Référence SPIP/X/2019.001 – Ed. 2.1

GUIDE
CONCEPTION DU PLUGIN CACHE FACTORY

FICHE D'IDENTIFICATION	
Rédacteur	Eric Lupinacci
Projet	SPIP
Étude	Conception du plugin Cache Factory
Nature du document	Guide
Date	19/03/2022
Nom du fichier	Guide - Le plugin Cache Factory.docx
Référence	SPIP/X/2019.001 – Ed. 2.1
Dernière mise à jour	20/03/2022 11:25:36
Langue du document	Français
Nombre de pages	26

TABLE DES MATIERES

1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception du plugin Cache Factory (version 1.2.0 et ultérieures) dont l'objectif est de fournir des API génériques de gestion de caches. Cette branche v1 est incompatible avec la branche v0 qui n'est plus maintenue.

Le plugin Cache Factory fournit également une interface utilisateur minimale pour le vidage des caches de chaque plugin utilisateur et l'affichage de la liste des caches.

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/ _DIR_ETC) ; const. SPIP
<sous-dossier-plugin >  = ["cache-"]<prefixe-plugin>          ; suivant la racine
<prefixe-plugin>        = ALPHA *(ALPHA / DIGIT / "_")

<sous-dossier>          = 1*(ALPHA / DIGIT / "-" / "_") "/"

<nom_fichier>           = [<prefixe-nom> <separateur>] <nom>
<nom>                   = <composant> *(<separateur> <composant>)
<prefixe-nom>           = 1*(ALPHA / DIGIT / "-" / "_")       ; sauf le séparateur
<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'énoncer 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, _DIR_ETC 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 et pouvant commencer par un préfixe fixe ;
- 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, **l'identification relative** d'un cache est un tableau associatif fournissant le sous-dossier - si utilisé et si différent du type de cache - et les composants du nom, 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. On comprend mieux l'intérêt de l'identifiant relatif du cache car le nommage représente en fait qu'une petite partie de la configuration des caches.

2.2.4 Type de cache et configuration

Le modèle de configuration des caches et la structure en composants sémantiques pour le nommage autorisent beaucoup de flexibilité pour regrouper les caches d'un plugin sous une configuration unique. Néanmoins, si un plugin utilisateur a besoin de caches JSON et PHP, ou de caches avec et sans péremption le plus efficace est de définir différentes configuration de caches.

Pour distinguer chaque configuration et associer une configuration donnée à un cache, Cache Factory définit le concept de **type de cache**. Un type de cache possède un identifiant unique au sein d'un plugin utilisateur composée d'une chaine alphanumérique.

C'est le couple (plugin utilisateur, type de cache) qui permet d'identifier de manière unique la configuration à appliquer à un cache. Cela a donc une incidence immédiate sur l'API de Cache Factory. Cette notion est la grande nouveauté de la branche v1 et elle introduit une incompatibilité avec la branche v0 dans laquelle la connaissance seule du plugin utilisateur permettait d'identifier la configuration à appliquer (unique pour un plugin utilisateur).

Si le sous-dossier est utilisé par le type de cache, par défaut, l'identifiant du type de cache sera affecté au sous-dossier si l'utilisateur ne le fournit pas dans l'identifiant relatif du cache. Cette évolution fonctionnelle est disponible à partir de la version 1.2.0.

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 et à une liste 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).

Toutes les fonctions d'API possède deux arguments obligatoires, à savoir, le préfixe du plugin utilisateur et le type de cache. Le fait que le couple (plugin utilisateur, type de cache) soit connu pour chaque API n'est pas rappelé dans les explications.

