Référence SPIP/X/2020.002 – Ed. 1.2

GUIDE
CONCEPTION DU PLUGIN CHECK FACTORY



FICHE D'IDENTIFICATION	
Rédacteur	Eric Lupinacci
Projet	SPIP
Étude	Conception du plugin Check Factory
Nature du document	Guide
Date	23/02/2024
Nom du fichier	Guide - Le plugin Check Factory.docx
Référence	SPIP/X/2020.002 – Ed. 1.2
Dernière mise à jour	24/03/2024 15:32:27
Langue du document	Français
Nombre de pages	26

TABLE DES MATIERES

1.	**INTRODUCTION**	**5**
2.	**CONCEPTS**	**5**
2.1	**LES TYPES DE CONTROLE**	**5**
2.1.1	DEFINITION	5
2.1.2	COMPOSANTS D'UN TYPE DE CONTROLE	5
2.1.3	FONCTIONS D'EXECUTION ET DE CORRECTION	6
2.1.4	CAS D'UTILISATION	6
2.2	**LES CONTROLES**	**6**
2.3	**LES OBSERVATIONS**	**7**
2.4	**LES DASHBOARDS**	**7**
3.	**PERIMETRE DU PLUGIN CHECK FACTORY**	**8**
3.1	**L'API DE GESTION DES TYPES DE CONTROLE**	**8**
3.2	**L'API DE GESTION DES OBSERVATIONS**	**9**
3.3	**L'API DE GESTION DES DASHBOARDS**	**10**
3.4	**L'INTERFACE UTILISATEUR DANS L'ESPACE PRIVE**	**10**
4.	**FONCTIONNEMENT DE CHECK FACTORY**	**11**
4.1	**LE CHARGEMENT DES CONFIGURATIONS**	**11**
4.1.1	LE FICHIER YAML D'UN TYPE DE CONTROLE	11
4.1.2	LE FICHIER YAML OU JSON D'UN DASHBOARD	12
4.1.3	LES PIPELINES	13
4.2	**LE PROCESSUS D'EXECUTION**	**13**
4.2.1	EXECUTION D'UN AFFICHAGE PARAMETRE	13
4.2.2	EXECUTION D'UNE FONCTION NON OBSERVABLE	13
4.2.3	EXECUTION D'UNE FONCTION OBSERVABLE	14
4.3	**LA GESTION DES OBSERVATIONS**	**14**
4.3.1	LEVER DES OBSERVATIONS	14
4.3.2	CLOTURER DES OBSERVATIONS	15
4.4	**L'INTERFACE D'EXECUTION ET DE SUIVI DES TYPES DE CONTROLE**	**15**
4.4.1	INTERFACE POUR UN ETAT	16
4.4.2	INTERFACE POUR UN AFFICHAGE PARAMETRE	16
4.4.3	INTERFACE POUR UNE FONCTION NON OBSERVABLE	17
4.4.4	INTERFACE POUR UNE FONCTION OBSERVABLE	17
4.5	**L'INTERFACE D'ADMINISTRATION**	**19**
4.5.1	L'ADMINISTRATION DES DASHBOARDS	19
4.5.2	L'ADMINISTRATION DES TYPES DE CONTROLE	19

5.	**DONNEES DE CHECK FACTORY**	**21**
5.1	**LA STRUCTURE DES DONNEES**	**21**
5.1.1	LES TYPES DE CONTROLE	21
5.1.2	LES CONTROLES	22
5.1.3	LES OBSERVATIONS	23
5.1.4	LES DASHBOARDS	24
5.2	**LES ESPACES DE STOCKAGE**	**24**
6.	**MISE EN PLACE D'UN DASHBOARD UTILISATEUR**	**25**
6.1	**MISE EN PLACE D'UN ETAT**	**25**
6.2	**MISE EN PLACE D'UN AFFICHAGE PARAMETRE**	**25**
6.3	**MISE EN PLACE D'UNE FONCTION NON OBSERVABLE**	**25**
6.4	**MISE EN PLACE D'UNE FONCTION OBSERVABLE**	**25**
7.	**REGLES DE CODAGE**	**26**
7.1	**NOMMAGE DES FONCTIONS**	**26**
7.2	**ARGUMENTS ET VARIABLES STANDARDISES**	**26**
7.3	**TYPAGE DES ARGUMENTS**	**26**

1. INTRODUCTION

Ce document a pour but de décrire les principes de base et les éléments de conception du plugin Check Factory (version 1.1.8 et ultérieures) dont l'objectif est de fournir des objets et des API génériques de gestion de contrôles et de leurs observations et de construction de dashboards de suivi.

Le plugin Check Factory fournit également une interface utilisateur limitée à l'administration des contrôles et des dashboards.

2. CONCEPTS

2.1 Les types de contrôle

2.1.1 Définition

Les types de contrôle sont des **fonctions** – au sens large – qui réalisent des **vérifications**, des **actions** ou des **états** et produisent des résultats sous forme d'**observations,** de **sorties physiques** (fichiers) et/ou d'**affichages**. Un type de contrôle peut concerner des objets ou des groupes d'objets du site sur lequel est installé le plugin Check Factory ou d'un site distant accessible, par exemple, au travers d'un service Web.

Un type de contrôle est généralement associé à un fichier YAML, `$type_controle.yaml`, qui décrit l'ensemble de ses caractéristiques. Son identifiant est alors le nom du fichier YAML sans extension (par exemple, `contrib_rubrique_prefixe`). L'identifiant doit être composé de caractères acceptables pour un nom de fonction PHP, donc en particulier éviter le tiret. Les fichiers YAML doivent être stockés dans le dossier `ezcheck/controles/` d'un plugin utilisateur.

Il est aussi possible de décrire les caractéristiques d'un type de contrôle via le pipeline `eztypecontrole_declarer` (voir paragraphe **Error! Reference source not found.**).

