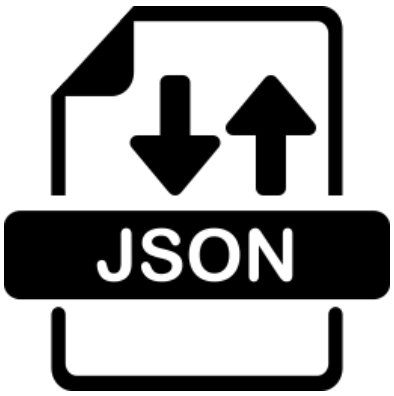
Référence SPIP/X/2019.002 – Ed. 1.0

GUIDE
CONCEPTION DU PLUGIN REST FACTORY



FICHE D'IDENTIFICATION	
Rédacteur	Eric Lupinacci
Projet	SPIP
Étude	Conception du plugin REST Factory
Nature du document	Guide
Date	21/10/2019
Nom du fichier	Guide - Le plugin REST Factory.docx
Référence	SPIP/X/2019.002 – Ed. 1.0
Dernière mise à jour	01/11/2019 17:57:59
Langue du document	Français
Nombre de pages	24

TABLE DES MATIERES

1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception du plugin REST Factory (version 0.1.0 et ultérieures) dont l'objectif est de fournir un cadre standard et simplifié pour développer des API REST.

Le plugin REST Factory fournit, d'une part, les fonction REST nécessaires à l'implémentation d'une API REST - dénommée `ezrest` - conforme à l'organisation imposée par le plugin Serveur HTTP abstrait et permet, d'autre part, une mise en œuvre simplifiée de collections et de ressources au travers de cette API.

Ce plugin est un outil de développement utilisable par d'autres plugins qui souhaitent mettre à disposition, avec le minimum d'effort, leurs données au travers d'une API REST.

2. CONCEPTS

2.1 L'API REST

2.1.1 Le concept d'API REST

Les API REST sont basées sur HTTP. L'échange s'opère au travers de requêtes client et de réponses du serveur. Ce sont des méthodes qui définissent les requêtes que le client peut effectuer, parmi lesquelles on peut citer GET, PUT, POST, DELETE.

Une API REST doit respecter des critères précis pour être considérée comme conforme (ou RESTful) :

- client-serveur ;
- sans état (stateless) ;
- avec mise en cache (cacheable) ;
- en couches ;
- avec une interface uniforme ;

Le principe du client-serveur définit les deux entités qui interagissent dans une API. Un client envoie une requête, et le serveur renvoie une réponse. Ce dernier doit avoir le plus d'informations possibles sur le client, car il est important qu'ils soient capables de travailler indépendamment l'un de l'autre.

Le fait d'être « sans état » signifie que le serveur n'a aucune idée de l'état du client entre deux requêtes. Du point de vue du serveur, chaque requête est une entité distincte des autres.

Le cache, pour les API REST, met en jeu le même principe que pour le reste d'Internet : un client doit être capable de garder en mémoire des informations sans avoir constamment besoin de demander tout au serveur.

La contrainte d'interface uniforme est fondamentale dans la conception de n'importe quel système REST. Elle simplifie et découple l'architecture, ce qui permet à chaque composant d'évoluer indépendamment. Les quatre contraintes de l'interface uniforme sont les suivantes.

- identification des ressources dans les requêtes ;
- manipulation des ressources par des représentations ;
- messages auto-descriptifs ;
- hypermédia comme moteur d'état de l'application (HATEOAS).

Les réponses du serveur pour les API REST peuvent être délivrées dans de multiples formats. JSON (JavaScript Object Notation) est souvent utilisé, mais XML, CSV, ou même RSS sont aussi possibles.

2.1.2 La mise en œuvre dans le plugin « Serveur HTTP abstrait »

Le plugin Serveur http abstrait est la première pierre dans la construction d'une API REST : il a pour but de normaliser des URL que l'on peut appeler pour manipuler les données d'un site. À chaque URL, il recherche une fonction précise, qui va implémenter la fonctionnalité demandée.

Pour cela, il fournit une unique action `action/api_http.php` qui va gérer trois paramètres :

- le **nom** de l'API réellement implémentée, appelée « format » dans le plugin ;
- une **collection**, à savoir le type des données convoitées, par exemple, le nom – au pluriel - d'un objet éditorial de SPIP ;
- une **ressource**, à savoir, l'identifiant unique d'un objet.

Ces trois paramètres sont tout simplement ajoutés à la suite dans l'URL, séparés par des « / ». Seul le premier est toujours obligatoire, les autres sont ajoutés en fonction de la requête (par exemple, `/http.api/atom/articles/12`, pour l'article d'id 12).

En outre, pour se simplifier la tâche, le plugin intègre et utilise la librairie HTTPFoundation fournie par Symfony qui fournit l'objet Request et l'objet Response, qui seront créés, utilisés et modifiés au fil du temps dans les fonctions d'implémentation.