API CACHES : INC/EZCACHE_CACHE.PHP	
Fonctions liées aux caches	
cache_ecrire	Ecrit un contenu donné dans un cache spécifié par son identifiant relatif ou par son chemin complet. Appelle le pipeline `post_cache` en fin de traitement sauf si explicitement interdit.
cache_est_valide	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, n'est pas périmé et répond éventuellement à des critères de validation spécifiques, la fonction renvoie le chemin complet, sinon elle renvoie une chaine vide.
cache_lire	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`.
cache_nommer	Renvoie le chemin complet du cache sans tester son existence. Cette fonction est une encapsulation du service `cache_cache_composer()`. Son utilisation par un plugin appelant doit rester limitée.
cache_repertorier	Retourne la description des caches d'un type de cache 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.

cache_supprimer	Supprime le cache spécifié par son identifiant relatif ou par son chemin complet. Appelle le pipeline `post_cache` en fin de traitement sauf si explicitement interdit.
cache_vider	Supprime, pour un type de cache d'un plugin utilisateur donné, les caches désignés par leur chemin complet. Appelle le pipeline `post_cache` à chaque suppression sauf si explicitement interdit.
Fonctions liées à la configuration	
configuration_cache_lire	Lit la configuration générale d'un type de caches d'un plugin utilisateur, de tous les types de cache d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistrés une configuration.
configuration_cache_effacer	Efface la configuration générale de tous les types de caches d'un plugin utilisateur ou de tous les plugins utilisateur ayant enregistrés une configuration.

3.2 L'interface utilisateur

3.2.1 Liste des caches

Cache Factory propose une balise `#CACHE_LISTE` qui fournit la liste des caches pour un plugin utilisateur donné, conformément aux filtres éventuels. Cette balise est une encapsulation de la fonction d'API `cache_repertorier()`. Il est possible de filtrer les caches d'un type donné en ajoutant le critère `type_cache` et sa valeur dans la liste des filtres.

Le prototype de la balise est le suivant : `#CACHE_LISTE{plugin[, filtres]}`.

Le plugin fournit un squelette d'affichage de la liste sous une forme tabulaire classique des objets SPIP. Ce squelette se nomme `prive/squelettes/liste/caches.html`. La boucle utilisée est une boucle DATA basée sur la liste des caches fournie par la balise `#CACHE_LISTE`.

Chaque cache est affiché avec son type, son nom, sa taille en octets, sa date de création et un lien de téléchargement. Il convient au plugin utilisateur de l'insérer lui-même dans une page car Cache Factory ne fournit aucune page par défaut.

3.2.2 Vidage des caches

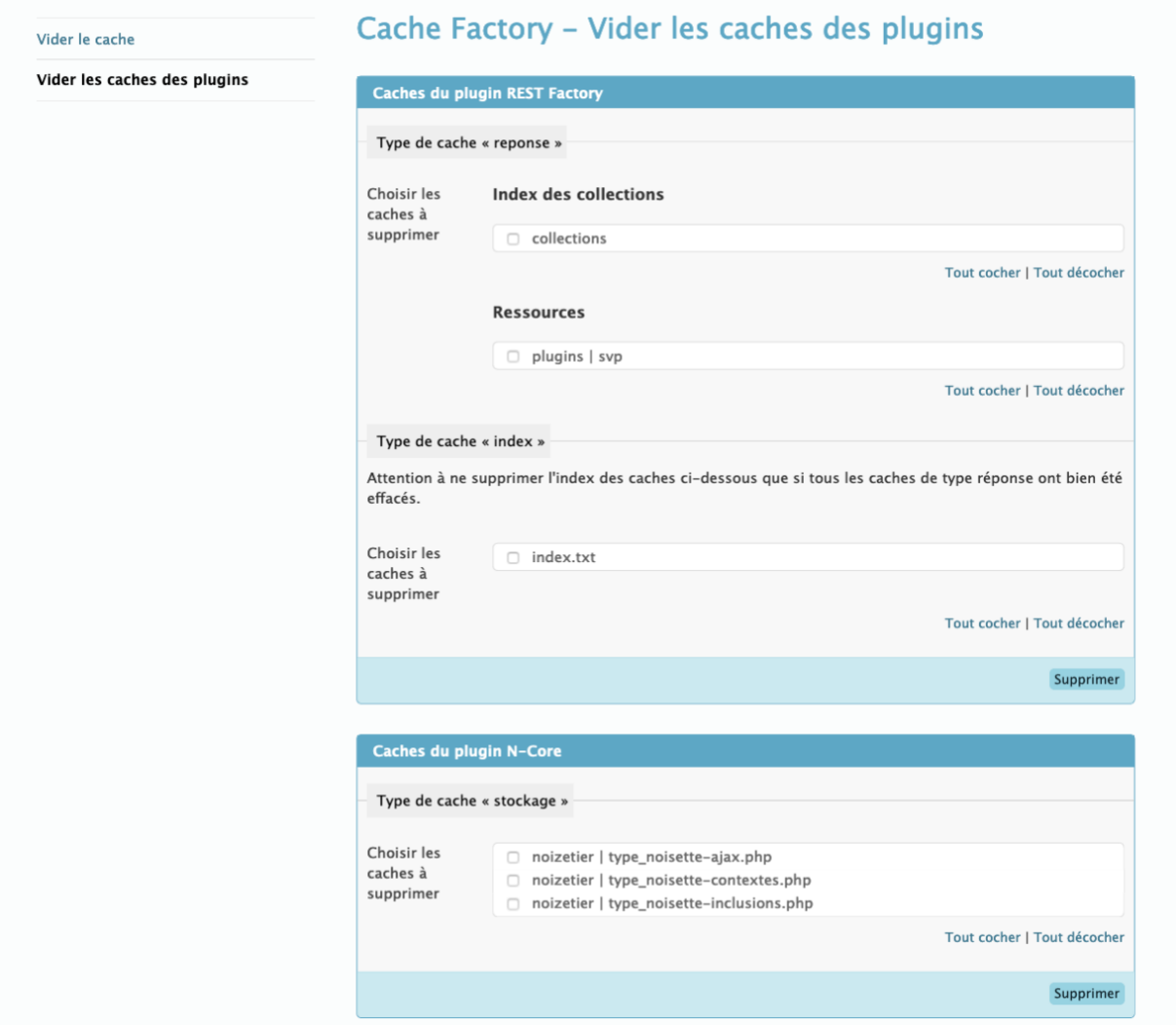
L'interface utilisateur de vidage des caches est composée d'une page dans l'espace privé, `cache_vider`, et d'un formulaire listant les caches d'un plugin utilisateur donné, `#FORMULAIRE_CACHE_VIDER`.