2.1.2 Composants d'un type de contrôle

Un type de contrôle est défini par :

- son **identification**, à savoir, son identifiant unique, son nom, sa description et son logo ;
- une éventuelle **fonction PHP d'exécution** du contrôle pour laquelle il est possible de préciser le nom, la localisation (fichier PHP) et des arguments qui seront fournis via un formulaire.
- une liste éventuelle d'**anomalies** (observation de type anomalie) pouvant être produites par le contrôle et pour lesquelles il est possible d'associer une fonction de correction automatique ;
- un **affichage complémentaire** définit par un squelette HTML auquel il est possible d'adjoindre un contexte fixe mais aussi une liste de paramètres fournis via un formulaire ;
- une liste de saisies de **paramètres** attachés soit à la fonction d'exécution, soit à l'affichage complémentaire.

Ces éléments font partie de la configuration du type de contrôle. La plupart sont facultatifs. Un type de contrôle sans fonction d'exécution est appelé un **état** : il se contente d'un affichage.

2.1.3 Fonctions d'exécution et de correction

Si le type de contrôle est basé sur une fonction PHP, celle-ci est identifiée par son nom et le fichier dans lequel elle est définie. Il en va de même pour les fonctions de corrections.

Par défaut, la fonction d'exécution porte un nom qui coïncide avec l'identifiant de son type de contrôle. Dans ce cas, il suffit d'indiquer la localisation de la fonction. Il est possible d'utiliser une fonction externe en précisant son nom et sa localisation.

Les fonctions de correction possèdent un nom imposé composé de l'identifiant du type de contrôle et de l'identifiant de l'anomalie à corriger.

2.1.4 Cas d'utilisation

On distingue quatre principaux cas d'utilisation des types de contrôle, à savoir :

- Un affichage. Ce type de contrôle ne nécessite qu'un squelette d'affichage auquel il est possible de fournir des variables de contexte valorisées et des paramètres saisis au travers d'un formulaire dédié. Un affichage sans paramètre est appelé un **état** versus un **affichage paramétré**.
- Une **fonction non observable** qui permet de lancer une API existante quelconque. Il est nécessaire de préciser le nom de la fonction à exécuter, sa localisation dans le path de SPIP et de fournir les éventuels arguments au travers d'un formulaire. Par contre, il n'est pas possible d'utiliser les observations.
- Une **fonction observable** qui lance la fonction homonyme du type de contrôle dans laquelle il est possible d'insérer le code de vérification ou d'action souhaité et de générer éventuellement des observations à partir de l'API fournie par Check Factory. Il est nécessaire de préciser à minima la localisation de la fonction et, si besoin, de fournir dans un tableau d'options une liste de paramètres saisis dans un formulaire dédié.

Les fonctions observables et non observables peuvent aussi afficher un squelette d'affichage complémentaire.

2.2 Les contrôles

Un contrôle est une **« instance d'un type de contrôle » exécutée à un instant t** suite à une demande d'un utilisateur autorisé ou à l'échéance d'un CRON (non disponible dans la version actuelle).

Outre son type, le contrôle possède un **statut d'exécution** qui rend compte du déroulement correct ou pas de l'exécution. Ce statut n'a rien à voir avec les observations éventuellement générées pendant l'exécution du contrôle.

Tout contrôle possède un identifiant unique `id_controle`. Il est aussi possible d'identifier de façon unique un contrôle avec le type de contrôle auquel il est associé, la date d'exécution et l'auteur, mais cette identification est peu utilisée.

Tout type de contrôle possédant une fonction d'exécution ou à minima un affichage complémentaire avec des paramètres génère des contrôles. Seuls les états ne génèrent pas de contrôle car ils ne font qu'afficher un squelette HTML.

2.3 Les observations

Les observations sont des **logs qui consignent des résultats de l'exécution du contrôle**. Les observations ne sont générées que par la fonction d'exécution PHP au travers de l'API fournie par le plugin Check Factory. Leur utilisation est néanmoins facultative.

Une observation possède un identifiant unique `id_observation` mais aussi un code alphanumérique unique au sein d'un type de contrôle donné. Un type de contrôle peut générer différents types d'observation au travers de ses vérifications. Une observation est associée à un contrôle et concerne un objet au sens large (objet SPIP ou tout autre élément observable).

Une **anomalie** est une observation particulière qui possède un statut ouverte ou fermée et qui peut être soit acquittée manuellement soit corrigée automatiquement par l'exécution d'une fonction spécifique de correction : elle passe alors dans l'état fermée. Le suivi des anomalies est une fonction essentielle du plugin Check Factory.

2.4 Les dashboards

Un dashboard est une page HTML qui organise les types de contrôle, permet leur exécution, affiche les résultats et facilite la gestion des anomalies. Il est décrit par un fichier de configuration YAML ou JSON dont le nom coïncide avec l'identifiant du dashboard lui-même.

Les fichiers YAML ou JSON doivent être stockés dans le dossier `ezcheck/dashboards/` d'un plugin utilisateur.

Il est possible de créer autant de dashboard que l'on souhaite et d'y associer tous les types de contrôle nécessaires. Pour simplifier la présentation, les types de contrôle sont organisés par groupe (onglets).

3. PERIMETRE DU PLUGIN CHECK FACTORY

Check Factory propose plusieurs API :

- La gestion des types de contrôle, à savoir, le chargement des fichiers YAML (ou du pipeline), leur stockage, la lecture des informations stockées et l'exécution des contrôles ;
- La gestion des dashboards, à savoir, le chargement, leur stockage et la lecture des informations de configuration ;
- La gestion des observations et, en particulier, la création des anomalies et leurs évolutions de statut jusqu'à la clôture ainsi que la gestion des corrections.

Un plugin utilisateur de Check Factory se contente en général de l'API des observations qu'il utilise dans ses fonctions de contrôle. Le reste des API n'est utile que si le plugin souhaite modifier la présentation des dashboards ou la gestion par défaut des concepts de Check Factory.

Check Factory propose également une interface minimale dans l'espace privé pour recharger les dashboards et les types de contrôle et aussi activer ou désactiver temporairement des types de contrôle.

3.1 L'API de gestion des types de contrôle

La gestion des types de contrôle consiste à stocker les descriptions en base de données et à permettre leur lecture, leur mise à jour et leur exécution au travers d'un contrôle.

