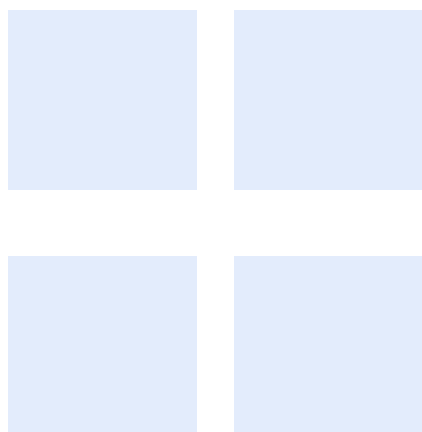
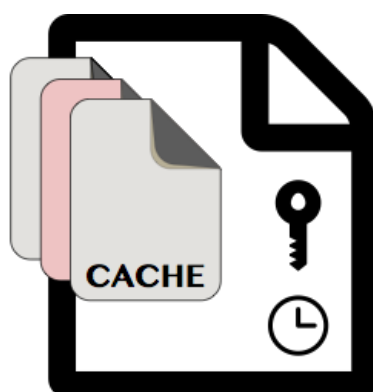


Référence SPIP/X/2019.001 – Ed. 1.4



## GUIDE CONCEPTION DU PLUGIN CACHE FACTORY



**FICHE D'IDENTIFICATION**

<b>Rédacteur</b>	Eric Lupinacci
<b>Projet</b>	SPIP
<b>Étude</b>	Conception du plugin Cache Factory
<b>Nature du document</b>	Guide
<b>Date</b>	08/02/2020
<b>Nom du fichier</b>	Guide - Le plugin Cache Factory.docx
<b>Référence</b>	SPIP/X/2019.001 – Ed. 1.4
<b>Dernière mise à jour</b>	08/02/2020 16:18:52
<b>Langue du document</b>	Français
<b>Nombre de pages</b>	22

## TABLE DES MATIERES

<b>1.</b>	<b>INTRODUCTION</b>	<b>5</b>
<b>2.</b>	<b>CONCEPTS</b>	<b>5</b>
<b>2.1</b>	<b>LES CACHES</b>	<b>5</b>
<b>2.2</b>	<b>LES CACHES « FICHIER »</b>	<b>5</b>
2.2.1	IDENTIFICATION COMPLETE D'UN FICHIER CACHE	5
2.2.2	IDENTIFICATION RELATIVE D'UN CACHE	6
2.2.3	LES ATTRIBUTS DES CACHES	6
<b>3.</b>	<b>PERIMETRE DU PLUGIN CACHE FACTORY</b>	<b>7</b>
<b>3.1</b>	<b>L'API DE GESTION DES CACHES</b>	<b>7</b>
<b>3.2</b>	<b>L'INTERFACE UTILISATEUR DE VIDAGE DES CACHES</b>	<b>8</b>
<b>4.</b>	<b>FONCTIONNEMENT DE CACHE FACTORY</b>	<b>9</b>
<b>4.1</b>	<b>LA DISSOCIATION API – SERVICES</b>	<b>9</b>
<b>4.2</b>	<b>L'AIGUILLAGE DES SERVICES</b>	<b>9</b>
<b>4.3</b>	<b>LES SERVICES</b>	<b>10</b>
<b>4.4</b>	<b>LE PIPELINE <code>POST_CACHE</code></b>	<b>12</b>
<b>5.</b>	<b>DONNEES DE CACHE FACTORY</b>	<b>13</b>
<b>5.1</b>	<b>LA CONFIGURATION GENERALE DES CACHES</b>	<b>13</b>
<b>5.2</b>	<b>L'ESPACE DE STOCKAGE DE LA CONFIGURATION GENERALE DES CACHES</b>	<b>14</b>
<b>6.</b>	<b>MISE EN PLACE D'UN PLUGIN UTILISATEUR</b>	<b>15</b>
<b>6.1</b>	<b>LA CONFIGURATION GENERALE</b>	<b>15</b>
<b>6.2</b>	<b>LE FORMULAIRE DE VIDAGE DES CACHES</b>	<b>15</b>
<b>6.3</b>	<b>L'IDENTIFICATION ET LA DESCRIPTION D'UN CACHE</b>	<b>16</b>
<b>6.4</b>	<b>UTILISATION DU PIPELINE <code>POST_CACHE</code></b>	<b>17</b>
<b>6.5</b>	<b>CONCLUSION</b>	<b>17</b>
<b>7.</b>	<b>REGLES DE CODAGE</b>	<b>19</b>
<b>7.1</b>	<b>NOMMAGE DES FONCTIONS</b>	<b>19</b>
<b>7.2</b>	<b>ARGUMENTS ET VARIABLES STANDARDISES</b>	<b>19</b>
<b>8.</b>	<b>GRAPHE D'APPEL API – SERVICES</b>	<b>20</b>



# 1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception du plugin Cache Factory (version 0.6.0 et ultérieures) dont l'objectif est de fournir des API génériques de gestion des caches.

Le plugin Cache Factory fournit également une interface utilisateur limitée à une page et un formulaire de vidage des caches pour chaque plugin utilisateur.