Enfin, le plugin organise la construction d'une API REST en fournissant le cadre technique pour son développement au travers de points d'entrée de service pour collecter les données, gérer les erreurs et définir les autorisations. De fait, un plugin utilisateur souhaitant proposer une API REST du nom `monapi` devra créer à minima un fichier `http/monapi.php` avec l'ensemble des services nécessaires conformément au cadre défini par le Serveur HTTP abstrait.

2.1.3 L'API REST du plugin « REST Factory »

Le plugin REST Factory implémente une API dénommée `ezrest` conformément au cadre défini par le plugin Serveur HTTP abstrait. De fait, les URL sont du type `/http.api/ezrest/…` et le fichier `http/ezrest.php` contient l'ensemble des fonctions nécessaires au fonctionnement de l'API.

Pour simplifier la mise en place d'API REST au-delà de ce que propose le plugin Serveur HTTP abstrait, REST Factory définit un cadre standard pour gérer les collections et ressources au travers de son API : **vérification, traitement de données et configuration sont standardisés** pour convenir à la plupart des besoins et, des points d'entrée spécifiques permettent de personnaliser le fonctionnement proposé par défaut. Les réponses sont toujours fournies au format JSON.

Dans cette version du plugin, seule la méthode GET pour une collection ou une ressource est mise à disposition.

2.2 Les collections

Une collection possède un identifiant unique de type chaine qui en général décrit l'objet au pluriel comme `plugins` ou `categories`.

Une collection est fournie par un plugin utilisateur de REST Factory : il déclare sa configuration et propose les services idoines permettant de vérifier les requêtes sur cette collection et d'en récupérer les données.

2.3 Les ressources

Une ressource est un objet particulier d'une collection désignée par un identifiant unique qui permet d'accéder à l'objet en base de données : par exemple, l'id pour un article ou le préfixe pour un plugin.

C'est la configuration qui indique si la collection accepte la lecture d'une ressource et si oui au travers de quel identifiant (un plugin peut être identifié par son id ou son préfixe). Il n'est pas toujours utile de proposer l'accès à une ressource précise (peu d'information). C'est le cas, par exemple, de la collection des catégories.

3. PERIMETRE DU PLUGIN REST FACTORY

REST Factory fournit les fonctions permettant d'implémenter l'API REST `ezrest` conformément aux principes du Serveur HTTP abstrait, un pipeline pour configurer les collections et des points d'entrée pour les services de vérification et de manipulation des collections et ressources.

3.1 Les fonctions REST de mise en œuvre du Serveur HTTP abstrait

Pour implémenter une API REST limitée à la méthode GET sur les collections et les ressources, le Serveur HTTP abstrait requiert le codage de la liste des fonctions décrites ci-dessous. Ces fonctions sont appelés « **fonctions REST** » dans la suite du guide.

FONCTIONS REST : HTTP/EZREST.PHP	
http_ezrest_get_index_dist	Fait un GET sur l'API `ezrest`. La requête est du type `/ezrest` et renvoie la liste des collections disponibles présentées par plugin fournisseur.
http_ezrest_get_collection_dist	Fait un GET sur une collection gérée par l'API `ezrest`. La requête est du type `/ezrest/xxx` et renvoie les objets associées contenus dans la base du serveur. Il est possible de filtrer la collection et de compléter le contenu de la réponse en utilisant le pipeline `post_ezcollection`.
http_ezrest_get_ressource_dist	Fait un GET sur une ressource d'une collection gérée par l'API `ezrest`. La requête est du type `/ezrest/xxx/yyy` et renvoie l'objet désigné dans la base. Il est possible de compléter le contenu de l'objet en utilisant le pipeline `post_ezressource`.
http_ezrest_erreur_dist	Traite les erreurs directement détectées par le serveur HTTP abstrait uniquement. Celles-ci sont mises au format de l'API REST `ezrest` et fournies au client systématiquement en JSON.

Ces fonctions font appel à des services standardisés pour effectuer les vérifications idoines de chaque requête et constituer la réponse. Certains de ces services sont personnalisables par les plugins utilisateur comme décrit au chapitre 0.

3.2 La configuration des collections

Le plugin REST Factory ne propose aucune collection par défaut mais attend des plugins utilisateur qu'ils déclarent les collections utilisables au travers d'un pipeline nommé `liste_ezcollection`. Le détail de cette configuration est fournie au chapitre 5.1.

4. FONCTIONNEMENT DE REST FACTORY

4.1 La dissociation Fonctions REST – Services

Le fonctionnement global du plugin REST Factory est similaire à celui des plugins N-Core et Cache Factory : il utilise la même architecture Fonctions Principales - Services.

De façon générale, un plugin utilisateur comme Nomenclatures va s'appuyer sur l'ensemble des fonctions REST de REST Factory sans faire aucune modification. Ce faisant, il doit définir la configuration des collections qu'il souhaite mettre à disposition et coder les traitements spécifiques de vérification et de manipulations de ces mêmes collections. Par conception, **REST Factory dissocie la fonction REST, des services propres à un plugin utilisateur qui en personnalisent l'usage**.