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]}`.

Un plugin possède un seul formulaire de vidage : les différents types de cache sont affichés dans un fieldset qui permet de séparer les listes de caches en fonction de leur type.

Chaque formulaire est personnalisable par le plugin utilisateur en surchargeant le service de chargement des paramètres du formulaire, le sous-squelette d'affichage du contenu du formulaire ou uniquement le squelette minimal d'affichage du label de chaque cache. Cette personnalisation est discutée au chapitre 6.



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 une 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 deux arguments obligatoires, `$plugin` et `$type_cache`, comme on peut le voir sur le prototype de `cache_lire()` :

```
function cache_lire($plugin, $type_cache, $cache)
```

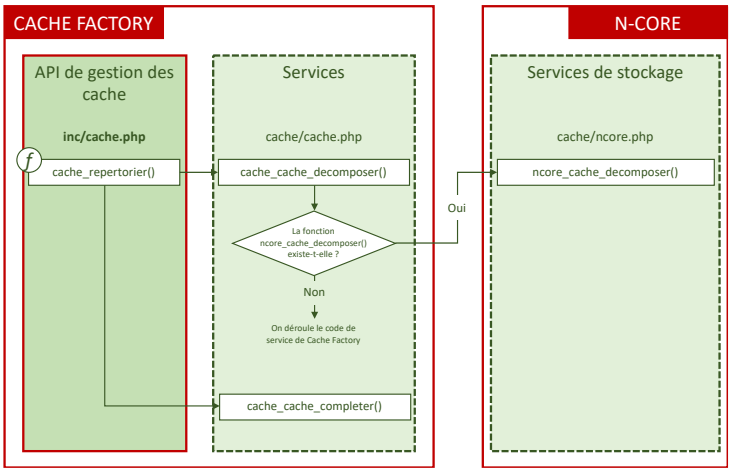
L'argument `$plugin` qualifie le module appelant, généralement un plugin comme N-Core. Il est donc recommandé d'utiliser le **préfixe du plugin** comme identifiant unique. Cet argument permet de distinguer les répertoires de stockage des caches d'un plugin utilisateur par rapport à d'autres.