Dans cette version du plugin un cache est toujours un fichier.

## 2. CONCEPTS

### 2.1 Les caches

Les caches sont des espaces secondaires de stockage de données dont le but est de fournir un accès plus rapide que le stockage principal de ces mêmes données. Un mécanisme de mise à jour des caches doit être mise en œuvre si le stockage principal subit des modifications : les caches ont en général une durée de vie limitée dans le temps.

Un cache est soit un fichier soit un espace mémoire. Dans cette version du plugin Cache Factory les caches sont toujours des fichiers.

### 2.2 Les caches « fichier »

#### 2.2.1 Identification complète d'un fichier cache

Le plugin Cache Factory fournit une API qui permet de simplifier la manipulation (écriture, lecture, suppression, test d'existence...) de fichiers cache pour un plugin utilisateur. Un fichier est identifié de façon unique par un **chemin sur le disque**. Le chemin des caches de Cache Factory est toujours formaté selon la grammaire ABNF (approximative) suivante :

```
<chemin> = <repertoire-plugin> [<sous-dossier>] <nom-fichier> <extension-fichier>

<repertoire-plugin>    = <racine> <sous-dossier-plugin> "/"
<racine>                = ( _DIR_CACHE / _DIR_VAR / _DIR_TMP ) ; constante SPIP
<sous-dossier-plugin>  > = ["cache-"]<prefixe-plugin> ; suivant la racine
<prefixe-plugin>       = ALPHA *(ALPHA / DIGIT / "-" / "_")
<sous-dossier>         = 1*(ALPHA / DIGIT / "-" / "_") "/"

<nom_fichier>          = <composant> *(<separateur> <composant>)
<composant>            = 1*(ALPHA / DIGIT / "-" / "_") ; sauf le séparateur
<separateur>           = ("-" / "_")
<extension-fichier>    = "." 1*(ALPHA / DIGIT)

ALPHA                  = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT                  = %x30-39 ; 0-9
```

L'interprétation de la grammaire formelle permet d'édicter les règles suivantes :

- un fichier cache est toujours localisé dans un répertoire propre au plugin utilisateur qui est un sous-dossier des répertoires standards dans lesquels SPIP a déjà l'habitude de stocker ses propres caches et fichiers temporaires (`_DIR_CACHE`, `_DIR_VAR` et `_DIR_TMP`) ;
- l'identification du cache à l'intérieur du répertoire du plugin utilisateur est donné par un sous-dossier facultatif et un nom qui est composé d'une série ordonnées de composants sémantiques prédéfinis ;
- un composant du nom d'un fichier cache ne peut pas contenir le caractère séparateur utilisé.

Par exemple, les caches de N-Core sont stockés dans un dossier `tmp/cache/ncore/` et se nomment `<sous-dossier>/<objet>-<fonction>.php`. Cela donne pour le cache Ajax du noizetier : `tmp/cache/ncore/noizetier/type_noisette-ajax.php`. Le tiret est utilisé comme séparateur car le composant `<objet>` peut contenir le caractère souligné.

### 2.2.2 Identification relative d'un cache

Pour simplifier l'identification d'un cache pour les plugins utilisateur, Cache Factory ne demande pas à chaque appel d'une API tous les éléments d'identification décrits au paragraphe précédent. La racine, le séparateur et l'extension sont des éléments invariables du chemin du fichier cache pour un plugin utilisateur donné et sont stockés dans une configuration générale (voir le chapitre 5).

Ainsi, connaissant le plugin utilisateur - ce qui est le cas dans chaque fonction d'API - **l'identification relative** d'un cache est un tableau associatif fournissant le sous-dossier - si utilisé - et les composants, chaque composant portant une sémantique spécifique. Dans l'exemple du paragraphe précédent tiré du plugin N-Core, l'identifiant relatif coïncide avec le tableau suivant :

IDENTIFICATION RELATIVE DU CACHE AJAX DE N-CORE		
sous_dossier	Reçoit le préfixe du plugin utilisateur de N-Core	'noizetier'
objet	Identifie l'objet de N-Core concerné par le cache	'type_noisette'
fonction	Identifie le type de cache pour l'objet concerné, en l'occurrence dans notre cas, les indicateurs Ajax des types de noisette.	'ajax'

Un plugin utilisateur manipule de façon préférentielle l'identifiant relatif et rarement son chemin complet.

### 2.2.3 Les attributs des caches

Les caches fichier possèdent dans la version actuelle du plugin quelques attributs : la sécurisation du cache (au sens de l'API SPIP), la sérialisation du contenu, la durée de conservation du cache et l'indicateur de décodage. Ces attributs sont stockés dans la configuration générale.

## 3. PERIMETRE DU PLUGIN CACHE FACTORY

Cache Factory propose une API fonctionnelle PHP permettant de configurer, d'écrire, de lire et de supprimer des caches et une interface utilisateur dans l'espace privé limitée à un formulaire de vidage des caches.

### 3.1 L'API de gestion des caches

La gestion des caches consiste principalement à définir le nom des caches à partir de la configuration générale d'un plugin utilisateur et de l'identification relative, et de gérer les écritures et lectures de façon à simplifier au maximum les appels et traitements de l'appelant. Aucune donnée n'est enregistrée par cache, seule la configuration générale est stockée en meta (voir le chapitre 5).