Dans le code de ses fonctions REST, REST Factory appelle des services, tous contenus dans le fichier `ezrest/ezrest.php`. Ces services proposent, soit des traitements standard (construction des réponses, analyse des requêtes) parfois personnalisables par le plugin utilisateur, soit ne font qu'aiguiller vers le traitement spécifique de ce même plugin utilisateur (état de l'API, récupération des données de la collection).

Pour permettre la personnalisation du plugin utilisateur, les services de REST Factory possède un argument obligatoire, `$plugin`, comme on peut le voir sur le prototype du service suivant :

```
function ezrest_reponse_informer_plugin($plugin, $contenu)
```

L'argument `$plugin` qualifie le module utilisateur, un plugin comme Nomenclatures. Il est donc indispensable d'utiliser le **préfixe du plugin** comme identifiant unique.

4.2 L'aiguillage des services

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

Pour cela, les services REST Factory appellent une fonction utilitaire `ezrest_service_chercher()` qui retourne le nom de la fonction de service spécifique au plugin utilisateur ou la chaine vide si aucune fonction n'est définie dans le plugin utilisateur :

```
if ($completer = ezrest_service_chercher($plugin, 'reponse_informer_plugin')) {
   $contenu = $completer($contenu);
}
```

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

```php
function ezrest_service_chercher($plugin, $fonction, $prefixe = '', $suffixe = '') {

   $fonction_trouvee = '';

   // Eviter la réentrance si on demande explicitement le plugin ezrest.
   if ($plugin != 'ezrest') {
      include_spip("ezrest/${plugin}");
      $fonction_trouvee = $prefixe ? ($suffixe ? "${prefixe}_${fonction}_${suffixe}" :
"${prefixe} ${fonction}") : "${plugin} ${fonction}";
      if (!function_exists($fonction_trouvee)) {
         $fonction_trouvee = '';
      }
   }

   return $fonction_trouvee;
}
```

4.3 Les services

4.3.1 Typologies des services

On distingue deux typologies des services suivant que l'on se place du coté du plugin REST Factory ou du plugin utilisateur.

Pour le plugin REST Factory, les services peuvent être classés en fonction de l'objet auxquels ils s'adressent. On distingue les services liés à l'API elle-même, les services liés aux requêtes (plus précisément l'adéquation à la configuration de la collection), les services liés aux réponses et les services liés à la gestion des données recherchées. Le paragraphe suivant décrit chacun de ces services en suivant cette typologie.

Pour un plugin utilisateur, la typologie adaptée considère le service comme lié ou pas à une collection. C'est cette distinction qui régit d'ailleurs le nommage des services spécifiques du plugin utilisateur. Pour un service non lié à une collection, le service spécifique du plugin utilisateur portera un nom homonyme à celui du service REST Factory. Pour un service lié à une collection, le nom sera construit à partir de l'identifiant de la collection et ne correspondra pas forcément à celui du service REST Factory appelant. Le chapitre 6 décrit la mise en place de ces services en suivant cette typologie.

4.3.2 Description des services

Les fonctions REST font donc appel à des services internes à REST Factory dont la description exacte est fournie ci-après. Le nom des fonctions est amputé du préfixe « ezrest_ ».

Si un service est personnalisable par un plugin utilisateur, la colonne de droite indique si le service du plugin utilisateur est lié à une collection (indication « C ») ou non lié à une collection (indication « N »). L'absence d'indication désigne un service non personnalisable.

SERVICES : EZREST/EZREST.PHP		
Services concernant l'API		
api_verifier_contexte	Détermine si le serveur est capable de répondre aux requêtes. Par défaut, l'API `ezrest` ne fait aucune vérification. C'est donc au plugin utilisateur de fournir un service spécifique si une vérification doit être effectuée afin d'assurer le fonctionnement de l'API.	NC
Services concernant la réponse		
reponse_initialiser_contenu	Initialise le bloc contenu d'une réponse (index 'requete', 'erreur' et 'donnees'). En particulier, la fonction stocke les éléments de la requête et positionne le bloc d'erreur par défaut à ok (code 200).	-
reponse_informer_plugin	Complète l'initialisation du bloc contenu d'une réponse avec des informations sur le plugin utilisateur. La fonction remplit de façon standard un nouvel index 'plugin' et permet ensuite au plugin utilisateur de personnaliser encore le contenu initialisé, si besoin.	NC
reponse_expliquer_erreur	Complète le bloc d'erreur avec le titre et l'explication de l'erreur. Les textes sont calculés à partir d'items de langue fournis par REST Factory ou par le plugin utilisateur suivant le contexte.	NC
reponse_construire	Finalise la réponse à la requête en complétant le header et convertissant le bloc contenu au format JSON avant de l'insérer dans l'objet réponse.	-
Services concernant la collection désignée dans la requête		
collection_verifier	Détermine si la collection demandée est valide. Par défaut, REST Factory vérifie que la collection est bien déclarée dans la liste des collections. Si c'est le cas, la fonction permet ensuite au plugin utilisateur de compléter la vérification.	C
collection_verifier_filtre	Détermine si les filtres passés dans la requête sont valides. Par défaut, la fonction vérifie l'absence d'un filtre obligatoire et la validité du nom de chaque critère de filtre. Si c'est le cas, la fonction permet ensuite au plugin utilisateur de vérifier la valeur fournie pour chaque filtre. Si plusieurs filtres sont fournis, la fonction s'interrompt dès qu'elle trouve un filtre invalide.	C

