisette concernée.

La **balise devant être appelée dans une boucle de noisettes**, les données de la noisette en cours de compilation sont toutes accessibles via la fonction `champ_sql()`. Seul le stockage spécifique si il diffère de celui du plugin appelant est à fournir en argument optionnel de la balise.

Pour fonctionner, la balise `#NOISETTE_COMPILER` utilise des filtres nécessaires à la détermination du contexte de la noisette, de son inclusion en ajax et de son inclusion dynamique ou pas.

API COMPILATION : NCORE_FONCTIONS.PHP	
noisette_contextualiser	Construit le contexte d'une noisette donnée à partir de la configuration de son type, de l'environnement et de son identifiant – id_noisette ou couple (conteneur, rang). La configuration du contexte des types de noisette est stockée dans un cache dédié qui est géré par la fonction.
type_noisette_ajaxifier	Détermine si une noisette doit être incluse en ajax ou pas en fonction de la configuration de son type de noisette et de la configuration générale du plugin. L'indicateur ajax de chaque type de noisette est stocké dans un cache dédié géré par la fonction.
type_noisette_dynamiser	Détermine si une noisette doit être incluse dynamiquement ou pas en fonction de la configuration de son type de noisette. L'indicateur d'inclusion dynamique de chaque type de noisette est stocké dans un cache dédié géré par la fonction.

Ces filtres sont présentés comme une API car ils peuvent éventuellement servir à un plugin appelant qui souhaite surcharger la balise `#NOISETTE_COMPILER`.

3.5 L'API de gestion des caches

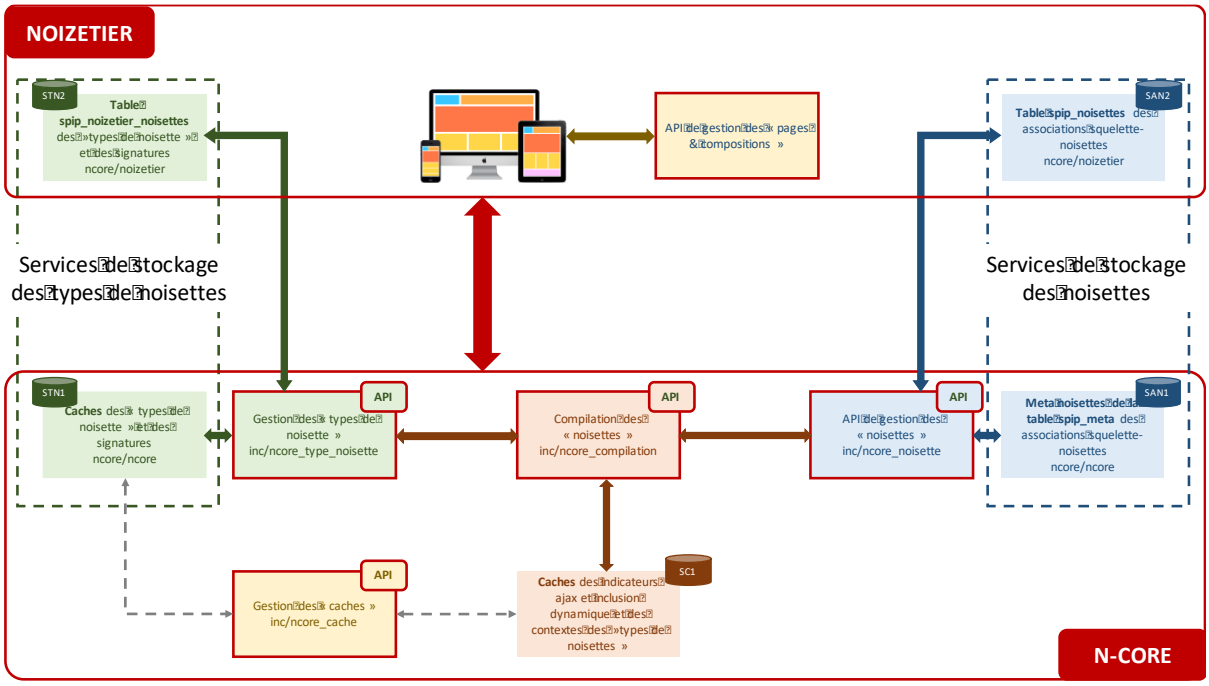
La gestion des caches est principalement à usage interne N-Core car celui-ci utilise de nombreux caches comme moyen de stockage. Néanmoins, cette interface est exposée afin de rester utilisable par d'autres plugins.