L'argument `$type_cache` identifie la configuration de cache à utiliser, configuration qui est ensuite fournie aux services. De fait, chaque API possède l'instruction initiale suivante :

```
// Lecture de la configuration des caches du plugin.
$configuration = configuration_cache_lire($plugin, $type_cache);
```

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 de :

```
cache_repertorier('ncore', 'stockage');
```

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

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

La fonction utilitaire `service_cache_chercher()` cherche en premier lieu si un service utilisateur correspondant au couple (plugin, type de cache) existe et si il n'existe pas cherche en second lieu si un service utilisateur correspondant au plugin seul existe. Il est possible de désactiver la première recherche en passant un type de cache vide : c'est le cas du service cache_configurer() qui est toujours unique pour un plugin utilisateur et renvoie la configuration de tous les types de cache.

Le code de la fonction utilitaire est le suivant :

```
function service_cache_chercher($plugin, $type_cache, $fonction) {
  $fonction_trouvee = '';
  if ($plugin != 'ezcache') {
    include_spip("ezcache/${plugin}");
    $fonction_trouvee = "${plugin}_${type_cache}_${fonction}";
    if (
      !$type_cache
      or !function_exists($fonction_trouvee)
    ) {
      $fonction_trouvee = "${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 constitué soit du plugin appelant, soit du plugin appelant et du type de cache comme expliqué au paragraphe précédent - voir le code la fonction `service_cache_chercher()`.

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 de **tous les types de cache d'un plugin utilisateur** sous la forme d'un tableau associatif dont les index sont les identifiants de type de cache.
Services de gestion des identifiants de cache	
cache_verifier	Vérifie le contenu de l'identifiant relatif du cache. Par défaut, Cache Factory complète l'identifiant avec le préfixe et le sous-dossier si nécessaire et vérifie que les composants obligatoires du nom sont bien présents.
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 connaitre 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 type de cache, le nom sans extension, l'extension du fichier, sa taille et sa date.
Services de gestion du contenu d'un cache	
cache_valider	Complète le processus de validation du cache de Cache Factory. Par défaut, Cache Factory vérifie l'existence du fichier cache et sa péremption pour déterminer la validité puis si les tests sont ok, appelle le service complémentaire si il existe.
cache_decoder	Effectue le décodage du contenu brut du cache (chaine) 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	Reçoit en entrée les valeurs déjà chargées du formulaire de vidage et renvoie le complément de chargement pour un type de cache d'un plugin utilisateur donné. Par défaut, le plugin Cache Factory propose une version simplifiée du formulaire où tous les fichiers caches sont groupés par type de cache et listés par ordre alphabétique sans possibilité de classification au sein d'un même type.

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

Cache Factory propose l'ensemble des services dans le fichier `ezcache/ezcache.php` et `formulaires/cache_vider.php` pour le service spécial `cache_formulaire_charger()`, 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.

Si un plugin utilisateur veut personnaliser certains services de Cache Factory, il doit créer un fichier `ezcache/${plugin}.php` et y coder les services adéquats. Une explication complète de la mise en œuvre d'un plugin utilisateur est donnée au chapitre 6.

4.4 Le pipeline `post_cache`

Suite à l'écriture d'un 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 annexe pour conclure l'opération d'écriture ou de suppression en cours.

Il est possible d'utiliser l'API de Cache Factory dans ce pipeline sans provoquer de réentrance infinie. Pour cela, les fonctions concernées possèdent un argument optionnel nommé `$post_cache`, par défaut positionné à `true`. Il suffit de le positionner à `false` pour désactiver l'appel au pipeline.

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

Les informations fournies en entrée du pipeline dans l'index 'args' du flux sont :

ELEMENTS DE L'INDEX ARGS DU PIPELINE	
plugin	Le préfixe du plugin utilisateur. Permet à un plugin utilisateur P1 d'éviter de lancer ses traitements initiés par un plugin P2