API TYPES DE CONTROLES : INC/EZCHECK_TYPE_CONTROLE.PHP	
type_controle_charger	Charge ou recharge les descriptions des types de contrôle à partir des fichiers YAML ou du pipeline `eztypecontrole_declarer`. Les fichiers YAML sont recherchés dans un répertoire relatif `ezcheck/controles/`. La fonction optimise le chargement en effectuant uniquement les traitements nécessaires en fonction des modifications, ajouts et suppressions des types de contrôle identifiés en comparant les md5 des descriptions complètes.
type_controle_lire	Retourne, pour un type de contrôle, la description complète ou une liste de champs donnée. Les champs de type tableau sont fournis désérialisés.
type_controle_repertorier	Renvoie une liste de types de contrôle éventuellement filtrée sur certains champs. Les données sont renvoyées brutes sous forme d'un tableau indexé par l'identifiant de chaque type de contrôle. Il est possible de récupérer toute la description ou un seul champ pour chaque type de contrôle.

type_controle_executer	Exécute une fonction de contrôle d'un type de contrôle donné. Si la fonction n'existe pas c'est que l'exécution ne sert qu'à enregistrer les paramètres du formulaire servant uniquement à l'affichage d'un squelette. Dans tous les cas, cette fonction crée un contrôle du type concerné.
type_controle_identifier_liaison	Renvoie l'identifiant du dashboard ou du groupe auquel est rattaché le type contrôle ou chaine vide sinon.

Check Factory propose les balises `#TYPE_CONTROLE_DASHBOARD` et `#TYPE_CONTROLE_GROUPE` qui fournissent respectivement l'identifiant du dashboard ou du groupe auquel est rattaché le type de contrôle. Ces balises sont une encapsulation de la fonction d'API `type_controle_identifier_liaison()`.

Les prototypes sont les suivants : `#TYPE_CONTROLE_DASHBOARD{type_controle}` et `TYPE_CONTROLE_GROUPE{type_controle}`.

3.2 L'API de gestion des observations

La gestion des observations consiste à gérer leur cycle de vie, de leur ouverture à leur clôture.

API OBSERVATIONS : INC/EZCHECK_OBSERVATION.PHP	
observation_ajouter	Crée une observation avec tous ses paramètres en base de données. Une observation de type anomalie se voit affecté d'un statut 'ouverte'.
observation_repertorier	Renvoie une liste d'observations éventuellement filtrée sur certains champs. Les données sont renvoyées brutes sous forme d'un tableau indexé par l'identifiant de chaque observation. Il est possible de récupérer toute la description ou un seul champ pour chaque observation.
observation_cloturer	Effectue les traitements adéquats pour clore une observation, à savoir, acquitte une anomalie, corrige une anomalie ou supprime une observation. Acquitter et supprimer sont des actions qui ne requièrent que la mise à jour du statut. L'action corriger elle nécessite l'appel à la fonction de correction configurée.

3.3 L'API de gestion des dashboards

La gestion des dashboards consiste essentiellement à charger leur description et à les restituer sur demande au travers d'une fonction d'API ou d'une balise.

API DASHBOARDS : INC/EZCHECK_DASHBOARD.PHP	
dashboard_charger	Charge ou recharge la configuration des dashboards à partir de leur fichier YAML ou JSON. Les fichiers YAML ou JSON sont recherchés dans un répertoire relatif `ezcheck/dashboards/`. La fonction compile les dashboards dans un cache unique sécurisé.
dashboard_lire	Retourne la description complète du dashboard. Les champs textuels peuvent subir un traitement typo si demandé.
dashboard_repertorier	Renvoie l'information brute demandée pour l'ensemble des dashboards ou toute les descriptions si aucun champ n'est explicitement demandé. Les données sont renvoyées sous forme d'un tableau indexé par l'identifiant de chaque dashboard.
dashboard_contextualiser	Renvoie la configuration complète d'un dashboard ainsi que ainsi que des informations supplémentaires provenant de l'environnement de la page dashboard (groupe et type de contrôle à afficher par défaut).

Territoires propose aussi des balises utilisables dans l'espace public, à savoir :

- `#DASHBOARD` qui fournit, dans un tableau, la configuration complète d'un dashboard donné ou de tous les dashboards.
- `DASHBOARD_CONTEXTE` qui fournit, dans un tableau, la configuration complète d'un dashboard ainsi que des informations supplémentaires provenant de l'environnement de la page dashboard. Cette balise est une encapsulation de l'API `dashboard_contextualiser()`.

Les prototypes des balises sont :

`#DASHBOARD{[dashboard]}` et `#DASHBOARD_CONTEXTE{dashboard[, groupe, type_controle]}`.

3.4 L'interface utilisateur dans l'espace privé

Le plugin Check Factory offre une interface utilisateur dans l'espace privé uniquement qui autorise l'administration des types de contrôle et des dashboards, ainsi que l'exécution et le suivi des contrôles.

L'interface est concentrée dans 3 pages `ezcheck&vue=type_controle`, `ezcheck&vue=dashboard` pour l'administration et `dashboard` pour l'exécution et le suivi des types de contrôles.

Les pages sont décrites dans les paragraphes 4.2 et 4.3.

4. FONCTIONNEMENT DE CHECK FACTORY

4.1 Le chargement des configurations

Que ce soit les types de contrôle ou les dashboards, il est nécessaire de procéder à leur chargement avant leur utilisation. Ce chargement a lieu soit automatiquement lors de l'affichage de la page d'administration des plugins (donc à l'installation de tout plugin) ou manuellement par l'intermédiaire des boutons présents dans les pages d'administration des types de contrôles et des dashboards.

La configuration des types de contrôle ou des dashboards est chargée, soit à partir des fichiers YAML (ou aussi JSON pour les dashboards), soit au travers des pipelines de déclaration `eztypecontrole_declarer` et `ezdashboard_declarer`. Les deux modes peuvent se compléter étant donné que la fonction de chargement appelle les pipelines suite à la lecture des fichiers.

4.1.1 Le fichier YAML d'un type de contrôle

La structure des fichiers YAML est composée de quatre bloc distincts, certains facultatifs suivant le cas d'utilisation. Le tableau suivant explicite la structure du fichier et les valeurs par défaut.