#### API CACHES : INC/CACHE.PHP

##### Fonctions liées aux caches

<b>cache_ecrire</b>	Ecrit un contenu donné dans un cache spécifié par son identifiant relatif ou par son chemin complet. Appelle le pipeline <code>post_cache</code> en fin de traitement.
<b>cache_est_valide</b>	Teste l'existence sur le disque d'un cache spécifié par son identifiant relatif ou par son chemin complet et, si il existe, teste si la date d'expiration du fichier est dépassée ou pas. Si le fichier existe et n'est pas périmé, la fonction renvoie le chemin complet, sinon elle renvoie une chaîne vide.
<b>cache_lire</b>	Lit le cache spécifié par son identifiant relatif ou son chemin complet et renvoie le contenu sous forme de tableau ou de chaîne suivant l'attribut de sérialisation ou de décodage. En cas d'erreur, la fonction renvoie <code>false</code> .
<b>cache_nommer</b>	Renvoie le chemin complet du cache sans tester son existence. Cette fonction est une encapsulation du service <code>cache_cache_composer()</code> . Son utilisation par un plugin appelant doit rester limitée.
<b>cache_repertorier</b>	Retourne la description des caches d'un plugin utilisateur filtrés sur un ensemble de critères. La description de base fournie par Cache Factory contient les éléments de l'identifiant relatif mais peut-être remplacée ou complétée par le plugin appelant au travers de services propres. Les filtres concernent uniquement les éléments de l'identifiant relatif.
<b>cache_supprimer</b>	Supprime le cache spécifié par son identifiant relatif ou par son chemin complet. Appelle le pipeline <code>post_cache</code> en fin de traitement.

<b>cache_vider</b>	Supprime, pour un plugin utilisateur donné, les caches désignés par leur chemin complet. Appelle le pipeline <code>post_cache</code> à chaque suppression.
<b>Fonctions liées à la configuration</b>	
<b>configuration_cache_lire</b>	Lit la configuration générale des caches d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistré une configuration.
<b>configuration_cache_effacer</b>	Efface la configuration générale des caches d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistré une configuration.

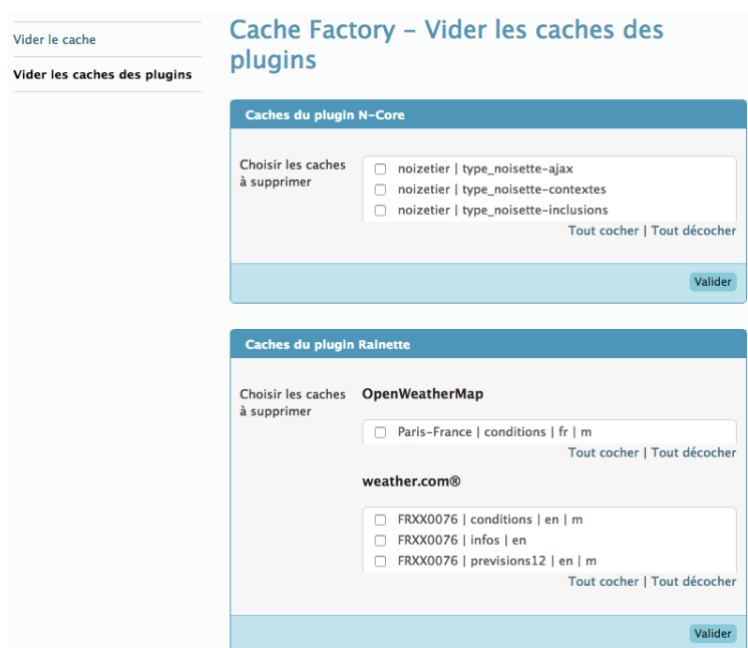
## 3.2 L'interface utilisateur de vidage des caches

L'interface utilisateur de vidage des caches est composée d'une page dans l'espace privé, `cache_vider`, et d'un formulaire listant les caches d'un plugin utilisateur donné, `#FORMULAIRE_CACHE_VIDER`.

La page `cache_vider` affiche les formulaires de vidage de chaque plugin utilisateur les uns sous les autres comme illustré ci-après.

Le formulaire de vidage possède un paramètre obligatoire, l'identifiant du plugin utilisateur. Le prototype est le suivant : `#FORMULAIRE_CACHE_VIDER{plugin[, options]}`.

Chaque formulaire est personnalisable par le plugin utilisateur en surchargeant le service de chargement des paramètres du formulaire et le sous-squelette d'affichage de la liste caches. Cette personnalisation est discutée au chapitre suivant.





## 4. FONCTIONNEMENT DE CACHE FACTORY

### 4.1 La dissociation API – Services

Le fonctionnement global du plugin Cache Factory est similaire à celui du plugin N-Core : il utilise la même architecture API - services.