fonction	Identifie la fonction générique faisant appel au pipeline. Prend les valeurs 'ecrire' ou 'supprimer'.
fichier_cache	Chemin complet du fichier cache.
cache	Identifiant relatif du cache.
configuration	Tableau de configuration du type de cache auquel appartient le cache concerné.

5. DONNEES DE CACHE FACTORY

5.1 La configuration générale des caches

La configuration d'un type de cache 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		
racine	Emplacement de base dans lequel sera créé le répertoire de stockage des caches du plugin utilisateur. Les valeurs possibles sont `'_DIR_CACHE'`, `'_DIR_VAR'`, `'_DIR_ETC'` et `'_DIR_TMP'`.	`'_DIR_CACHE'`
sous_dossier	Indique si le cache est inclus dans un sous-dossier du répertoire de stockage du plugin. Si `true`, par défaut, Cache Factory utilise le type de cache comme nom de sous-dossier ce qui permet de ne pas le fournir dans l'identifiant du cache. Néanmoins, il est toujours possible d'écraser la valeur par défaut en le fournissant dans l'identifiant relatif du cache.	`false`
nom_prefixe	Chaine fixe facultative appliquée systématiquement comme premier composant obligatoire du nom. Une fois configuré, il n'est plus jamais à fournir dans l'identifiant relatif du cache.	`''`
nom_obligatoire	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.	`''(*)`
nom_facultatif	Tableau des composants facultatifs et ordonnés du nom d'un cache. Les composants facultatifs suivent toujours les composants obligatoires.	`array()`
separateur	Caractère de séparation de chaque composant obligatoire ou facultatif du nom. Prend les valeurs `'_'`, `'-'` ou `''` si le nom ne comporte qu'un seul composant. Ce caractère ne doit pas être utilisé dans les composants du nom.	`''`
extension	Extension du fichier cache. Si le fichier cache est sécurisé l'extension est toujours forcée à `'.php'`.	`'.txt'`

securisation	Indique si le fichier doit être sécurisé ou pas.	`false`
serialisation	Indique si le contenu à stocker est un tableau à sérialiser ou pas.	`true`
decodage	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.	`false`
conservation	Durée de conservation du cache exprimée en secondes. La valeur 0 est utilisée pour indiquer que le cache est permanent.	`0`
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 `'_DIR_VAR'` alors le répertoire du plugin est un sous-dossier de nom `cache-${plugin}/`. Sinon, le sous-dossier porte le nom du préfixe du plugin.	
nom	Tableau regroupant dans l'ordre les noms obligatoires puis les noms facultatifs.	
type_cache	Recopie de l'identifiant du type de cache qui matérialise l'index du présent tableau.	

(*) : la valeur par défaut de l'index 'nom_obligatoire' dépend de l'existence ou pas d'un préfixe. Si un préfixe existe, le 'nom_obligatoire' peut être vide et, dans ce cas, le nom du cache vaut le nom du préfixe (cache unique). Si le préfixe est vide, l'index 'nom_obligatoire' a pour valeur par défaut 'nom'.

On notera que la configuration d'un type de cache fournie à tous les services de Cache Factory et donc à ceux éventuels du plugin utilisateur contient toujours le type de cache. C'est pour cela que cette donnée n'est jamais passée en argument principal des services.

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 des plugins utilisateur. Pour cela, Cache Factory utilise une meta nommée `cache`. Le contenu de la meta est un tableau de chaque configuration (voir la structure au paragraphe précédent) indexé par le préfixe du plugin et par le type de cache.

De cette façon, il est possible de lire toute la meta, les configurations d'un plugin donné, ou la configuration d'un type de cache d'un plugin donné.

6. MISE EN PLACE D'UN PLUGIN UTILISATEUR

6.1 La configuration générale

Cache Factory a besoin de connaitre 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()` qui est unique pour un plugin utilisateur.

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.

Cette configuration est insérée à l'index dont le nom coïncide avec l'identifiant du type de cache.

Par exemple, le plugin N-Core déclare la configuration suivante pour ses caches de type « stockage » via la fonction `ncore_cache_configurer()` incluse dans le fichier `ezcache/ncore.php` :

```
$configuration = array(
  'stockage' => array(
    'racine'         => '_DIR_CACHE',
    'sous_dossier'    => true,
    'nom_obligatoire' => array('objet', 'fonction'),
    'nom_facultatif'  => array(),
    'extension'      => '.php',
    'securisation'    => true,
    'serialisation'   => true,
    '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.