collection_verifier_ressource	Détermine si la ressource demandée est valide. Par défaut, la fonction vérifie que la collection autorise bien l'accès à une ressource. Si c'est le cas, la fonction permet ensuite au plugin utilisateur de vérifier la validité de l'identifiant de la ressource.	C
Services concernant les données		
indexer	Peuple le bloc 'donnees' en réponse à la requête d'index des collections disponibles sur le serveur. Les collections sont présentées en regard de chaque plugin fournisseur avec ses principaux paramètres de configuration.	-
collectionner(*)	Peuple le bloc 'donnees' en réponse à la requête de lecture d'une collection. Par défaut, le service REST Factory ne fait rien, tout est du ressort du plugin utilisateur.	C
ressourcer(*)	Peuple le bloc 'donnees' en réponse à la requête de lecture d'une ressource. Par défaut, le service REST Factory ne fait rien, tout est du ressort du plugin utilisateur.	C

Les services notés (*) doivent toujours être définis par le plugin utilisateur, les autres sont optionnels car REST Factory propose déjà des traitements standard.

REST Factory propose l'ensemble des services dans son fichier `ezrest/ezrest.php` ce qui permet de minimiser les développements pour la plupart des plugins utilisateur car les traitements par défaut s'avèrent souvent suffisants. A minima, un plugin utilisateur peut se contenter de déclarer les collections qu'il propose et de coder les services de récupération des données associés.

4.4 Les pipelines `post_ezcollection` et `post_ezressource`

Suite à l'appel des services de récupération des données, ezrest_collectionner() et ezrest_ressourcer(), les fonctions REST de traitement d'un GET sur une collection ou une ressource font appel à un pipeline respectivement post_ezcollection et post_ezressource. Ces pipelines permettent à un plugin de compléter l'ensemble des données récupérées.

C'est le cas du plugin SVP Typologie qui rajoute la catégorie et le tag de chaque plugin retourné (voir aussi le paragraphe 6.3.1).

4.5 La gestion des erreurs

REST Factory simplifie aussi la gestion des erreurs pour les plugins utilisateur. En effet, ceux-ci n'ont jamais à se préoccuper de remplir le bloc d'erreur complet (voir le paragraphe 5.3.2) mais se contentent, en général, de fournir un **titre** et/ou un **détail** au travers d'items de langue du plugin voir l'élément sur lequel porte l'erreur. Les index modifiables du bloc d'erreur dépendent du service (voir le chapitre 6).

Le format des items de langue est imposé par REST Factory et se déduit des index du bloc d'erreur. Seul les index `titre` et `detail` du bloc d'erreur sont concernés. Les formats sont les suivants, respectivement pour le titre et le détail :

```
${plugin}:erreur_${statut}_${type}_titre
${plugin}:erreur_${statut}_${type}_message
```

Enfin, lors de la traduction de l'item de langue, REST Factory fournit toujours les paramètres `collection`, `element`, `valeur` et `extra` qui peuvent être utilisés dans l'item de langue pour améliorer la précision du texte.

5. DONNEES DE REST FACTORY

Le plugin REST Factory ne possède aucune configuration ni aucun stockage de données permanent. Il manipule néanmoins des données comme les collections, les requêtes, les réponses et les erreurs qui possèdent des structures de données standard expliquées ci-après.

5.1 La configuration des collections

Au travers du pipeline `liste_ezcollection`, REST Factory permet à un plugin utilisateur de déclarer les collections qu'ils proposent et leur configuration. La configuration d'une collection se matérialise par un tableau associatif dont les index sont expliqués ci-dessous. Les valeurs par défaut sont fournies dans la colonne de droite et la caractère obligatoire « O » ou facultatif « F » dans la colonne qui précède.

CONFIGURATION D'UNE COLLECTION			
Paramètres généraux			
module	Préfixe du plugin fournissant la collection.	O	
filtres	Tableau des filtres autorisés sur la collection. Peut-être vide	F	`array()`
ressource	Champ identifiant la ressource de façon unique dans la table matérialisant la collection et servant à la désigner dans l'url. Par exemple, le préfixe pour les plugins.	F	`''`
Paramètres d'un filtre			
critere	Nom du filtre dans l'URL. Peut coïncider ou pas au champ dans la table.	O	
est_obligatoire	Indique si le filtre doit toujours être utilisé lors d'une requête sur la collection ou pas.	F	`false`
champ_table	Désigne le nom du champ de la table représentant le critère. Ce paramètre ne sert qu'au plugin utilisateur pour construire la condition de lecture de la collection. Peut être omis si le paramètre critère désigne déjà le champ de la table.	F	`''`