API CACHES : INC/NCORE_CACHE.PHP	
cache_lire	Lit le cache spécifié et renvoie le contenu sous forme de tableau éventuellement vide.
cache_ecrire	Ecrit le contenu d'un tableau dans le cache spécifié.
cache_supprimer	Supprime le cache spécifié.

4. FONCTIONNEMENT DE N-CORE

4.1 Schéma de principe

Le fonctionnement global du plugin N-Core est illustré ci-dessous en regard de ce que pourrait être le plugin noiZetier v3.



4.2 La dissociation API – Services

De façon générale, un plugin utilisateur comme le noiZetier va s'appuyer sur l'ensemble des API publiques N-Core (types de noisette, noisettes, conteneur et compilation). Néanmoins, un plugin utilisateur pourrait se passer des API de gestion des types de noisette, noisettes et conteneurs mais pas de l'API de compilation.

Si un plugin utilisateur choisit d'utiliser les API de gestion, il doit définir le stockage qu'il souhaite pour ses types de noisette ou ses noisettes. Par conception, **N-Core dissocie la fonction de gestion d'un objet,de l'espace et des services de stockage de ce même objet**.

Dans le code de ses API, N-Core appelle des fonctions de service qui :

- si la fonction de service homonyme existe dans le plugin utilisateur, va l'appeler et utiliser le stockage propre au plugin ;
- sinon, va dérouler la fonction de N-Core avec son propre stockage.

Mais N-Core va plus loin en permettant à un **plugin utilisateur d'exposer ses fonctions de service et leur stockage comme une librairie**. Le stockage et les fonctions de service d'un plugin utilisateur sont alors réutilisables par un autre plugin utilisateur à l'instar de celui de N-Core. Par exemple, un plugin « préfixe » pourrait utiliser le stockage des noisettes du plugin noiZetier, à savoir, la table SPIP dédiée

à cet usage. Cette fonctionnalité impose bien entendu des contraintes aux fonctions de service qui sont détaillées ci-après.

Toute fonction d'API de N-Core possède deux arguments incontournables, `$plugin` qui est obligatoire et `$stockage` qui est optionnel, comme on peut le voir sur le prototype de `type_noisette_charger()` :

```
function type_noisette_charger($plugin, $dossier = 'noisettes/', $recharger = false, $stockage = '')
```

L'argument $plugin qualifie le module appelant, généralement un plugin comme le noiZetier. Il est donc recommandé d'utiliser le **préfixe du plugin** comme identifiant unique. Cet argument permet de distinguer les espaces de stockage d'un plugin utilisateur par rapport à d'autres. Par exemple, N-Core utilise un cache pour stocker les types de noisette dont le chemin dans `_DIR_CACHE` est `ncore/${plugin}/types_noisette_description.php`.