TYPE DE CONTROLE - CONFIGURATION YAML		
Données d'identification		
nom	Nom littéral du type de contrôle en texte ou en item de langue.	`Id du type de contrôle`
description	Description facultative du type de contrôle en texte ou en item de langue.	`''`
icone	Chemin de l'icône limité à son nom de fichier car recherché dans le thème privé.	`controle_defaut-24.svg`
necessite	Tableau des préfixes des plugins nécessaires au fonctionnement du type de contrôle (sauf ezcheck)	`[]`
Données d'exécution – bloc : ***execution***		
include	Indique le chemin relatif du fichier dans le path, sans extension, dans lequel est incluse la fonction. La syntaxe admise est : • `/nom_fichier` : indique que le fichier est dans le répertoire relatif par défaut, soit `ezcheck/controles/`. • `dir_relatif/nom_fichier` : indique qu'il faut utiliser le chemin relatif fourni tel que. • `nom_fichier` : indique que le fichier est à la racine d'un dossier du path (cas particulier du précédent).	`''`

fonction	Nom de la fonction d'exécution du contrôle si celle-ci diffère de l'id du type de contrôle	`''`
parametres	Liste ordonnée des paramètres de la fonction fournie sous la forme d'une liste de saisies. Il est possible de passer des paramètres « fixes » avec une saisie de type hidden.	`[]`
Données d'observation – bloc : ***anomalies***		
include	Indique le chemin relatif du fichier dans le path, sans extension, dans lequel sont incluses les fonctions de correction automatique des anomalies (même format que l'include du bloc 'execution').	`''`
acquitter	Liste des identifiants d'anomalie que l'on peut acquitter manuellement.	`[]`
corriger	Liste des identifiants d'anomalie que l'on peut corriger automatiquement via une fonction PHP.	`[]`
Données d'affichage complémentaire – bloc : ***affichage***		
squelette	Indique le chemin relatif du fichier squelette dans le path, sans extension (toujours HTML).	`''`
contexte	Liste de variables supplémentaires de contexte fournie au squelette. Ces variables sont valorisées comme des entiers, chaines ou tableaux. Les items de langue sont acceptés.	`[]`
paramètres	Liste de paramètres fournie au contexte du squelette et définie comme une liste de saisies.	`[]`

Check Mashup fourni un modèle YAML `ezcheck/controles/type_controle.yaml.template` qu'il est conseillé d'utiliser pour créer des types de contrôle utilisateur.

4.1.2 Le fichier YAML ou JSON d'un dashboard

La structure des fichiers YAML ou JSON est composée de deux bloc distincts obligatoires. Le tableau suivant explicite la structure du fichier et les valeurs par défaut.

TYPE DE CONTROLE - CONFIGURATION YAML		
nom	Nom littéral du dashboard en texte ou en item de langue.	`Id du dashboard`
description	Description facultative du dashboard en texte ou en item de langue.	`''`
icone	Chemin de l'icône limité à son nom de fichier car recherché dans le thème privé.	`dashboard_defaut-24.svg`
groupes	Liste des configurations de groupe.	`Cf.` (1)

Données d'organisation d'un groupe[1]		
identifiant	Identifiant unique du groupe dans le dashboard (obligatoirement non vide)	
nom	Titre du groupe qui sera affiché dans l'onglet de la page `dashboard`	`Id du groupe`
controles	Liste des identifiants de contrôle associés au groupe et qui seront affichés dans le menu de navigation	`[]`

Check Mashup fourni un modèle YAML `ezcheck/controles/dashboard.yaml.template` qu'il est conseillé d'utiliser pour créer des dashboards utilisateur.

4.1.3 Les pipelines

L'utilisation des pipelines est expliquée dans le chapitre 6. Les pipelines reçoivent en entrée la liste des types de contrôles ou des dashboards sous la forme d'un tableau dont les index sont les identifiants.

Si un plugin souhaite rajouter ses types de contrôle ou ses dashboards il doit rajouter les descriptions complètes sous la forme tabulaire reproduisant exactement le résultat du décodage des fichiers YAML ou JSON. Il est donc conseillé d'utiliser en priorité les fichiers de configuration.

4.2 Le processus d'exécution

Sauf si le type de contrôle est un état - c'est-à-dire juste un affichage sans paramètre - **chaque exécution est consignée dans un objet contrôle**. L'exécution, déclenchée par le bouton « lancer l'exécution » du formulaire de la page `dashboard`, se passe en premier lieu dans la fonction de service `type_controle_executer()` dont le prototype est le suivant :

```
function type_controle_executer(string $id_type_controle, int $id_auteur, array $options) : bool
```

Cette fonction est activée pour une fonction de contrôle, observable ou pas et aussi pour consigner les paramètres d'un affichage paramétré.

4.2.1 Exécution d'une fonction non observable

La fonction `type_controle_executer()` exécute une fonction d'API précisée dans la configuration du type de contrôle. Elle construit pour cela le tableau des arguments attendus qui doivent être fournis au travers d'un formulaire adéquat. Le code correspondant est le suivant :

```
// La fonction de contrôle est appelée avec sa liste d'arguments et retourne toujours ok.
include_spip($type_controle['include']);
$arguments = [];
foreach ($options['fonction'] as $_parametre) {
   $arguments[] = $_parametre;
}
call_user_func_array($fonction, $arguments);
$retour = true;
```

L'exécution est consignée dans un contrôle et renvoie toujours `true`.

4.2.2 Exécution d'une fonction observable

La fonction `type_controle_executer()` exécute cette fois une fonction dont le nom est celui du type de contrôle et dont le prototype est identique au sien à l'exception du retour qui devient une chaine.

Le tableau associatif `$options` permet de passer des paramètres : ceux-ci sont saisis dans un formulaire dédié avant le lancement du contrôle. Le code correspondant est le suivant :

```
// -- la fonction de contrôle peut détecter et créer des observations et retourne une chaine //    vide ou un identifiant d'erreur suivant que le traitement s'est bien déroulé ou pas.
//    Aucune information n'est fournie par la fonction sur le nombre d'anomalies détectées.
include_spip($type_controle['include']);
$erreur = $fonction($id_controle, $id_auteur, $options['fonction']);
if (!$erreur) {
   // Tout s'est bien passé
   $retour = true;
} else {
   // On met à jour l'état d'exécution du contrôle en erreur
   $maj_controle['etat_execution'] = "exec_nok_{$erreur}";
}
```

La fonction d'exécution retourne une chaine, vide si aucune erreur ne s'est produite, ou avec un identifiant d'erreur dans le cas contraire. L'exécution est consignée dans un contrôle et renvoie `true` ou `false` si une erreur s'est produite.