L'identifiant de la collection ne fait pas partie du bloc de configuration car il est passé comme index du tableau global de toutes les collections. Cet identifiant est une chaine qui représente en général un type d'objet au pluriel (par exemple, plugins). Par exemple, la déclaration de la collection `pays` se fait comme suit :

```
$collections['pays'] = array(
   'ressource' => 'code alpha2',
   'module'    => 'isocode',
   'filtres'   => array(
      array(
         'critere'         => 'region',
         'champ_table'     => 'code_num_region',
         'est_obligatoire' => false
      ),
      array(
         'critere'         => 'continent',
         'champ_table'     => 'code_continent',
         'est obligatoire' => false
      ),
   )
);
```

Un plugin n'est pas obligé de fournir uniquement des collections entières mais peut parfois ne rajouter qu'un filtre. C'est le cas du plugin SVP Typologie qui permet de filtrer la collection `plugins` sur la catégorie. Dans ce cas, il est nécessaire de s'assurer que la collection de base a bien été déclarée au préalable dans la séquence du pipeline.

5.2 La structure des requêtes

Le plugin Serveur HTTP abstrait utilise la librairie HTTPFoundation fournie par Symfony ce qui lui évite de gérer toute la tripaille des requêtes et des réponses en HTTP. Cette librairie permet au plugin Serveur HTTP abstrait de traduire l'URL en un objet **Request** contenant les éléments de la demande et fourni au plugin REST Factory. En particulier, le plugin Serveur HTTP abstrait initialise un bloc de l'objet Request nommé `attributes` destiné à accueillir des données spécifiques à l'application.

De cette façon, le plugin Serveur HTTP abstrait, ajoute trois paramètres essentiels dans le bloc `attributes`, à savoir, le format (l'identifiant de l'API), la collection et la ressource :

```
$requete->attributes->add(array(
   'format' => $format,
   'collection' => $collection,
   'ressource' => $ressource,
));
```

De son côté, le plugin REST Factory ne manipule l'objet Request qu'en lecture.

5.3 La structure des réponses

La librairie HTTPFoundation permet également au plugin Serveur HTTP abstrait d'initialiser un objet **Response** qui est ensuite fourni au plugin REST Factory. Cet objet Response contient un bloc `content` qui recueille le contenu spécifique. C'est ce bloc `content` que le plugin REST Factory, d'une part, normalise pour limiter le travail des plugins utilisateur à la fourniture des données et, d'autre part, remplit au format JSON.

Jusqu'à l'instruction de mise à jour du bloc `content`, le plugin REST Factory manipule ce bloc sous la forme d'un tableau associatif dont les index de premier niveau sont : `requête`, `erreur`, `donnees` et `fournisseur`. Ces sous-blocs sont décrits dans les paragraphes suivants.

5.3.1 La structure du sous-bloc `requete`

Le sous-bloc `requete` est toujours présent dans le contenu d'une réponse. Ce sous-bloc est entièrement initialisé par le plugin REST Factory à partir des informations de la requête et n'est pas personnalisable. Il permet d'expliquer comment l'API a compris la requête et de comparer avec les données reçues en cas de problème. C'est un outil de débusquage.

Le contenu exact est décrit ci-dessous, la colonne de droite indiquant la valeur ou le type de valeur possible.

SOUS-BLOC REQUETE		
methode	La méthode HTTP associée à la requête.	`'GET'`
format	L'identifiant de l'API, soit dans ce cas celle de REST Factory	`'ezrest'`
collection	Identifiant de la collection si elle existe. Si l'identifiant est vide c'est que la requête concerne la lecture de l'index des collections.	`chaine`
ressource	Identifiant de la ressource. Cette valeur doit être compatible avec le champ de la table choisi pour être l'identifiant (id d'un article, le préfixe d'un plugin…)	`chaine ou entier`
filtres	Tableau des filtres sous la forme [critère] = valeur.	`array`
format_contenu	Format du contenu retourné.	`'json'`

5.3.2 La structure du sous-bloc `erreur`

Le sous-bloc d'erreur est toujours présent dans le contenu d'une réponse. Ce sous-bloc est initialisé par le plugin REST Factory et éventuellement personnalisé par le plugin utilisateur si il détecte lui-même une erreur.

SOUS-BLOC ERREUR		
Index renvoyés dans la réponse		
statut	Code standard de la réponse HTTP. Positionné par REST Factory, peut prendre théoriquement l'ensemble des valeurs autorisées.	
type	Identifiant unique de l'erreur au format [a-z_]	
element	Libellé de l'élément concerné par l'erreur	
valeur	Valeur de l'élément provoquant l'erreur	
titre	Texte explicatif court traduit d'un item de langue	

detail	Texte complémentaire fournissant des détails sur l'erreur ou sa résolution. Traduit d'un item de langue.	
Index temporaires utilisés pour traduire les textes (non renvoyés dans la réponse)		
extra	Information complémentaire fournie en paramètre des items de langue.	
module	Tableau indiquant quel module de langue utiliser pour le titre et le détail séparément. Prend la valeur 'ezrest' ou celle du préfixe du plugin fournisseur suivant le contexte.	