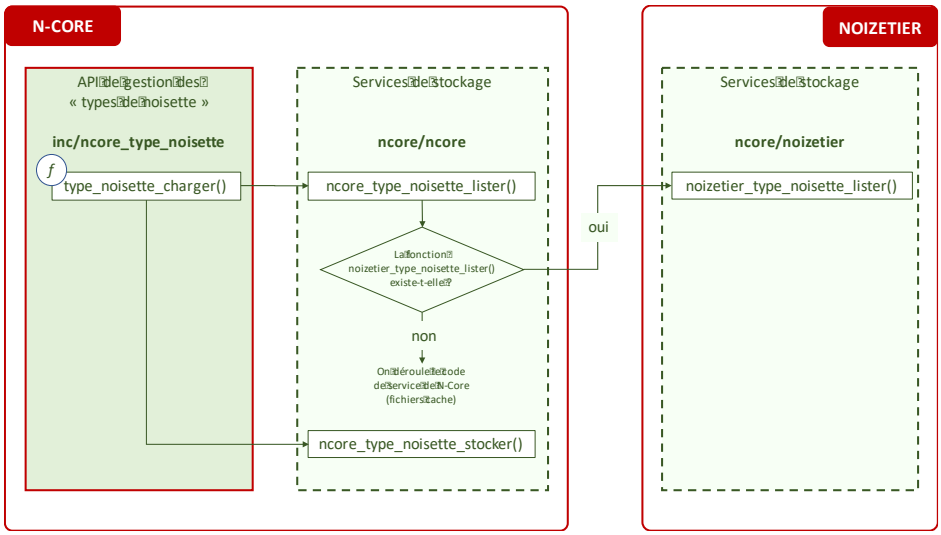
L'argument `$stockage`, lui, permet la réutilisation des services d'un plugin utilisateur par un autre plugin utilisateur ou de forcer l'utilisation des services N-Core (ce qui se fait plus simplement en omettant de créer le service homonyme dans le plugin appelant). Par exemple, le plugin utilisateur « préfixe » pourrait appeler le stockage du noiZetier pour charger ses types de noisette :

```
type_noisette_charger('prefixe', 'types_noisette/',  false, 'noizetier');
```

Il est donc impératif que les plugins utilisateur prévoit toujours cette possibilité dans la conception de leur espace de stockage. Par exemple, pour le noiZetier, une colonne « plugin » a été rajoutée dans les tables de stockage des types de noisette et des noisettes.

4.3 L'aiguillage des services

Par conception et pour des raisons de lisibilité du code, les fonctions d'API de N-Core appellent systématiquement les fonctions de service de N-Core. Ce sont ces **fonctions de service de N-Core qui réalisent l'aiguillage vers le service souhaité**.



Le schéma ci-dessus illustre de déroulement du code suite à l'appel par le plugin noiZetier :

```
type_noisette_charger('noizetier', 'noisettes/');
```

La fonction `type_noisette_charger()` de l'API fait appel à une fonction de service de N-Core, `ncore_type_noisette_lister()`, qui doit renvoyer la liste des signatures des fichiers YAML des types de noisettes. Cette fonction de service va déterminer quelle fonction appliquer, celle du noiZetier ou elle-même. Pour cela, elle appelle une fonction utilitaire `ncore_chercher_service()` qui lui retourne le nom de la fonction de service ou vide si aucune fonction n'est définie dans le plugin appelant :

```
// Initialisation du tableau de sortie
$types_noisettes = array();

// On cherche le service de stockage à utiliser.
include_spip('inc/ncore_utils');
if ($lister = ncore_chercher_service($plugin, 'type_noisette_lister', $stockage)) {
   // On passe le plugin appelant à la fonction car cela permet ainsi de mutualiser
les services de stockage.
   $types_noisettes = $lister($plugin, $information);
} else {
   // Le plugin ne propose pas de fonction propre ou le stockage N-Core est
explicitement demandé.
   // ...suite du code de la fonction de service N-Core.
```

Le code de la fonction utilitaire `ncore_chercher_service()` est le suivant :

```
function ncore_chercher_service($plugin, $service, $stockage = '') {

   $fonction_trouvee = '';

   // Si le stockage n'est pas précisé on cherche la fonction dans le plugin
appelant.
   if (!$stockage) {
      $stockage = $plugin;
   }

   // Eviter la réentrance si on demande explicitement le stockage N-Core
   if ($stockage != 'ncore') {
      include_spip("ncore/${stockage}");
      $fonction_trouvee = "${stockage}_${service}";
      if (!function_exists($fonction_trouvee)) {
         $fonction_trouvee = '';
      }
   }

   return $fonction_trouvee;
}
```

4.4 Les services

Les fonctions d'API de N-Core font donc appel à des services dont la liste précise est fournie ci-après.