De façon générale, un plugin utilisateur comme N-Core va s'appuyer sur l'ensemble des API publiques de Cache Factory (gestion des caches, configuration et interface utilisateur). De fait, il doit définir la configuration qu'il souhaite pour ses caches. Par conception, **Cache Factory dissocie la fonction d'API, des services propres à un plugin utilisateur qui en personnalisent la gestion.**

Dans le code de ses API, Cache Factory appelle des fonctions de service qui :

- si la fonction de service homonyme existe dans le plugin utilisateur, va l'appeler et l'utiliser ;
- sinon, va dérouler la fonction de Cache Factory.

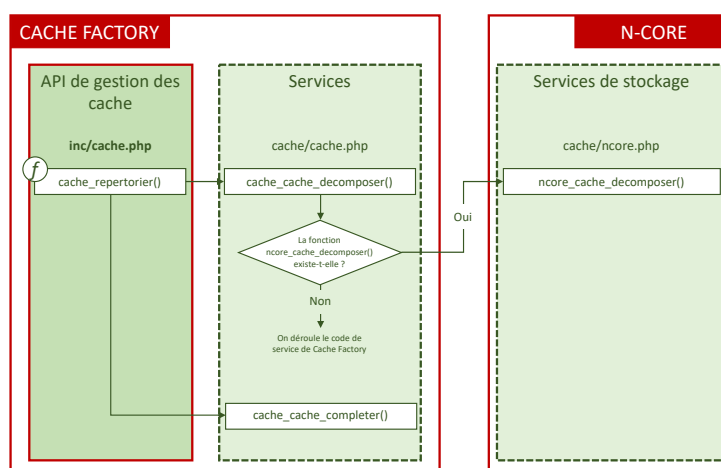
Toute fonction d'API de Cache Factory possède un argument obligatoire, `$plugin`, comme on peut le voir sur le prototype de `cache_ecrire()` :

```
function cache_ecrire($plugin, $cache, $contenu)
```

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

### 4.2 L'aiguillage des services

Par conception et pour des raisons de lisibilité du code, les fonctions d'API de Cache Factory appellent systématiquement les fonctions de service de Cache Factory. Ce sont les **fonctions de service de Cache Factory qui réalisent l'aiguillage vers le service souhaité** ce qui leur permet aussi d'effectuer des traitements génériques et donc de limiter encore plus la complexité pour les plugins utilisateur.



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

```
cache_repertorier('ncore');
```

La fonction `cache_repertorier()` de l'API fait appel à une fonction de service de Cache factory, `cache_cache_decomposer()`, qui renvoie la description des éléments constitutifs de l'identifiant de chaque cache répertorié. Cette fonction de service va déterminer quelle fonction dérouler, celle de N-Core ou son propre code. Pour cela, elle appelle une fonction utilitaire `cache_service_chercher()` qui lui retourne le nom de la fonction de service ou vide si aucune fonction n'est définie dans le plugin appelant :

```
if ($decomposer = cache_service_chercher($plugin, 'cache_decomposer')) {  
    $cache = $decomposer($plugin, $fichier_cache, $configuration);  
}
```

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

```
function cache_service_chercher($plugin, $fonction) {  
    $fonction_trouvee = '';  
  
    // Eviter la réentrance si on demande explicitement le service du plugin Cache Factory.  
    if ($plugin != 'cache') {  
        include sip("cache/${plugin}");  
        $fonction_trouvee = "${plugin} ${fonction}";  
        if (!function_exists($fonction_trouvee)) {  
            $fonction_trouvee = '';  
        }  
    }  
  
    return $fonction_trouvee;  
}
```

## 4.3 Les services

Les fonctions d'API de Cache Factory font donc appel à des services dont la liste exacte est fournie ci-après. Le nom des fonctions est amputé du préfixe du plugin appelant.

Ces **fonctions de service ne doivent pas être appelées par le plugin appelant** qui doit utiliser exclusivement les fonctions d'API. Le plugin appelant peut définir ses propres services qui seront appelés par ceux de Cache Factory, mais en aucun cas les utiliser dans son code.

## SERVICES

### Services de configuration

#### cache\_configurer (\*)

Renvoie la configuration générale des caches d'un plugin utilisateur sous la forme d'un tableau associatif dont les index sont standardisés.

### Services de gestion des identifiants de cache

#### cache\_composer

Construit le chemin complet du fichier cache à partir du tableau de l'identifiant relatif du cache.

#### cache\_decomposer

Décompose le chemin complet du fichier cache en éléments constitutifs. Par défaut, le tableau obtenu coïncide avec l'identifiant relatif du cache. La fonction utilise la configuration générale pour connaître la structure du chemin du fichier.

#### cache\_completer

Complète la description canonique d'un cache issue du service `cache_decomposer()`. Le plugin Cache Factory complète la description avec le nom sans extension et l'extension du fichier.

### Services de gestion du contenu d'un cache

#### cache\_decoder

Effectue le décodage du contenu brut du cache (chaîne) en fonction de l'extension du fichier cache. La fonction propose des traitements standard pour le JSON, YAML, et XML. Tout plugin utilisateur peut proposer sa fonction spécifique pour le format `$format` en codant le service `cache_decoder_${format}`.  
**Il n'existe pas de service d'encodage inverse. C'est au plugin utilisateur d'encoder son contenu avant l'écriture du cache.**

### Services de gestion du formulaire de vidage

#### cache\_formulaire\_charger

Effectue le chargement du formulaire de vidage des caches pour un plugin utilisateur donné. Par défaut, le plugin Cache Factory propose une version simplifiée du formulaire où tous les fichiers caches sont listés par ordre alphabétique sans possibilité de regroupement.