5.3.3 La structure du sous-bloc `donnees`

Le sous-bloc `donnees` ne possède pas de structure imposée. C'est le plugin fournisseur de la collection qui décide de la forme de son contenu. La seule recommandation est que cette structure soit la plus claire possible et corresponde le mieux aux besoins des utilisateurs.

Ce sous-bloc peut être un tableau vide car aucune données ne correspond aux critères de la requête. Dans ce cas aucune erreur n'est remontée.

5.3.4 La structure du sous-bloc `fournisseur`

Le sous-bloc fournisseur permet de connaitre le plugin fournisseur de la collection, sa version et éventuellement son schéma de données. Ce sous-bloc est personnalisable par les plugins fournisseur de collections.

De fait, si une erreur est détectée au niveau de l'API par le service `api_verifier_contexte()` ou de la collection demandée par le service `requête_vérifier_collection()`, ce bloc ne sera pas présent dans la réponse car le plugin n'est identifié qu'à partir du moment où la collection est valide (voir la configuration des collections).

SOUS-BLOC FOURNISSEUR		
plugin	Le préfixe du plugin fournisseur de la collection.	`chaine`
version	La version courante du plugin sur le serveur.	`version`
schema	Si il existe, la version du schéma de données du plugin.	`version ou entier`

6. LA MISE EN PLACE D'UN PLUGIN UTILISATEUR

Ce chapitre décrit en détail l'écriture des fonctions nécessaires à la mise en place d'une API REST de format `ezrest`.

6.1 La déclaration des collections

La déclaration des collections fournies par un plugin utilisateur de l'API ezrest s'effectue par l'intermédiaire du pipeline `liste_ezcollection`. Le plugin utilisateur doit donc déclarer le pipeline dans son paquet.xml et fournir la fonction `${prefixe}_liste_ezcollection()`, où `${prefixe}` correspond au préfixe du plugin.

Le format de déclaration est défini au paragraphe 5.1. Le plugin REST Factory fournit également un fichier « template », `prefixe_pipelines.php.template`, pour faciliter le codage.

Le plugin SVP API qui fournit les collections `plugins` et `depots` (données de SVP) utilise le fichier `svpapi_pipelines.php` pour coder la fonction `svpapi_liste_collection()`. La déclaration de la collection plugins est la suivante :

```
function svpapi_liste_ezcollection($collections) {

   // Initialisation du tableau des collections
   if (!$collections) {
      $collections = array();
   }

   // Les index désignent les collections
   $collections['plugins'] = array(
      'ressource' => 'prefixe',
      'module'    => 'svpapi',
      'filtres'   => array(
         array(
            'critere'         => 'compatible_spip',
            'est obligatoire' => false
         ),
      )
   );

   $collections['depots'] = array(
      'module'  => 'svpapi',
      'filtres' => array(
         array(
            'critere'         => 'type',
            'est_obligatoire' => false
         )
      )
   );

   return $collections;
}
```

Le plugin SVP Typologie, comme indiqué dans le paragraphe 5.1, complète la configuration de la collection `plugins` en ajoutant le filtre `categorie`. Pour ce faire, le code suivant est inclus dans la fonction `svptype_liste_collection()` :

```
// Rajoute le filtre de catégorie dans la collection plugins de SVP API.
if (isset($collections['plugins'])) {
   $collections['plugins']['filtres'][] = array(
      'critere'         => 'categorie',
      'module'          => 'svptype',
      'est_obligatoire' => false
```

```
    );
}
```

6.2 Les services non liés à une collection

Pour un plugin utilisateur de REST Factory, ces services sont contenus dans le fichier `ezrest/${prefixe}.php`. Les services sont nommés par homonymie avec le service de REST Factory en utilisant le préfixe du plugin à la place du préfixe `ezrest`.

Le plugin REST Factory fournit également un fichier « template », `ezrest/prefixe.php.template`, pour faciliter le codage.

6.2.1 Le service `api_verifier_contexte()`

Le service de REST Factory, `ezrest_api_verifier_contexte()`, appelle le service homonyme du plugin utisateur, `/${prefixe}_api_verifier_contexte()`, en lui passant par référence le bloc d'erreur initialisé avec un code 501 uniquement et attend un retour booléen, `false` si une erreur a été détectée, `true` sinon.

Il convient au plugin utilisateur, en cas d'erreur, de compléter le bloc avec le reste des index ce qui est logique étant donné que le contexte du serveur peut être très différent d'un plugin à un autre. De façon cohérente, le plugin utilisateur devra fournir les items de langue correspondants.