SERVICES	
Groupe des services de gestion des types de noisette	
type_noisette_completer	Complète si nécessaire la description d'un type de noisette issue de la lecture de son fichier YAML avant son stockage. Ce service est facultatif.
type_noisette_stocker	Stocke les descriptions des types de noisette en distinguant les types de noisette obsolètes, les types de noisettes modifiés et les nouveaux types de noisettes. Chaque description de type de noisette est un tableau associatif dont tous les index possibles - y compris la signature - sont initialisés quel que soit le contenu du fichier YAML.
type_noisette_decrire	Renvoie la description brute d'un type de noisette sans traitement typo des champs textuels ni désérialisation des champs de type tableau sérialisé.
type_noisette_lister	Renvoie, pour l'ensemble des types de noisette utilisés par le plugin appelant, le champ demandé ou la description complète si aucun champ n'est explicitement spécifié. Les données renvoyées sont brutes sous forme d'un tableau indexé par l'identifiant (nom du fichier YAML sans extension) de chaque type de noisette.
Groupe des services de gestion des noisettes	
noisette_completer	Complète si nécessaire la description d'une noisette avec des champs spécifiques au plugin utilisateur. Ce service est facultatif.
noisette_stocker	Stocke la description d'une nouvelle noisette ou modifie les paramètres d'affichage d'une noisette existante.
noisette_ranger	Positionne une noisette à un rang différent de celui qu'elle occupe dans le conteneur.
noisette_destocker	Efface de l'espace de stockage la description d'une noisette donnée.
noisette_lister	Renvoie, pour un conteneur ou pour l'ensemble des noisettes utilisées par le plugin utilisateur, le champ demandé ou la description complète si aucun champ n'est explicitement spécifié. Les données sont renvoyées brutes sous forme d'un tableau indexé soit par l'identifiant de chaque noisette soit par le couple (identifiant de conteneur, rang).
noisette_decrire	Renvoie la description brute d'une noisette sans traitement typo des champs textuels ni désérialisation des champs de type tableau sérialisé.

Groupe des services de gestion des conteneurs	
conteneur_identifier (*)	Renvoie, pour un conteneur donné, son identifiant calculé à partir de ses éléments. L'identifiant est une chaine de caractères unique non vide.
conteneur_destocker	Retire, de l'espace de stockage, toutes les noisettes d'un conteneur.
Groupe des services de compilation des noisettes	
type_noisette_initialiser_ajax	Renvoie la configuration par défaut de l'ajax à appliquer pour la compilation des noisettes. Cette information est utilisée si la description YAML d'un type noisette ne contient pas de tag ajax ou contient un tag ajax à 'defaut'.

Les services notés (*) doivent toujours être définies par le plugin utilisateur, les autres sont optionnels. Les services sont toutefois indissociables par « groupe » : si on définit un service comme `type_noisette_stocker()`, il faut alors définir tous les services non facultatifs du même groupe, à savoir, dans ce cas ceux gérant les types de noisette à l'exception de `type_noisette_completer()` si rien n'est à rajouter.

N-Core propose l'ensemble de ces services dans son fichier `ncore/ncore.php` associé à ses propres espaces de stockage ce qui permet de minimiser les développements pour la plupart des plugins utilisateur. Néanmoins, le service `conteneur_identifier()` de N-Core ne retourne rien par défaut ; N-Core considère que la construction de l'identifiant est un sujet spécifique au plugin appelant.

Le noiZetier v3, propose, lui, l'ensemble des services dans son fichier `ncore/noizetier.php` car il utilise ses propres espaces de stockage.

4.5 La compilation des noisettes

N-Core utilise une balise nommée `#NOISETTE_COMPILER` pour afficher une noisette donnée. La compilation d'une noisette s'effectue via le pipeline `recuperer_fond` en tenant compte de plusieurs paramètres distincts :

- Le **contexte de la noisette** qui est une combinaison de l'environnement de la page dans laquelle est inclue la noisette, de la configuration du contexte du type de noisette concernée et des variables spécifiques liées à la noisette comme son identifiant ;
- l'**indicateur d'inclusion dynamique** qui est défini au niveau de chaque type de noisette ;
- l'**indicateur d'inclusion en Ajax** qui est une combinaison du paramétrage défini au niveau de chaque type de noisette et du paramétrage global de l'Ajax pour le plugin appelant ;

Chacun de ces paramètres est géré par une fonction d'API décrites au paragraphe 3.4. Chacune de ces fonctions gère un cache propre pour accélérer les traitements de compilation. Ces caches sont décrits au paragraphe 5.2.