Les services notés (\*) doivent toujours être définis par le plugin utilisateur, les autres sont optionnels.

Cache Factory propose l'ensemble des services dans son fichier `cache/cache.php` ce qui permet de minimiser les développements pour la plupart des plugins utilisateur. A minima, un plugin utilisateur peut se contenter de décrire uniquement sa configuration générale s'il n'a aucune spécificité par rapport aux services natifs de Cache Factory.

Le plugin Rainette définit la configuration générale de ses caches et personnalise le chargement du formulaire de vidage des caches en codant dans le fichier `cache/rainette.php` les fonctions `rainette_cache_configurer()` et `rainette_cache_formulaire_charger()`.

## 4.4 Le pipeline `post_cache`

Suite à l'écriture d'une cache - fonction `cache_ecrire()` – ou à sa suppression – fonctions `cache_supprimer()` et `cache_vider()` – Cache Factory fait appel à un pipeline nommé `post_cache`. Ce pipeline n'a pas pour but de modifier des données mais permet de lancer un traitement éventuel pour conclure l'opération d'écriture ou de suppression en cours.

De fait, le pipeline fournit quelques informations en entrée regrouper dans l'index 'args' du flux. Les éléments fournis sont :

### ELEMENTS DE L'INDEX ARGS DU PIPELINE

<b>plugin</b>	Le préfixe du plugin utilisateur de Cache Factory. Permet à un plugin utilisateur P1 d'éviter de lancer ses traitements initiés par un plugin P2
<b>fonction</b>	Identifie la fonction générique faisant appel au pipeline. Prend les valeurs 'ecrire' ou 'supprimer'.
<b>fichier_cache</b>	Chemin complet du fichier cache.
<b>cache</b>	Tableau canonique identifiant du cache.
<b>configuration</b>	Tableau de configuration des caches du plugin utilisateur.

Le pipeline peut être utilisé pour tenir à jour un index des fichiers cache comme pour le plugin REST Factory (voir paragraphe 6.4).

## 5. DONNEES DE CACHE FACTORY

### 5.1 La configuration générale des caches

La configuration générale des caches d'un plugin est décrite ci-dessous. Seul le premier groupe de paramètres est à fournir par le plugin utilisateur. Il est possible pour le plugin utilisateur d'omettre les paramètres dont les valeurs par défaut coïncident avec les besoins du plugin. Les valeurs par défaut sont fournies dans la colonne de droite.

CONFIGURATION GENERALE DES CACHES		
Paramètres pouvant être fournis par le plugin utilisateur		
<b>racine</b>	Emplacement de base dans lequel sera créé le répertoire de stockage des caches du plugin utilisateur. Les valeurs possibles sont ' <code>_DIR_CACHE</code> ', ' <code>_DIR_VAR</code> ' et ' <code>_DIR_TMP</code> '.	<code>'_DIR_CACHE'</code>
<b>sous_dossier</b>	Indique si le cache est inclus dans un sous-dossier du répertoire de stockage du plugin.	<code>false</code>
<b>nom_obligatoire</b>	Tableau des composants obligatoires et ordonnés du nom d'un cache. Le nom des composants sert d'index quand il s'agit de fournir le tableau identifiant un cache.	<code>array('nom')</code>
<b>nom_facultatif</b>	Tableau des composants facultatifs et ordonnés du nom d'un cache. Les composants facultatifs suivent toujours les composants obligatoires.	<code>array()</code>
<b>separateur</b>	Caractère de séparation de chaque composant obligatoire ou facultatif du nom. Prend les valeurs ' <code>_</code> ', ' <code>-</code> ' ou ' <code> </code> ' si le nom ne comporte qu'un seul composant. Ce caractère ne doit pas être utilisé dans les composants du nom.	<code>' '</code>
<b>extension</b>	Extension du fichier cache. Si le fichier cache est sécurisé l'extension est toujours forcée à ' <code>.php</code> '.	<code>'.txt'</code>
<b>securisation</b>	Indique si le fichier doit être sécurisé ou pas.	<code>false</code>
<b>serialisation</b>	Indique si le contenu à stocker est un tableau à sérialiser ou pas.	<code>true</code>
<b>decodage</b>	Indique si le contenu lu est à décoder ou pas avant fourniture à l'appelant. Le type de décodage est déterminé par l'extension du fichier cache. Cette option est incompatible avec la sérialisation.	<code>false</code>
<b>conservation</b>	Durée de conservation du cache exprimée en secondes. La valeur 0 est utilisée pour indiquer que le cache est permanent.	<code>0</code>

### Paramètres calculées par Cache Factory

dossier_plugin	Répertoire de stockage du plugin calculé à partir de la racine et du préfixe du plugin. Si la racine prend la valeur ' <code>_DIR_VAR</code> ' alors le répertoire du plugin est un sous-dossier de nom <code>cache-<code>{plugin}</code></code> /. Sinon, le sous-dossier porte le nom du préfixe du plugin.	<code>"<code>{plugin}</code>/"</code>
nom	Tableau regroupant dans l'ordre les noms obligatoires puis les noms facultatifs.	<code>array('nom')</code>

## 5.2 L'espace de stockage de la configuration générale des caches

Le seul espace de stockage utilisé par Cache Factory est celui nécessaire à la conservation de la configuration générale des plugins utilisateur. Pour cela, Cache Factory utilise une meta nommée `cache`. Le contenu de la meta est un tableau de chaque configuration générale (voir la structure au paragraphe précédent) indexé par le préfixe du plugin.