Pour mémoire, ce service ne peut pas être rendu par une fonction spécifique pour le couple (plugin, type de cache). Il est toujours global à un plugin et renvoie tous les types de cache. Le plugin Rest Factory est un exemple de plugin avec plusieurs types de cache (voir le fichier correspondant sur la forge https://git.spip.net/spip-contrib-extensions/ezREST/src/branch/master/ezcache/ezrest.php).

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, par type de cache, une liste de tous les caches du type avec le sous-dossier, si il existe, et le nom du cache. Chaque liste d'un type donné est incluse dans un fieldset. 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 de type météo regroupés par service et décomposés selon les éléments constitutifs de leur nom.

Pour permettre à Rainette et à d'autres plugins de personnaliser l'affichage, Cache Factory propose plusieurs niveaux de personnalisation, à savoir :

- compléter le chargement des valeurs du formulaires pour un type de cache donné ;
- surcharger le squelette minimal qui réalise l'affichage d'un label de cache de la liste pour un type de cache donné ;
- remplacer complètement le squelette principal qui réalise l'affichage hors titre et boutons du formulaire d'un plugin donné. Cette personnalisation n'est pas recommandée car il faut alors recoder un squelette assez complexe proposée par Cache Factory.

6.2.1 Fonctionnement standard du formulaire cache_vider

Le formulaire de vidage des caches est matérialisé par un fichier HTML dont le contenu principal hors balises de construction du formulaire, du titre et des boutons par le code suivant :

```
[(#CHEMIN{[formulaires/inc-(#ENV{_prefixe_plugin})_cache_vider.html]}|oui)
  <INCLURE{fond=formulaires/inc-#ENV{_prefixe_plugin}_cache_vider, env} />
]
[(#CHEMIN{[formulaires/inc-(#ENV{_prefixe_plugin})_cache_vider.html]}|non)
  <INCLURE{fond=formulaires/inc-ezcache_cache_vider, env} />
]
```

L'environnement du formulaire est composé à minima de la liste des variables suivantes, chargées par Cache Factory :

- `_prefixe_plugin`, le préfixe du plugin ;
- `_nom_plugin`, le nom du plugin ;
- `_types_cache`, le tableau des types de cache du plugin ;
- `_caches`, le tableau formaté de la liste des caches.

L'inclusion par défaut, `formulaires/inc-ezcache_cache_vider.html` reçoit l'environnement précité et affiche chaque type de cache dans un fieldset et, si besoin, présente la liste des caches par regroupements dont la sémantique reste à la charge du plugin utilisateur. Par défaut, le regroupement existe structurellement (tableau `_caches`, voir ci-après) mais n'est pas visible.

Une inclusion affiche le label de chaque checkbox (donc de chaque cache) de la façon suivante, ce qui autorise la personnalisation pour chaque type de cache d'un plugin donné :

```
<label for="#GET{id}"[ title="(#VALEUR{nom_cache})"]>
[(#CHEMIN{[formulaires/inc-(#ENV{_prefixe_plugin})_[(#GET{type_cache})_]cache_label.html]}|oui)
  <INCLURE{fond=formulaires/inc-#ENV{_prefixe_plugin}_#GET{type_cache}_cache_label, cache=#VALEUR} />
]
[(#CHEMIN{[formulaires/inc-(#ENV{_prefixe_plugin})_[(#GET{type_cache})_]cache_label.html]}|non)
  [[(#VALEUR{sous_dossier})&amp;nbsp;|&amp;nbsp;](#VALEUR{nom_cache})]
]
</label>
```

La fonction standard de chargement des données du formulaire renvoie pour la variable d'environnement `_caches` la structure suivante :

```
$valeurs['_caches'][$_type_cache][$plugin] = array(
  'titre'      => '',
  'explication' => '',
  'liste'      => $caches
);
```

6.2.2 Personnalisation du chargement

La fonction de chargement standard de Cache Factory, `formulaires_cache_vider_charger()`, fournit l'environnement standard décrit précédemment. Néanmoins, elle peut donner la main à une fonction de service spécifique au couple (plugin, type de cache) ou au plugin uniquement de façon à personnaliser l'environnement, et plus particulièrement, la liste des caches :

```
if ($charger = service_cache_chercher($plugin, $_type_cache, "cache_formulaire_charger")) {
  $valeurs = $charger($plugin, $valeurs, $options, $_configuration);
```

Le plugin peut donc remplir la variable `_caches` lui-même en respectant ou pas la structure attendue. Si il ne respecte pas la structure ou si il ajoute des données, il devra recoder entièrement l'inclusion principale (voir paragraphe 6.2.1).