4.3 La gestion des observations

Un type de contrôle « fonction observable » est en général utilisé pour lever des observations et gérer leur cycle de vie que ce soit des anomalies ou pas. L'API observation de Check Factory en fournit les outils.

4.3.1 Lever des observations

Dans une fonction observable il est possible de lever des observations pour révéler des problèmes liés à l'objet – au sens large – concerné. Pour cela, il est nécessaire d'utiliser dans le code la fonction `observation_ajouter()` dont le prototype est le suivant :

```
/**
 * Ajoute une observation à la table `spip_observations`.
 *
 * @param bool      $est_anomalie Indique si l'observation est une anomalie ou pas
 * @param int       $id_controle  Identifiant numérique du contrôle
 * @param null|array $valeurs     Tableau des valeurs de certains champs de l'observation
 *
 * @return int Identifiant numérique de l'observation crée ou 0 si erreur.
 */
function observation_ajouter(bool $est_anomalie, int $id_controle, ?array $valeurs = []): int
```

Les observations sont toujours associées à une instance de contrôle et peut être une anomalie ou un simple log. L'argument `$valeurs` permet de fournir tout ou partie des champs de l'objet observation à créer.

4.3.2 Clôturer des observations

Le cycle de vie d'une observation est très simple et s'articule selon le type d'observation autour des statuts ouverte, fermée et poubelle.

Un simple log est créé avec le statut fermé et ne peut être que mis à la poubelle.

Une anomalie est créée avec le statut ouverte et doit être fermée avant d'être mise à la poubelle. Suivant la configuration, l'action de fermeture d'une anomalie consiste soit à l'acquitter soit à déclencher une correction automatique par le biais d'une fonction utilisateur. Ces actions sont lancées via des boutons idoines disponibles dans la liste des anomalies.

Pour créer une correction automatique d'anomalie, il suffit de coder une fonction dont le nom est composé de l'identifiant du type de contrôle et de l'identifiant de l'anomalie et dont le prototype est toujours le suivant :

```
function $id_type_controle_$id_anomalie(int $id_observation, int $id_auteur) : string
```

La fonction peut renvoyer si nécessaire un code d'erreur interprétable.

4.4 L'interface d'exécution et de suivi des types de contrôle

La page `dashboard` est la page d'exécution, pour un dashboard donné, des types de contrôle et de gestion des anomalies pour un type de contrôle donné. Elle est construite automatiquement à partir des informations collectées dans les fichiers de configuration et au travers des pipelines de chargement des dashboards et des types de contrôles si ceux-ci sont utilisés.

La page est organisée avec un menu d'onglets, un onglet représentant un groupe de types de contrôle, un menu des types de contrôle du groupe sélectionné et une partie centrale fournissant l'ensemble des éléments constitutifs du contrôle exécuté ou à exécuter.

La partie centrale est composée de différents éléments distincts concernant l'exécution du contrôle et les résultats de l'exécution. Les éléments présentés diffèrent selon la configuration du type de contrôle mais on distinguer :

- Le **formulaire de lancement de l'exécution**, qui affiche toujours une explication sur le contrôle et, éventuellement, une liste de saisies nécessaires pour exécuter le contrôle et un bouton de lancement de l'exécution d'un nouveau contrôle ;
- Le **dernier contrôle** effectué, avec en particulier, la date et le statut d'exécution ;
- La **liste des contrôles précédents** dont les anomalies n'ont pas toutes été résolues (dépend du type de contrôle et du contexte d'exécution) ;
- La liste des **anomalies ouvertes** (dépend du type de contrôle et du contexte d'exécution);
- La liste des **anomalies fermées** et non encore effacées (dépend du type de contrôle et du contexte d'exécution) ;
- Un **affichage complémentaire** optionnel fournissant des résultats divers sur l'exécution du contrôle au travers d'un squelette.

4.4.1 Interface pour un état

L'interface utilisateur d'un état est extrêmement sommaire : le formulaire est réduit à sa plus simple expression (description du type de contrôle) car aucune saisie n'est proposée et le bouton d'exécution n'est pas visible car un état ne fait qu'afficher un squelette. Sous le formulaire, le squelette d'affichage complémentaire est donc présent et reflète d'emblée l'état du moment pour l'objet ou les objets observés (objet au sens large).

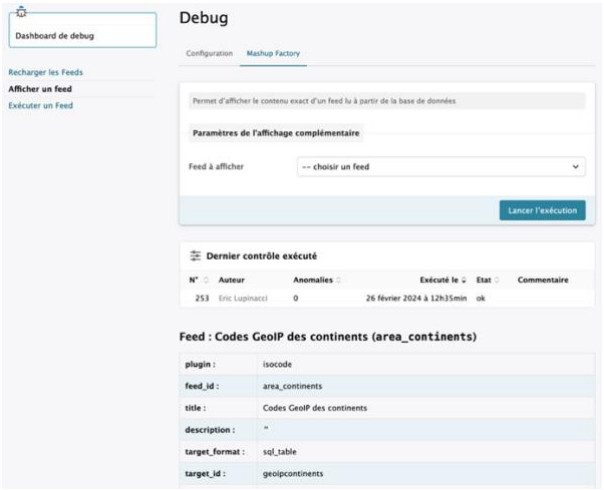
Un exemple d'état inclus dans le dashboard de SPIP-Contrib est présenté ci-dessous : il permet de vérifier les plugins sans catégorie.



4.4.2 Interface pour un affichage paramétré

L'interface utilisateur d'un affichage paramétré est similaire à celui d'un état si ce n'est que le formulaire comporte des saisies utiles à l'affichage complémentaire ainsi que le bouton « Lancer l'exécution ». Les saisies du formulaire sont incluses dans un fieldset nommé « Paramètres de l'affichage complémentaire ». Sous le formulaire est affiché le dernier contrôle exécuté ainsi que le squelette d'affichage complémentaire.