## 6. MISE EN PLACE D'UN PLUGIN UTILISATEUR

### 6.1 La configuration générale

Cache Factory a besoin de connaître la configuration générale des caches d'un plugin utilisateur pour fonctionner. Le **plugin utilisateur doit obligatoirement déclarer sa configuration** par l'intermédiaire du service `cache_configurer()`.

Le tableau associatif de la configuration à fournir par un plugin utilisateur a une structure prédéfinie qui est décrite au paragraphe 5.1. Le plugin utilisateur peut fournir tout ou partie de cette configuration sachant que les index non fournis seront complétés par Cache Factory avec une valeur par défaut. Les valeurs par défaut sont consultables au paragraphe 5.1.

Par exemple, le plugin N-Core déclare la configuration suivante :

```
$configuration = array(
    'racine' => '_DIR_CACHE',
    'sous_dossier' => true,
    'nom_obligatoire' => array('objet', 'fonction'),
    'nom_facultatif' => array(),
    'extension' => '.php',
    'securisation' => true,
    'serialisation' => true,
    'encodage' => false,
    'separateur' => '-',
    'conservation' => 0
);
```

On note l'utilisation d'un sous-dossier, les deux composants obligatoires du nom et le tiret séparateur comme déjà discuté au paragraphe 2.2.1. N-Core pourrait se passer de fournir certains index comme les composants facultatifs, la racine, l'attribut de sérialisation ou la durée de conservation car ils correspondent aux valeurs par défaut.

### 6.2 Le formulaire de vidage des caches

Cache Factory permet de personnaliser la présentation des caches d'un plugin utilisateur dans le formulaire de vidage.

En effet, par défaut, Cache Factory affiche une liste unique de tous les caches du plugin utilisateur avec le sous-dossier, si il existe, et le nom du cache sans son extension. Les caches sont classés alphabétiquement. Cette présentation par défaut suffit, par exemple, au plugin N-Core mais pas à Rainette qui préfère présenter ses caches regroupés par service.

Pour permettre à Rainette et à d'autres plugins de personnaliser l'affichage, la portion du squelette qui réalise l'affichage de la liste est codée dans une inclusion. Par défaut, l'inclusion fournie par Cache Factory se nomme `formulaires/inc-cache_cache_vider.html`. Un plugin qui souhaite « remplacer » cette inclusion doit créer sa propre inclusion nommée `formulaires/inc-{$plugin}_cache_vider.html`. Si elle existe elle sera appelée en lieu et place de celle par défaut.

Le code du formulaire `formulaires/cache_vider.html` contient donc les instructions suivantes :

```
[ (#CHEMIN{[formulaires/inc- (#ENV{ _prefixe }) _cache_vider.html]}|oui)
  <INCLUDE {fond=formulaires/inc-#ENV{ _prefixe } cache vider, env} />
]
[ (#CHEMIN{[formulaires/inc- (#ENV{ _prefixe }) _cache_vider.html]}|non)
  <INCLUDE {fond=formulaires/inc-cache_cache_vider, env} />
]
```

En général, pour produire un affichage différent de l’affichage par défaut, un plugin regroupe ses caches selon certains critères et/ou rajoute des données pour chaque cache. Il est souvent nécessaire de modifier le tableau de chargement du formulaire pour qu’il coïncide avec l’objectif d’affichage.

Pour ce faire, Cache Factory propose le service `cache_formulaire_charger()`. Par exemple, le plugin Rainette modifie le chargement du formulaire avec le code contenu dans la fonction `rainette_cache_formulaire_charger()` et présenté ci-dessous :

```
function rainette_cache_formulaire_charger($plugin, $configuration) {

    $valeurs = array();

    // On constitue la liste des services requis par l'appel
    include_spip('rainette_fonctions');
    $services = rainette_lister_services();

    // On récupère les caches et leur description pour donner un maximum d'explication sur le
    contenu.
    include_spip('inc/cache');
    foreach ($services as $service => $titre) {
        // On récupère les caches du service
        $filtres = array('sous_dossier' => $service);
        $caches = cache_repertorier('rainette', $filtres);

        // Si il existe des caches pour le service on stocke les informations recueillies
        if ($caches) {
            $valeurs['_caches'][$_service]['titre_service'] = $_titre;
            $valeurs['_caches'][$_service]['caches'] = $caches;
        }
    }

    return $valeurs;
}
```

## 6.3 L’identification et la description d’un cache

Cache Factory possède des mécanismes par défaut pour composer le nom d’un fichier cache et inversement pour le décomposer en composants unitaires. Ces mécanismes sont simples et non contextuels.