Pour l'instant **N-Core ne gère pas la balise `<div>` englobante** : il convient au plugin appelant de s'en charger.

N-Core propose aussi un squelette nommé `compiler_noisettes.html` qui affiche une liste de noisettes fournies en argument en faisant donc appel à la balise `#NOISETTE_COMPILER` et propose une gestion de la balise englobante `<div>`. Ce squelette peut être utilisé si le besoin du plugin appelant est satisfait sinon il convient de créer son propre squelette sur ce modèle (c'est le cas du plugin noiZetier).

Ce squelette `compiler_noisettes.html` peut a minima servir d'exemple pour un plugin appelant, le code est donc présenté ci-dessous.

```
[(#REM) <!--  #COMPILER_NOISETTES

   Compile une liste de noisettes fournies en paramètres de l'inclusion.

   @api

   @param string plugin
          Identifiant qui permet de distinguer le module.

   @param array  noisettes
          Liste des descriptions de noisette à compiler dans l'ordre de leur rang.
   @param string stockage
          Identifiant du service de stockage à utiliser si précisé

-->]
<BOUCLE_compilation(DATA){source table, #ENV{noisettes, #ARRAY}}{plugin}{par rang noisette}>
   #SET{avec_div, #VALEUR{balise}|=={defaut}|?{oui, #VALEUR{balise}}}
   [[(#GET{avec_div}|oui)<div class="noisette noisette #NOISETTE[ (#VALEUR{css})]">]
      (#NOISETTE_COMPILER{#ENV{stockage, ''}})
   [(#GET{avec_div}|oui)</div>]]
</BOUCLE_compilation>
```

5. DONNEES DE N-CORE

5.1 La structure des données

5.1.1 Les types de noisette

Les données relatives aux types de noisettes proviennent principalement des fichiers YAML associés. La description d'un type de noisette est structurée dans N-Core dans un tableau associatif dont tous les champs possibles sont initialisés.

DESCRIPTION D'UN TYPE DE NOISETTE	
Données issues du fichier YAML	
type_noisette	Identifiant du type de noisette. Correspond au nom du fichier YAML sans extension.
nom	Titre du type de noisette sous forme textuelle ou d'un item de langue. Par défaut coïncide avec l'identifiant du type de noisette.
description	Texte ou item de langue décrivant le rôle du type de noisette.
icon	Nom du fichier d'icône représentant le type de noisette (sans chemin). Par défaut, prend la valeur `noisette-24.png`.

necessite	Liste des plugins – préfixes – nécessairement actifs pour utiliser le type de noisette. Ce champ est un tableau, éventuellement vide, de format « [] = préfixe ».
contexte	Liste des variables de contexte à fournir à la noisette lors de la compilation. Ce champ est un tableau, éventuellement vide, de format « [] = variable ». Les mots-clés « aucun » ou « env » peuvent être utilisés.
ajax	Indicateur d'inclusion en ajax de la noisette lors de la compilation. Prend les valeurs « defaut » (par défaut), « oui », « non ».
inclusion	Indicateur d'inclusion dynamique de la noisette lors de la compilation. Prend les valeurs « statique » (par défaut), « dynamique ».
parametres	Tableau, éventuellement vide, définissant le paramétrage du type de noisette et permettant la génération automatique du formulaire via l'API du plugin Saisies.
Données complémentaires	
plugin	L'identifiant du plugin utilisateur, à savoir, en général, son préfixe.
signature	md5 du fichier YAML calculé lors de son chargement.

Ces données sont initialisées par N-Core qui transmet la description au service de stockage. Il convient au plugin utilisateur de compléter ou pas cette description avant stockage via le service prévu à cet effet. Ces données ne sont jamais modifiées unitairement mais complètement lors du premier chargement ou d'un rechargement du fichier YAML.