L'exemple d'affichage paramétré suivant visualise la description tabulaire d'un feed du plugin Nomenclatures suite à l'exécution du type de contrôle sur le feed 'Codes GeoIP des continents (area_continents)'.

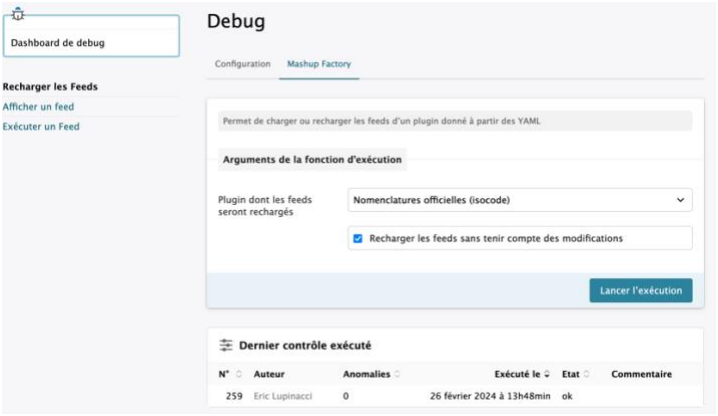


4.4.3 Interface pour une fonction non observable

L'interface utilisateur d'une fonction non observable fait aussi appel au formulaire et au bouton « Lancer l'exécution ». Si la fonction requiert des arguments, ceux-ci doivent être fournis au travers de saisies incluses dans un fieldset qui cette fois se nomme « Arguments de la fonction d'exécution ». Sous le formulaire est affiché le dernier contrôle exécuté ainsi qu'un éventuel squelette d'affichage complémentaire auquel il est possible de fournir des paramètres saisis inclus dans un fieldset nommé « Paramètres de l'affichage complémentaire »..

L'exemple suivant permet de lancer une fonction de l'API des feeds de Mashup Factory sans coder une seule ligne. La fonction `feed_charger()` dont le prototype est décrit ci-après est exécutée en utilisant les paramètres saisis. Aucun affichage complémentaire n'est nécessaire.

```
function feed_charger(string $plugin, ?bool $recharger = false) : array
```



4.4.4 Interface pour une fonction observable

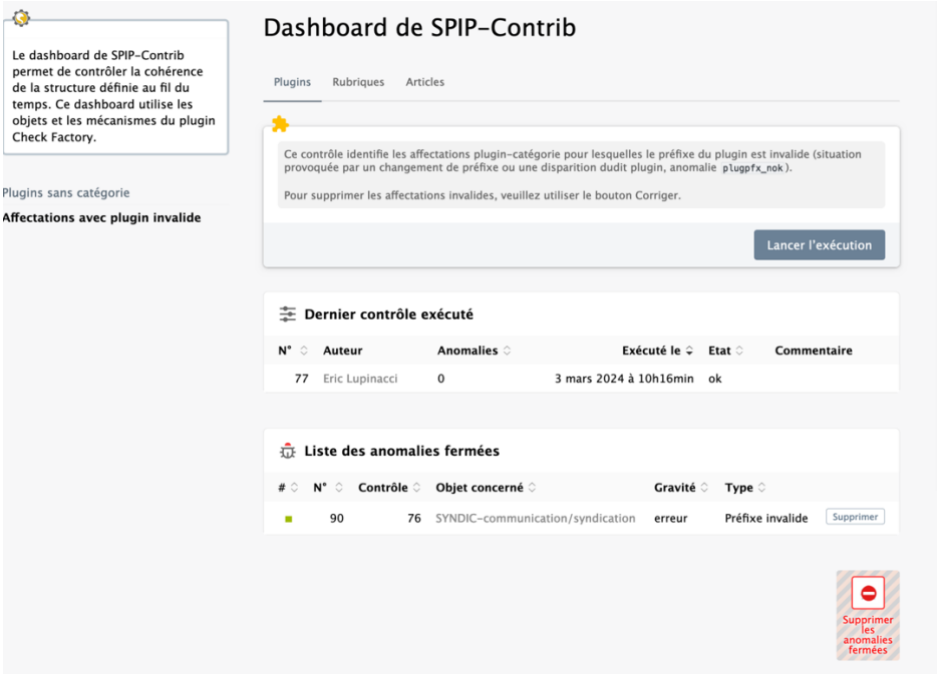
L'interface utilisateur d'une fonction observable est la plus complexe car elle permet, même si cela n'est pas obligatoire, de créer et de suivre la résolution d'observations, anomalie ou log. La fonction observable est codée spécifiquement pour le type de contrôle dont elle porte le nom et son prototype est standard comme expliqué dans le paragraphe 4.2.2. Il est possible de lui fournir des paramètres au travers du formulaire dont les saisies sont incluses dans le fieldset « Arguments de la fonction d'exécution ».

Sous le formulaire est affiché le dernier contrôle exécuté, les éventuelles observations générées ainsi qu'un éventuel squelette d'affichage complémentaire auquel il est possible de fournir des paramètres saisis inclus dans un fieldset nommé « Paramètres de l'affichage complémentaire ».