Par exemple, pour constituer le nom d’un fichier cache le service natif `cache_composer()` lit les composants attendus (obligatoires et facultatifs) et les concatène en utilisant le séparateur configuré. Pour retrouver les composants à partir du nom du fichier cache, le service natif `cache_decomposer()` fait l’inverse en scindant le nom en composants. Les composants sont interprétés dans l’ordre donné par le paramètre de configuration `nom` (voir paragraphe 5.1) sans aucune vérification sémantique.

Ces mécanismes sont en général suffisants pour la plupart des plugins. Néanmoins, pour les plugins utilisateur qui souhaiteraient personnaliser la composition ou la décomposition du nom du fichier cache, Cache Factory leur offre la possibilité de fournir leur propres services en codant les fonctions



`$_{plugin}_cache_composer()` et `$_{plugin}_cache_decomposer()` qui seront utilisées en lieu et place des services natifs.

Cache Factory propose également de compléter la description d'un cache renvoyée par l'API `cache_repertorier()`. La base de cette description est constituée du tableau fournie par le service `cache_decomposer()` auquel Cache Factory rajoute les index 'nom\_cache' et 'extension\_cache' au travers de son service natif `cache_completer()`.

De même, ce mécanisme est souvent suffisant mais un plugin comme Taxonomie a besoin de rajouter le nom scientifique du taxon concerné par un cache pour l'affichage dans le formulaire. Il code donc son propre service `taxonomie_cache_completer()` pour rajouter cette information à la description retournée par `cache_repertorier()`.

## 6.4 Utilisation du pipeline `post_cache`

L'utilisation du pipeline `post_cache` est assez rare car le plugin Cache Factory fournit par défaut déjà de nombreuses possibilités de personnalisation, en particulier concernant l'identifiant du cache. Néanmoins, certains cas sont difficilement résolubles avec les fonctions de gestion de l'identifiant car un ou plusieurs composants sont eux-mêmes calculés : il devient alors impossible de trouver une fonction réversible pour composer et décomposer le nom.

Le plugin REST Factory est confronté à ce problème : les filtres ou les ressources peuvent contenir des caractères inappropriés pour la constitution d'un nom de fichier. Pour éviter ce problème sans utiliser d'improbable conversion, ce composant nommé 'complement' est calculé en cryptant sa source avec un md5. Il est donc impossible de décrypter le contenu de ce complément si on veut afficher les informations sous-jacentes dans le formulaire de vidage des caches.

C'est pourquoi REST Factory utilise un index des caches créés où il stocke les éléments source ayant permis d'élaborer le nom. Il utilise cet index lors de l'affichage du formulaire de vidage des caches pour afficher les filtres ou la ressource de façon claire.

Pour ce faire, le pipeline est utilisé comme suit lors de l'écriture et de la suppression d'un cache, fonction `ezrest_post_cache()`. Il est essentiel de tester que le plugin initiant le pipeline est bien le plugin même plugin utilisateur car sinon on déclencherait un traitement inadéquat.

```
// On vérifie que l'appel du pipeline est bien lié à une action sur les caches de REST Factory
if ($flux['args']['plugin'] == 'ezrest') {
    // Identification du fichier d'index des caches
    $configuration = $flux['args']['configuration'];
    $fichier_index = constant($configuration['racine']) . $configuration['dossier_plugin'] .
    'index.txt';

    // Lecture du fichier d'index et récupération du tableau des caches.
    lire_fichier($fichier_index, $contenu_index);
    $index = $contenu_index ? unserialize($contenu_index) : array();

    // Extraction du fichier cache : on utilise juste le nom et le répertoire du plugin ce qui
    suffit pour être unique.
    $fichier_cache = basename(dirname($flux['args']['fichier_cache'])) . '/' .
    basename($flux['args']['fichier_cache']);

    if ($flux['args']['fonction'] == 'ecrire') {
        // On vient d'écrire un cache, on le loge dans l'index.
        $index[$fichier_cache] = $flux['args']['cache'];
    }
}
```

```

} elseif ($flux['args']['fonction'] == 'supprimer') {
    // On vient de supprimer un cache, on le retire de l'index.
    if ($index and isset($index[$fichier_cache])) {
        unset($index[$fichier_cache]);
    }
}

// Mise à jour de l'index
ecrire_fichier($fichier_index, serialize($index));
}

```

## 6.5 Conclusion

Pour un plugin utilisateur, la mise en œuvre de Cache Factory peut se limiter à :

- créer à la racine du plugin un fichier `cache/${plugin}.php` ;
- fournir un tableau de configuration générale en codant la fonction `${plugin}_cache_configurer()` ;
- et utiliser les API proposées.