5.1.2 Les noisettes

Les données relatives aux noisettes proviennent du type de noisette, de la localisation de son inclusion dans un conteneur et du paramétrage choisi pour la noisette. La description d'une noisette est structurée dans N-Core dans un tableau associatif dont tous les champs possibles sont initialisés.

DESCRIPTION D'UNE NOISETTE	
Données d'identification et de localisation	
id_conteneur	Identifiant unique du conteneur au format chaine de caractères.
rang_noisette	Position de la noisette dans la liste des noisettes incluses dans le même squelette. Ce champ est un entier supérieur ou égal à 1.
id_noisette	Identifiant unique de la noisette retourné lors de la création d'une nouvelle noisette. Ce champ est soit un entier (id d'une table SPIP) soit une chaine auquel cas il est calculé en utilisant la fonction PHP `uniqid()` avec le préfixe `${plugin}_` (cas de N-Core).
conteneur	Tableau associatif représentatif du conteneur éventuellement limité à l'index `squelette`. Sa composition dépend totalement du plugin utilisateur.

type_noisette	Identifiant du type de noisette. Correspond au nom du fichier YAML sans extension.
Données de paramétrage	
parametres	Tableau, éventuellement vide, définissant les valeurs des paramètres du type de noisette saisis dans le formulaire d'édition de la noisette.
balise	Indicateur d'inclusion de la noisette dans une balise `<div>` englobante. Prend les valeurs « defaut » (par défaut), « oui », « non ».
css	Styles CSS à affecter à la balise `<div>` englobante si celle-ci est requise pour la noisette.
Données complémentaires	
plugin	L'identifiant du plugin utilisateur, à savoir, en général, son préfixe.

Ces données sont initialisées par N-Core qui transmet la description au service de stockage. Il convient au plugin utilisateur de compléter ou pas cette description avant stockage via le service prévu à cet effet. Seules les données de paramétrage et le rang (en rouge) peuvent être modifiées unitairement après la création globale de la noisette. Les autres données sont, elles, statiques après création.

5.2 Les espaces de stockage obligatoires de N-Core

N-Core utilise plusieurs caches pour accéder rapidement à des données utiles à la compilation des noisettes. Ces caches ne peuvent pas être surchargés par le plugin appelant.

5.2.1 Les éléments de contexte des types de noisette

Ce cache se contente de consolider pour chaque type de noisette le tableau des éléments de contexte qui seront à minima fournis à la compilation. Cette information est configurée dans le fichier YAML caractérisant le type de noisette sous le tag `contexte:`.

Ce cache est géré par la fonction `noisette_contextualiser()`.

5.2.2 L'indicateur d'inclusion Ajax des types de noisette

Ce cache consolide pour chaque type de noisette l'indication d'inclusion de la noisette en Ajax ou pas. Cette information est tout d'abord configurée dans le fichier YAML caractérisant le type de noisette sous le tag `ajax:` et peut prendre les valeurs `defaut`, `oui` ou `non`. Si la valeur est `defaut`, une configuration globale permet de déterminer cette valeur par défaut. L'information dans le cache est un booléen qui indique si le type de noisette doit être utilisé en Ajax ou pas.

Ce cache est géré par la fonction `type_noisette_ajaxifier()`.

5.2.3 L'indicateur d'inclusion dynamique des types de noisette

Ce cache consolide pour chaque type de noisette l'indication du mode d'inclusion des noisettes de ce type, à savoir statique ou dynamique. Cette information est configurée dans le fichier YAML

caractérisant le type de noisette sous le tag `inclusion:`. L'information dans le cache est un booléen qui indique si le type de noisette doit être utilisé dynamiquement ou pas.

Ce cache est géré par la fonction `type_noisette_dynamiser()`.

5.3 Les espaces de stockage optionnels de N-Core

Pour simplifier le développement d'un plugin manipulant des noisettes, N-Core propose par défaut un espace de stockage pour les types de noisettes et un autre pour les noisettes. Les plugins appelant peuvent utiliser l'un ou l'autre ou les deux espaces proposés par N-Core ou développer leur propre espace de stockage comme le fait le plugin noiZetier.