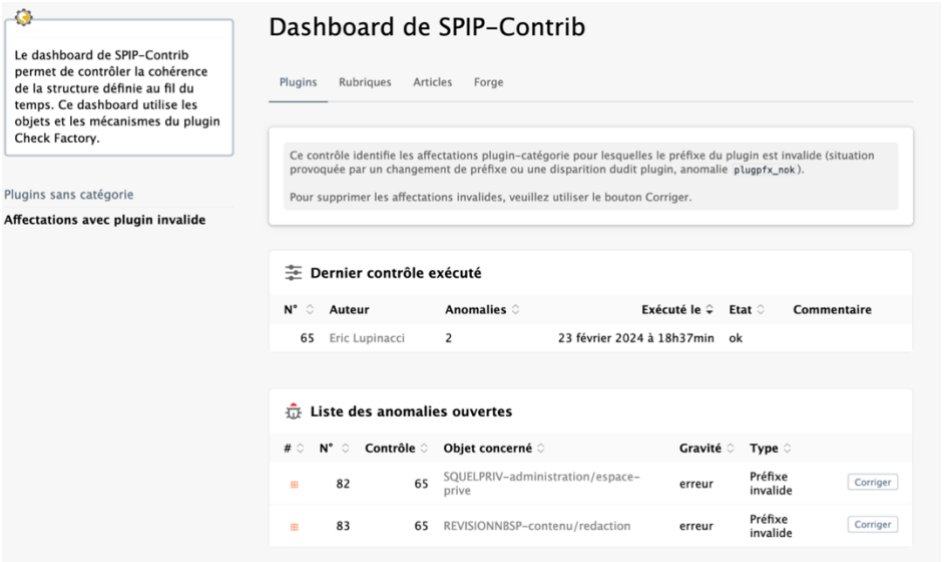
L'exemple suivant montre une page d'exécution d'un type de contrôle qui vient d'être exécuté (tableau dernier contrôle exécuté) et qui a généré deux anomalies de type « préfixe invalide » et pouvant être corrigées automatiquement (bouton Corriger) ou acquittées si l'on préfère faire la correction manuellement (bouton Acquitter).

Le bouton d'exécution du type de contrôle normalement affiché dans le formulaire est masqué tant que des anomalies sont ouvertes. Une fois traitées, il réapparaitra pour une nouvelle exécution.

Avant l'exécution du contrôle :



Après l'exécution du contrôle :



4.5 L'interface d'administration

L'interface d'administration est matérialisée par la page `ezcheck` et possède deux vues distinctes, la vue dashboard (`&vue=dashboard`) pour l'administration des dashboards et la vue type de contrôle (`&vue=type_controle`) pour l'administration des types de contrôle.

4.5.1 L'administration des dashboards

L'interface utilisateur du plugin Check Factory propose, dans l'espace privé, une page de gestion sommaire des dashboard qui permet :

- d'afficher la liste des dashboards ;
- de recharger la liste des dashboards.

Chaque dashboard est présenté avec son identifiant, son libellé et, éventuellement, sa description et son icone.



Un bouton d'action permet de recharger les dashboards en lançant l'action `action/recharger_dashboards()`.

4.5.2 L'administration des types de contrôle

L'interface utilisateur du plugin Check Factory propose, dans l'espace privé, une page de gestion sommaire des types de contrôle, `ezcheck&vue=type_controle`. Elle permet :

- d'afficher la liste des types de contrôle ;
- d'activer ou de désactiver temporairement des types de contrôle ;
- de recharger la liste des types de contrôle.

Chaque type de contrôle est présenté avec son identifiant, son libellé et, éventuellement, sa description et son icone. Si une erreur est détectée dans la configuration du type de contrôle (YAML ou pipeline), un message d'erreur est affiché dans la zone du type de contrôle.



Un bouton d'action permet de recharger les dashboards en lançant l'action `action/recharger_types_controle()`. Le panneau d'administration permet d'activer ou de désactiver un type de contrôle au travers de l'action `inverser_activite_type_controle()`. Cette action est utile si on désire provisoirement retirer un type de contrôle d'un dashboard sans en modifier sa configuration.

En outre, l'affichage de la liste des types de contrôle permet de repérer des erreurs de configuration des fichiers ou de la déclaration du pipeline. Les erreurs détectées par la fonction de chargement des types de contrôle sont les suivantes :

- **not** : le squelette d'affichage des résultats est introuvable ;
- **nof** : la fonction PHP d'exécution du contrôle est introuvable dans le fichier indiqué ;
- **noc** : la fonction PHP de correction d'une anomalie est introuvable ;
- **nop** : certains plugins nécessités par le type de contrôle, sont désactivés ;
- **noy** : le YAML du type de contrôle comporte des erreurs de syntaxe ;
- **nod** : le type de contrôle n'est affecté à aucun dashboard ;
- **nok** : le type de contrôle est mal configuré, l'erreur étant différente des précédentes.

5. DONNEES DE CHECK FACTORY

5.1 La structure des données

5.1.1 Les types de contrôle

Un type de contrôle est un objet SPIP matérialisé par un enregistrement dans la table `spip_types_controles`. Les champs de cette table sont listés ci-dessous avec leur valeur par défaut.

TABLE SPIP_TYPES_CONTROLES	
Données de base issues du fichier YAML ou du pipeline	
identifiant	Identifiant de l'objet type de contrôle au format chaine [a-z0-9_] ; correspond au nom du fichier de configuration YAML ou de l'index du tableau issu du pipeline `eztypecontrole_declarer`.
nom	Libellé du type de contrôle. Par défaut, est initialisé avec l'identifiant.
description	Texte descriptif du type de contrôle.
icone	Icône type de contrôle au format png ou svg. Si aucun icone n'est défini, l'icône `contrôle_defaut-24.svg` est utilisé.
necessite	Liste des préfixes des plugins nécessités par le type de contrôle pour assurer son exécution ou tableau vide sinon. Le tableau est stocké sérialisé.
include	Chemin relatif du fichier contenant la fonction de contrôle ou chaine vide sinon.
fonction	Nom de la fonction d'exécution. Par défaut, en l'absence de précision, coïncide avec l'identifiant du type de contrôle.
parametres	Paramètres à fournir en entrée de la fonction ou du squelette sous forme de saisies. Possède à minima deux index, l'un pour les paramètres de la fonction, 'fonction', l'autre pour les paramètres du squelette, 'squelette'. Le tableau est stocké sérialisé.
anomalies	Liste des identifiants d'anomalies rangée suivant les actions acquitter ou corriger et chemin de l'include des fonctions de correction. Le tableau donc les index 'acquitter', 'corriger' et 'include'. Le tableau est stocké sérialisé.
squelette	Chemin relatif du squelette HTML permettant un affichage complémentaire spécifique au contrôle ou chaine vide sinon.
contexte	Tableau de contexte spécifique au squelette.

Données complémentaires calculées lors du chargement	
est_etat	Indique que le type de contrôle est un état.
actif	Indicateur d'activité du contrôle. Si différent de 'oui', aucun contrôle de ce type ne peut être réalisé. Dans ce cas, la valeur permet de déterminer le type de l'erreur constatée au chargement et commence par 'no'.
signature	MD5 de la description du type de contrôle (YAML ou pipeline).
maj	Timestamp automatique de l'objet.