Le plugin Taxonomie définit le service `taxonomie_apirest_cache_formulaire_charger()` dans le fichier `ezcache/taxonomie.php` pour regrouper ses caches de type « apirest » par service (ITIS, Wikipedia ou IUCN).

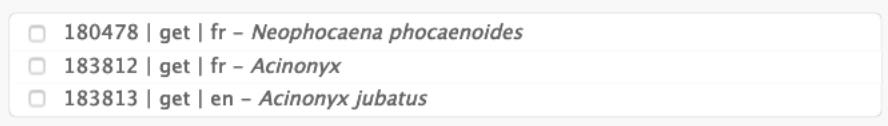
Le code de la fonction est le suivant :

```
function taxonomie_apirest_cache_formulaire_charger($plugin, $valeurs, $options, $configuration) {

  // On constitue la liste des services requis par l'appel
  include_spip('inc/taxonomie');
  $services = taxon_lister_services();

  // On récupère les caches et leur description pour donner un maximum.
  include_spip('inc/ezcache_cache');
  foreach ($services as $_service => $_titre) {
    // On récupère les caches du service
    $filtres = array('service' => $_service);
    $caches = cache_repertorier('taxonomie', 'apirest', $filtres);

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

  return $valeurs;
}
```

6.2.3 Personnalisation du label de chaque cache

La portion du squelette qui réalise l'affichage de chaque label de cache est personnalisable par le couple (plugin utilisateur, type de cache) uniquement. Pour ce faire, le plugin utilisateur doit créer une inclusion nommée `formulaires/inc-${plugin}_${type_cache}_cache_label.html`. Si elle existe elle remplacera l'affichage standard du sous-dossier et du nom du cache.

Le plugin Taxonomie possède une inclusion personnalisée qui permet d'afficher les composants sémantiques du nom du cache de façon distincte `formulaires/inc-taxonomie_apirest_cache_label.html`.



6.2.4 Remplacement du contenu du formulaire

Il est aussi possible de surcharger entièrement le contenu du formulaire de vidage des caches en créant son propre squelette `formulaires/inc-${plugin}_cache_vider.html` mais cette option est très peu utilisée étant donné la facilité d'utilisation et la généricité des méthodes présentées ci-dessus.

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 `${plugin}_${type_cache}_cache_[de]composer()` ou `${plugin}_cache_[de]composer()` 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 'type_cache', 'nom_cache', 'extension_cache', 'taille_cache' et 'date_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()`.

Etant donné que Taxonomie n'utilise qu'un type de cache cette personnalisation fonctionne correctement. Néanmoins, il aurait été plus pertinent et surtout plus pérenne de nommer la fonction avec le type de cache « apirest », soit `taxonomie_apirest_cache_completer()`.

6.4 La validation d'un cache

Cache Factory propose des critères par défaut pour vérifier si un cache est valide ou pas. Ces critères sont l'existence du fichier cache et sa péremption (durée du cache échue comparée à sa configuration).

Néanmoins, en créant une fonction de service `${plugin}_${type_cache}_cache_valider()` ou `${plugin}_cache_valider()`, un plugin utilisateur peut rajouter la vérification de critères propres. La fonction renvoie le résultat de la vérification sous la forme d'un booléen.

Ce service peut être utile quand la durée de conservation n'est pas un critère pertinent et que l'on veut invalider un cache sur un évènement du système (mise à jour en base de données, etc).

6.5 L'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 même plugin utilisateur car sinon on déclencherait un traitement inadéquat.