Le plugin SVP API fournit une telle fonction qui détermine si le mode runtime de SVP est bien désactivé. Son code est le suivant :

```
function svpapi_api_verifier_contexte(&$erreur) {

   // Initialise le retour à true par défaut.
   $est_valide = true;

   include_spip('inc/svp_phraser');
   include_spip('inc/config');
   $mode = lire_config("svp/mode_runtime", 'non');
   if (_SVP_MODE_RUNTIME
   or (!_SVP_MODE_RUNTIME and ($mode == 'oui'))) {
      $erreur['type'] = 'runtime_nok';
      $erreur['element'] = 'svp_mode_runtime';
      $erreur['valeur'] = _SVP_MODE_RUNTIME;

      $est valide = false;
   }

   return $est valide;
}
```

Conjointement, SVP API propose les items de langue suivants dans le fichier `lang/svpapi_fr.php` :

```
'erreur_501_runtime_nok_message' => 'Le serveur est actuellement en mode « SVP runtime »
incompatible avec le service REST SVP.',
'erreur_501_runtime_nok_titre'   => 'Le serveur SVP n\'est pas correctement configuré',
```

6.2.2 Le service `reponse_informer_plugin()`

Le service de REST Factory, `ezrest_reponse_informer_plugin()`, appelle le service homonyme du plugin utisateur, `/${prefixe}_ reponse_informer_plugin ()`, en lui passant le bloc fournisseur initialisé avec les informations de base du plugin et attend en retour ce même bloc mis à jour.

Le plugin SVP API fournit un telle fonction dans le but d'ajouter au bloc le schéma de données du plugin SVP car lui-même n'a aucun schéma et s'appuie uniquement sur SVP. Le code est le suivant :

```
function svpapi_reponse_informer_plugin($contenu) {

   // Récupération du schéma de données SVP.
   include_spip('inc/filtres');
   $informer = charger_filtre('info plugin');
   $schema = $informer('svp', 'schema', true);

   $contenu['fournisseur']['schema'] = "${schema} (SVP)";

   return $contenu;
}
```

6.2.3 Le service `reponse_expliquer_erreur()`

La gestion des erreurs permet déjà de personnaliser les erreurs au travers des services spécifiques au plugin utilisateur et de l'utilisation éventuel du paramètre extra. Cela est en général suffisant pour la plupart des plugins. Néanmoins, si il s'avère nécessaire d'aller plus loin, REST Factory permet de modifier le bloc d'erreur après qu'il ait été complètement finalisé.

Dans ce cas, le plugin utilisateur est en mesure de modifier tout ou partie des éléments du bloc en respectant bien entendu la logique des codes de réponse HTTP. Pour ce faire, le service de REST Factory, `ezrest_reponse_expliquer_erreur()`, appelle le service homonyme du plugin utilisateur, `/${prefixe}_reponse_expliquer_erreur()`, en lui passant le bloc fournisseur et attend en retour ce même bloc mis à jour.

6.3 Les services liés à une collection

Pour un plugin utilisateur de REST Factory, ces services sont contenus dans le fichier `ezrest/${prefixe}.php`. Les services sont nommés en utilisant la collection systématiquement comme préfixe de la fonction et, le nom du critère voire de l'identifiant de la ressource suivant le cas pour compléter celui du service.

Le plugin REST Factory fournit également un fichier « template », `ezrest/prefixe.php.template`, pour faciliter le codage.

6.3.1 Le service `collection_verifier()`

Le service de REST Factory, `ezrest_collection_verifier()`, appelle le service du plugin utilisateur, `/${collection}_verifier()`, où `$collection` est le nom de la collection, en lui passant par référence le bloc d'erreur complètement initialisé avec un code 400 et attend un retour booléen, `false` si une erreur a été détectée, `true` sinon.

Il convient au plugin utilisateur, en cas d'erreur, de compléter le bloc avec un éventuel paramètre extra et de fournir uniquement l'item de langue correspondant au message de détail.

Cette fonction est rarement proposée par le plugin utilisateur.

6.3.2 Le service `collection_verifier_filtre()`

Le service de REST Factory, `collection_verifier_filtre()`, appelle pour chaque filtre le service du plugin utilisateur correspondant, `/${collection}_verifier_filtre_${critere}()`, où `$collection` est le nom de la collection et `$critere` le nom du critère du filtre. Le service de REST Factory passe par référence le bloc d'erreur initialisé avec un code 400, l'élément (le critère) et sa valeur et attend un retour booléen, `false` si une erreur a été détectée, `true` sinon.

Il convient au plugin utilisateur, en cas d'erreur, de compléter le bloc avec le type ce qui lui permet ensuite de fournir les items de langue correspondants aux textes du titre et du message de détail.

Le plugin SVP API fournit une telle fonction pour le filtre compatible_spip qui détermine si le fomat de la valeur est bien une version, une branche ou une liste de branches. Son code est le suivant :

```
function plugins_verifier_filtre_compatible_spip($valeur, &$erreur) {

   $est_valide = true;

   if (!preg_match('#^((?:\d+)(?:\.\d+){0,2})(?:,(\d+\.\d+)){0,}$#', $valeur)) {
      $est valide = false;
      $erreur['type'] = 'critere_compatible_spip_nok';
   }

   return $est_valide;
}
```

6.3.3 Le service `collection_verifier_ressource()`

Le service de REST Factory, `collection_verifier_ressource()`, appelle le service du plugin utilisateur, `/${collection}_verifier_ressource_${champ}()`, où `$collection` est le nom de la collection et `$champ` le nom du champ de la table identifiant la ressource. Le service de REST Factory passe par référence le bloc d'erreur complètement initialisé avec un code 400 et attend un retour booléen, `false` si une erreur a été détectée, `true` sinon.