Un exemple classique d'utilisation des API est donnée ci-dessous (plugin Spiper Ipsum) :

```

if (!$fichier_cache = cache_est_valide('spiperipsum', $cache))
or (defined('_SPIPERIPSUM_FORCER_CHARGEMENT') ? _SPIPERIPSUM_FORCER_CHARGEMENT : false) {
    // Utilisation de la fonction de chargement du service.
    $charger = "${service} charger";
    $tableau = $charger($code_langue, $date);

    // Mise à jour du cache
    cache_ecrire('spiperipsum', $cache, $tableau);
} else {
    // Lecture des données du fichier cache valide
    $tableau = cache_lire('spiperipsum', $fichier_cache);
}

```

Certains plugins vont vouloir personnaliser l'affichage du formulaire de vidage en codant leur version du service `cache_formulaire_charger()` et de l'inclusion `formulaires/inc-${plugin}_cache_vider.html`.

Ces deux personnalisations sont en général largement suffisantes pour 90% des plugins utilisateurs.

## 7. REGLES DE CODAGE

### 7.1 Nommage des fonctions

Le nommage des fonctions appartenant à l'API de Cache Factory 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 d'un cache se nomme `cache_lire()`.

### 7.2 Arguments et variables standardisés

Toutes les fonctions des API Cache Factory possèdent à minima l'argument `$plugin`.

L'argument **obligatoire** `$plugin` est toujours le **premier** argument du prototype des fonctions d'API. C'est une chaîne de caractères qui **identifie le module utilisant la fonction** qui est dans tous les cas ou presque, un plugin à l'instar de N-Core. Pour un plugin, l'utilisation du préfixe est recommandée.

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

Par exemple, l'argument `$cache` désigne toujours l'identifiant du cache, qu'il soit relatif (sous forme de tableau) ou complet (chaîne représentant le chemin). Si l'argument est une liste de cache la variable est écrite au pluriel, `$caches`.

De même `$contenu` désigne toujours le contenu du cache et `$configuration` la configuration générale d'un plugin utilisateur.

Les variables locales utilisées dans le code des fonctions du plugin sont aussi normalisées tant que faire se peut.

C'est le cas de `$contenu_cache` qui désigne le contenu du fichier, de `$fichier_cache` qui représente toujours le chemin complet du cache et de `$nom_cache` qui identifie uniquement le nom avec ou sans extension du fichier (basename).

## 8. GRAPHE D'APPEL API – SERVICES

API – SERVICE	
<i>Fonctions liées aux caches</i>	
<b>cache_ecrire</b>	configuration_cache_lire cache_cache_composer pipeline post_cache
<b>cache_est_valide</b>	configuration_cache_lire cache_cache_composer
<b>cache_lire</b>	configuration_cache_lire cache_cache_composer cache_cache_decoder
<b>cache_nommer</b>	configuration_cache_lire cache_cache_composer
<b>cache_repertorier</b>	configuration_cache_lire cache_cache_decomposer cache_cache_completer
<b>cache_supprimer</b>	configuration_cache_lire cache_cache_composer pipeline post_cache
<b>cache_vider</b>	pipeline post_cache
<i>Fonctions liées à la configuration</i>	
<b>configuration_cache_effacer</b>	
<b>configuration_cache_lire</b>	cache_cache_configurer

## 9. PROTOTYPES DES API

---

*boolean* **cache\_ecrire**( *string* \$plugin , array | *string* \$cache , array | *string* \$contenu )

*Écrit un contenu donné dans un cache spécifié par son identifiant relatif ou par son chemin complet.*

---

*string* **cache\_est\_valide**( *string* \$plugin , array | *string* \$cache )

*Teste l'existence d'un cache sur le disque spécifié par son identifiant relatif ou par son chemin complet et, si il existe, teste si la date d'expiration du fichier n'est pas dépassée. Si le fichier existe et n'est pas périmé, la fonction renvoie le chemin complet, sinon elle renvoie une chaine vide.*

---

array | *string* | *boolean* **cache\_lire**( *string* \$plugin , array | *string* \$cache )

*Lit le cache spécifié par son identifiant relatif ou son chemin complet et renvoie le contenu sous forme de tableau ou de chaine suivant l'attribut de sérialisation ou de décodage. En cas d'erreur, la fonction renvoie false.*

---

*string* **cache\_nommer**( *string* \$plugin , array \$cache )

*Renvoie le chemin complet du cache sans tester son existence. Cette fonction est une encapsulation du service cache\_cache\_composer().*

---

array **cache\_repertorier**( *string* \$plugin , array \$filtres = array() )

*Retourne la description des caches d'un plugin utilisateur filtrée sur un ensemble de critères. La description de base fournie par Cache Factory contient les éléments de l'identifiant relatif mais peut-être remplacée ou complétée par le plugin appelant au travers de services propres. Les filtres concernent uniquement les éléments de l'identifiant relatif.*

---

*boolean* **cache\_supprimer**( *string* \$plugin , array | *string* \$cache )

*Supprime le cache spécifié par son identifiant relatif ou par son chemin complet.*

---

*boolean* **cache\_vider**( *string* \$plugin , array \$caches )

*Supprime, pour un plugin donné, les caches désignés par leur chemin complet.*

---

*boolean* **configuration\_cache\_effacer**( *string* \$plugin = " )

*Efface la configuration standard des caches d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistrés une configuration.*

---

*boolean* **configuration\_cache\_lire**( *string* \$plugin = " )

*Lit la configuration standard des caches d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistrés une configuration.*