5.3.1 Les types de noisette

N-Core stocke les descriptions des types de noisette telles que définies au paragraphe 5.1.1 dans un cache sécurisé, installé dans un sous-dossier `ncore/${plugin}/` de `_DIR_CACHE` et nommé `type_noisette_descriptions.php`.

Le cache contient le tableau sérialisé de tous les types de noisette détectés par N-Core ou le plugin utilisateur. La description complète est incluse dans le cache et la clé d'index est l'identifiant du type de noisette (qui est aussi inclus dans la description).

Pour optimiser certains traitements, N-Core utilise un autre cache sécurisé installé dans le même dossier et nommé `type_noisette_signatures.php`. Ce cache contient uniquement le tableau sérialisé des signatures des fichiers YAML indexé par l'identifiant du type de noisette. La signature est aussi présente dans le cache des descriptions.

Ces deux caches sont créés ou mis à jour simultanément lors de l'appel à la fonction d'API `type_noisette_charger()`.

5.3.2 Les noisettes

N-Core stocke les affectations de noisettes telles de définies au paragraphe 5.1.2 dans une meta nommée `${plugin}_noisettes`.

Cette meta contient le tableau sérialisé de toutes les noisettes affectées à divers conteneurs utilisés par le plugin utilisateur. Chaque affectation de noisette est un tableau indexé par identifiant de conteneur et par rang dans le conteneur. L'identification d'une noisette par le couple (identifiant de conteneur, rang) est donc optimale pour le stockage N-Core.

6. CONCEPTION DES API N-CORE

6.1 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 : **`<objet>_<verbe_infinitif>`**. Par exemple, la fonction de lecture de la

description d'un type de noisette se nomme `type_noisette_lire()` et la fonction d'ajout d'une noisette se nomme `noisette_ajouter()`.

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'une noisette se nomme `noisette_lire()`.

6.2 Arguments standardisés

Toutes les fonctions des API N-Core possèdent à minima deux arguments récurrents , à savoir, `$plugin` et `$stockage`.

L'argument **obligatoire** `$plugin` est toujours le **premier** argument du prototype des fonctions d'API. C'est une chaine de caractères qui **identifie le module utilisant la fonction** qui est dans tous les cas ou presque, un plugin à l'instar du noiZetier. Pour un plugin, l'utilisation du préfixe est recommandée. Cet argument est principalement utilisé pour distinguer les espaces de stockage d'un plugin utilisateur par rapport à d'autres.

L'argument **facultatif** `$stockage` est toujours le **dernier** argument du prototype des fonctions d'API. C'est une chaine de caractères qui est initialisée à vide si l'argument n'est pas fourni et qui identifie le **type de stockage à utiliser en priorité** indépendamment du plugin appelant `$plugin`. C'est normalement le préfixe du plugin fournissant le stockage souhaité. Cet argument permet la réutilisation des services d'un plugin utilisateur par un autre plugin utilisateur.

Les autres arguments dépendent de chaque fonction mais leur nommage est toujours le même d'une fonction à une autre.

Par exemple, l'argument `$information` désigne toujours un champ de la description d'une noisette ou d'un type de noisette.

L'argument `$type_noisette` désigne toujours l'identifiant d'un type de noisette qui coïncide avec le nom du fichier YAML sans extension.

De même `$id_noisette` et `$id_conteneur` représentent toujours respectivement l'identifiant unique d'une noisette ou d'un conteneur.

Par contre, l'argument `$noisette` s'il identifie bien de façon unique une noisette, peut revêtir deux formes : celle d'un *id* unique - i.e. `$id_noisette` - ou celle d'un couple *(identifiant de conteneur, rang)* – i.e. `$id_conteneur` et `$rang`. Cette souplesse permet d'optimiser l'adressage de la noisette en fonction du format de stockage. La conséquence est qu'il sera demandé aux fonction de services de supporter les deux adressages.

Enfin, l'argument `$conteneur` identifie le conteneur soit par son identifiant unique – i.e. `$id_conteneur` – soit de façon explicite par son tableau associatif.