Il est aussi à noter dans l'exemple de REST Factory fourni ci-dessous que l'index des caches est aussi un cache (unique) dont le nom est réduit au préfixe ('index') et pour lequel l'écriture est réalisée en désactivant l'appel au pipeline pour éviter la réentrance (argument final à `false`).

```
// On vérifie que l'appel du pipeline est bien consécutif à une action sur les caches de REST Factory
if (
  ($flux['args']['plugin'] === 'ezrest')
  and ($flux['args']['configuration']['type_cache'] !== 'index')
) {
  // Lecture du fichier d'index et récupération du tableau des caches enregistrés.
  include_spip('inc/ezcache_cache');
  $cache_index = array();
```

```
  $index = cache_lire('ezrest', 'index', $cache_index);

  // 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 sans appeler le pipeline post_cache pour éviter la réentrance.
  cache_ecrire('ezrest', 'index', $cache_index, $index, false);
}
```

6.6 Le choix du niveau de surcharge

A partir de la branche v1, le plugin Cache Factory permet de surcharger un service :

- soit en définissant une fonction spécifique au couple (plugin, type de cache), en utilisant le préfixe `${plugin}_${type_cache}_` devant le nom du service ;
- soit en définissant une fonction spécifique au plugin uniquement, en utilisant le préfixe `${plugin}_` devant le nom du service.

Il est **recommandé d'utiliser la première méthode de préférence** car si la liste des types de cache évolue, les services existants ne seront pas modifiés. Néanmoins, il se peut que les traitements soient plus ou moins identiques pour tous les types de cache d'un plugin. Dans ce cas, il peut être avantageux de définir une unique fonction spécifique au plugin et de gérer les différences au sein de cette fonction. Mais c'est un cas assez rare.

Il est d'ailleurs toujours possible d'utiliser la deuxième méthode en particulier si un seul type de cache est utilisé, ce qui est fréquent. Les plugins utilisateurs déjà codés utilisent encore cette stratégie mais seront migrés petit à petit vers la méthode recommandée.

6.7 Conclusion

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

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

Un exemple classique d'utilisation des API est donnée ci-dessous (plugin Spiper Ipsum) pour le type de cache nommé « lecture » :

```
if ((!$fichier_cache = cache_est_valide('spiperipsum', 'lecture', $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', 'lecture', $cache, $tableau);
} else {
  // Lecture des données du fichier cache valide
  $tableau = cache_lire('spiperipsum', 'lecture', $fichier_cache);
}
```

Certains plugins voudront également personnaliser l'affichage du formulaire de vidage en codant leur version du service `cache_formulaire_charger()` et de l'inclusion `formulaires/inc-${plugin}_${type_cache}_cache_label.html`.

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

Les plugins utilisateurs de Cache Factory sont actuellement :

- Rainette,
- Taxonomie,
- Rest Factory,
- N-Core,
- Spiper Ipsum,
- et Check Factory.

Ils donnent un aperçu complet de l'utilisation de Cache Factory et de ses personnalisations.

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 les arguments `$plugin` et `$type_cache`.

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 de N-Core. Pour un plugin, l'utilisation du préfixe est recommandée.

L'argument **obligatoire** `$type_cache` est toujours le **deuxième** argument du prototype des fonctions d'API. C'est une chaine de caractères qui **identifie le type de cache**.

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

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

7.3 Documentation

Le plugin Cache Factory est entièrement documenté suivant la syntaxe PHPDoc. De fait, cette documentation technique est disponible sur le site SPIP de codes des plugins : https://code.spip.net/ezcache/.

8. GRAPHE D'APPEL API – SERVICES

API – SERVICE	
Fonctions liées aux caches	
cache_ecrire	configuration_cache_lire cache_cache_composer pipeline post_cache
cache_est_valide	configuration_cache_lire cache_cache_composer cache_cache_valider
cache_lire	configuration_cache_lire cache_cache_composer cache_cache_decoder
cache_nommer	configuration_cache_lire cache_cache_composer
cache_repertorier	configuration_cache_lire cache_cache_decomposer cache_cache_completer
cache_ supprimer	configuration_cache_lire cache_cache_composer pipeline post_cache
cache_vider	pipeline post_cache
Fonctions liées à la configuration	
configuration_cache_effacer	
configuration_cache_lire	cache_cache_configurer