Il convient au plugin utilisateur, en cas d'erreur, de compléter le bloc avec un éventuel paramètre extra et de fournir uniquement l'item de langue correspondant au message de détail.

Le plugin SVP API fournit une telle fonction pour la ressource plugin identifiée par son préfixe qui vérifie l'existence du préfixe dans la base. Son code est le suivant :

```
function plugins_verifier_ressource_prefixe($prefixe) {

   $est valide = true;

   // On teste si le préfixe est syntaxiquement correct pour éviter un accès SQL dans ce cas.
   if (!preg_match('#^(\w){2,}$#', strtolower($prefixe))) {
      $est_valide = false;
   } else {
      // On vérifie ensuite si la ressource est bien un plugin fourni par un dépôt
      // et pas un plugin installé sur le serveur uniquement.
      include_spip('inc/svp_plugin');
      if (!plugin_lire($prefixe)) {
         $est valide = false;
      }
   }

   return $est_valide;
}
```

6.4 Les pipelines `post_ezcollection` et `post_ezressource`

Une fois les données récupérées par l'intermédiaire des services `ezrest_collectionner()` ou `ezrest_ressourcer()`, REST Factory appelle un pipeline, respectivement `post_ezcollection` et `post_ezressource`, afin de permettre au plugin utilisateur de compléter les données.

Le plugin REST Factory fournit également un fichier « template », `prefixe_pipelines.php.template`, pour faciliter le codage de ces fonctions.

Le plugin SVP Typologie utilise le fichier `svptype_pipelines.php` pour coder la fonction `svptype_post_ezcollection()` dont le but est de rajouter la catégorie et les tags éventuels de chaque plugin retourné. Il en est de même pour la ressource plugin en utilisant le pipeline `post_ezressource`.

```
function svptype_post_ezcollection($flux) {
   $collection = $flux['args']['collection'];
   if ($collection == 'plugins') {
      include_spip('inc/config');
      $configurations_typologie = lire_config('svptype/typologies', array());
      include_spip('inc/svptype_type_plugin');
      foreach ($configurations_typologie as $_typologie => $_configuration) {
         foreach ($flux['data'] as $_prefixe => $_plugin) {
            $affectations = type_plugin_repertorier_affectation(
               $ typologie,
               array('prefixe' => $_prefixe)
            );
            if ($ configuration['max affectations'] == 1) {
               $affectations = array shift($affectations);
               $flux['data'][$ prefixe][$ typologie] = $affectations['identifiant mot'];
            } else {
               $flux['data'][$_prefixe][$_typologie] = array_column($affectations,
'identifiant_mot');
            }
         }
      }
   }
   return $flux;
}
```

7. REGLES DE CODAGE

7.1 Nommage des fonctions

Le nommage des fonctions REST de REST Factory est imposé par le plugin Serveur HTTP abstrait. Les fonctions de service spécifiques à REST Factory suivent des règles strictes qui simplifient l'identification de l'objet et de l'action appliquée et qui dépendent de la typologie de service. Ces règles sont expliquées au chapitre 6.

7.2 Arguments et variables standardisés

Les services personnalisables de REST Factory possèdent à minima l'argument `$plugin`. Cet argument `$plugin` est toujours le **premier** argument du prototype des fonctions d'API. C'est toujours le préfixe du plugin qui doit être utilisé.

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

Par exemple, l'argument `$requete` désigne toujours l'objet Request fourni par le plugin Serveur HTTP abstrait au travers de la librairie HTTPFoundation. L'argument `$response` désigne de même l'objet Response de HTTPFoundation.

L'argument `$erreur` désigne toujours le bloc d'erreur tel que décrit dans le paragraphe 5.3.2. De même, les arguments `$collection`, `$ressource` et `$configuration` désignent respectivement le nom d'une collection, l'identifiant d'une ressource et le bloc de configuration d'une collection.

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

8. GRAPHE D'APPEL FONCTIONS REST – SERVICES

FONCTIONS REST – SERVICE	
http_ezrest_get_index_dist	ezrest_reponse_initialiser_contenu ezrest_indexer ezrest_reponse_construire
http_ezrest_get_collection_dist	ezrest_reponse_initialiser_contenu ezrest_collection_verifier ezrest_reponse_informer_plugin ezrest_api_verifier_contexte ezrest_collection_verifier_filtre ezrest_collectionner ezrest_reponse_expliquer_erreur ezrest_reponse_construire
http_ezrest_get_ressource_dist	ezrest_reponse_initialiser_contenu ezrest_collection_verifier ezrest_reponse_informer_plugin ezrest_api_verifier_contexte ezrest_collection_verifier_ressource ezrest_ressourcer ezrest_reponse_expliquer_erreur ezrest_reponse_construire
http_ezrest_erreur_dist	ezrest_reponse_initialiser_contenu ezrest_reponse_construire