7. APPLICATION POUR LE PLUGIN NOIZETIER

7.1 Le plugin

Le noiZetier permet d'associer des noisettes à des blocs de pages. Il offre une API de gestion des pages au sens SPIP (et de leurs compositions) et fournit une interface utilisateur complète pour remplir ces pages avec des noisettes.

La branche v3 est développée sur la base du plugin N-Core comme décrit dans la suite de ce chapitre.

7.2 Les espaces de stockages

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 les noisettes.

La structure des tables est composée des champs imposés par N-Core (voir le paragraphe 5.1) 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 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 concerné. C'est ainsi que le noiZetier stocke les éléments qui constitue le conteneur d'une noisette (en plus de l'id_conteneur).

7.3 La gestion des types de noisette

7.3.1 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', 'noisettes/')` 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 de l'interface privée du noiZetier.

Pour que cette API N-Core fonctionne avec l'espace de stockage du noiZetier, le noiZetier doit proposer les 3 fonctions de services suivantes :

- `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 listes des types de noisette ajoutés, modifiés et supprimés, fournies par N-Core.

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

```php
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;
}
```

```php
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.

7.3.2 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 la dernière 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;
```

7.3.3 Conclusion

Les 4 fonctions de services nécessaires pour faire fonctionner l'espace de stockage des types de noisette propre au noiZetier ont un code court 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. Néanmois, 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.

7.4 La gestion des conteneurs

7.4.1 Identifier un conteneur

Pour le noiZetier, un conteneur est un bloc d'une page (ou d'une composition) ou un bloc d'un objet SPIP donné. Cela revient donc toujours à un squelette, soit générique comme `content/article` ou `content/sommaire`, soit associé à un objet précis comme `content/article` de l'article d'id 12.

Comme on l'a vu au paragraphe 7.2, l'espace de stockage 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'id du conteneur sous une forme spécifique au plugin noiZetier, résultat de la fonction de service `noizetier_conteneur_identifier()`, seule fonction obligatoire quel que soit l'espace de stockage utilisé.

Le code de cette fonction est le suivant :

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

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

   if ($conteneur) {
      // Le nom du squelette en premier si il existe (normalement toujours).
      if (!empty($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|12`.

7.4.2 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 des noisettes d'un conteneur donné. Pour cela elle utilise l'unique API de N-Core `conteneur_vider()` en passant l'id du conteneur. Pour fonctionner cette API a besoin d'une fonction de service nommée `noizetier_conteneur_destocker()` dont le code est fournie ci-dessous.

```php
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.

7.4.3 Conclusion

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

7.5 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.

7.5.1 Ajouter une noisette à un conteneur

Le noiZetier propose dans la page de configuration des noisettes d'une page ou d'une composition ou d'un objet deux moyens d'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.

Pour ajouter la noisette dans ces deux cas, le noiZetier utilise l'API de gestion des noisettes de la façon suivante :

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

Pour que cette API N-Core fonctionne avec l'espace de stockage du noiZetier, le noiZetier 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 id 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 et c'est bien le cas pour le noiZetier.

7.5.2 Supprimer une noisette d'un conteneur

Le noiZetier fournit pour chaque noisette insérée dans un bloc de page, un bouton affublé uniquement d'un icone pour retirer une noisette du conteneur auquel elle appartient. Le noiZetier appelle ainsi la fonction d'API `noisette_supprimer()`.

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 absolument trivial car il se contente de faire appel à un `sql_fetsel()` pour la première et à un `sql_delete()` pour la seconde.

7.5.3 Déplacer une noisette dans un conteneur

Le noiZetier permet de déplacer par drag'n drop les noisettes d'un bloc d'une position à une autre. Le noiZetier appelle ainsi 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, `ncore_noisette_decrire()`, `ncore_noisette_lister()` et `ncore_noisette_ranger()`.

7.5.4 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).

7.6 La compilation des noisettes

7.7 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 un peu laborieux et répétitifs une série de filtres ont été définis spécifiquement pour le noiZetier.

8. APPLICATION POUR LE PLUGIN DASHBOARD

8.1 Le plugin

8.2 Les espaces de stockages

8.3 Les services