5.1.2 Les contrôles

Un contrôle est un objet SPIP matérialisé par un enregistrement dans la table `spip_controles`. Les champs de cette table sont listés ci-dessous avec leur valeur par défaut.

TABLE SPIP_CONTROLES	
id_controle	Identifiant incrémental du contrôle.
type_controle	Identifiant du type de contrôle dont le contrôle est une instance.
parametres	Tableau, éventuellement vide, définissant les valeurs des paramètres du type de contrôle saisis dans le formulaire d'exécution.
date	Date de l'exécution
etat_execution	Statut retourné par la fonction d'exécution. Vaut 'exec_ok' ou 'exec_nok_defaut'.
id_auteur	Identifiant de l'auteur ayant exécuté le contrôle ou 0 si le lancement a été automatique (CRON, non disponible actuellement).
nb_anomalies	Nombre d'anomalies encore dans l'état ouvert.
maj	Timestamp automatique de l'objet.

5.1.3 Les observations

Une observation est un objet SPIP matérialisé par un enregistrement dans la table `spip_observations`. Les champs de cette table sont listés ci-dessous avec leur valeur par défaut.

TABLE SPIP_OBSERVATIONS	
id_observation	Identifiant incrémental de l'observation.
id_controle	Identifiant du contrôle ayant remonté l'observation.
type_controle	Identifiant du type de contrôle. Cette information est reprise du contrôle afin d'optimiser les boucles.
objet	Type d'élément sur lequel porte l'observation. Peut désigner un type d'objet SPIP ou un nom quelconque sinon.
id_objet	Identifiant de l'objet SPIP sur lequel porte l'observation ou 0 sinon.
url_objet	URL de l'élément sur lequel porte l'observation si celui-ci n'est pas un objet SPIP.
est_anomalie	Indique si l'observation est une anomalie, 'oui', ou pas, 'non'.
gravite	Indique si l'observation est une information, 'i', ou une erreur 'e'. Une anomalie est toujours considérée comme une erreur.
code	Identifiant textuel relatif de l'observation.
statut	Statut de l'observation. Une anomalie est créée avec le statut 'ouverte', peut passer à 'fermee' après correction ou acquittement puis être mise à la poubelle. Une observation non constitutive d'une anomalie est créée avec le statut 'fermee' directement car c'est juste un log.
parametres	Tableau, éventuellement vide, permettant d'expliquer précisément l'anomalie en fournissant des paramètres utilisables dans l'item de langue explicatif.
date	Date du passage dans le statut courant de l'observation.
maj	Timestamp automatique de l'objet.

5.1.4 Les dashboards

La description d'un dashboard est structurée dans un tableau associatif dont tous les champs possibles sont initialisés.

DESCRIPTION D'UN DASHBOARD	
Structure générale	
identifiant	Identifiant du dashboard au format chaine [a-z0-9_] ; correspond au nom du fichier de configuration YAML ou de l'index du tableau issu du pipeline `ezdashboard_declarer`.
nom	Libellé du dashboard. Si non défini est initialisé avec l'identifiant.
description	Texte descriptif du dashboard.
icone	Icône du dashboard png ou svg. Si aucun icone n'est défini, l'icône `dashboard_defaut-24.svg` est utilisé.
groupes	Tableau de configuration des groupes. Le tableau est indexé par l'identifiant de chaque groupe (cf. note [(1)]).
Structure du bloc groupes[(1)]	
identifiant	Identifiant du groupe au format chaine [a-z0-9_].
nom	Libellé du groupe qui est affiché comme titre de l'onglet correspondant.
controles	Liste des identifiants des types de contrôle associés au groupe.

5.2 Les espaces de stockage

Check Factory utilise la base de données SPIP comme espace de stockage privilégié. Ainsi, les types de contrôles, les contrôles et les observations sont stockées dans une table SPIP nommée respectivement `spip_types_controles`, `spip_controles` et `spip_observations`.

Les dashboards ne sont pas stockés en base de données mais dans un cache sécurisé géré par le plugin Cache Factory et nommé `tmp/dashboards.php`. Les données du cache sont sérialisées et ne sont jamais périmées : c'est à l'utilisateur de recharger manuellement les dashboards.

6. MISE EN PLACE D'UN DASHBOARD UTILISATEUR

Ce chapitre explique par l'exemple la mise en œuvre des différents cas d'utilisation des types de contrôle afin d'en appréhender la facilité. Tout est fait pour minimiser le code à écrire par chaque plugin utilisateur.

6.1 Mise en place d'un état

On reprend l'exemple du paragraphe 4.4.1 dont l'objectif est d'afficher la liste des plugins sans catégorie. Il est déjà nécessaire de créer un YAML de configuration du type de contrôle dont le nom défini son identifiant :

6.2 Mise en place d'un affichage paramétré

6.3 Mise en place d'une fonction non observable

6.4 Mise en place d'une fonction observable

7. REGLES DE CODAGE

7.1 Nommage des fonctions

Le nommage des fonctions appartenant à l'API de Check 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 type de contrôle se nomme `type_controle_lire()`.

7.2 Arguments et variables standardisés

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

Par exemple, l'argument `$id_type_controle` désigne toujours l'identifiant d'un type de contrôle. De façon identique, les identifiants de contrôle, d'observation ou de dashboard, qu'ils soient numériques ou chaines de caractères seront toujours précédés du préfixe `$id_` suivi du nom de l'objet en question.

Les variables ou arguments désignant la description de l'objet lui-même sont nommées avec le nom de l'objet comme `$type_controle`. Un tableau de descriptions est nommés de la même façon avec un caractère `s` final comme `$dashboards`.

7.3 Typage des arguments

Depuis la version 1.1.5, toutes les fonctions du plugin ont un prototype dont les arguments et le retour sont typés.

Seules 2 fonctions dont le retour peut accepter plusieurs types font exception car le type `mixed` n'est disponible qu'en PHP 8.0 alors que le plugin fonctionne encore à partir de PHP 7.4.