Gestion des traps SNMP

Sommaire

Concept général

Shinken Entreprise permet d'effectuer des remplacements dynamiques de contenu (*autrefois appelé "MACRO"*).

- Cela permet de paramétrer avec une grande flexibilité les commandes lancées par Shinken, l'affichage des seuils, les liens externes des éléments…
- La notation entre dollars **"$"** est utilisée pour permettre ce remplacement.
- Les **variables** vous donnent la capacité d'utiliser dans la **configuration de la supervision**,
  - des informations dites globales,
  - ainsi que les propriétés et les données présentes sur les éléments (Hôtes, Cluster, Check, Utilisateurs).
    - par exemple, les propriétés d'un hôte dans la vérification d'un check.

Exemple :

On dispose d'une commande qui se charge de contacter un hôte pour déterminer s'il est joignable ou non.

- On veut donc que la commande récupère automatiquement l'adresse de l'hôte sans avoir à spécifier manuellement l'adresse pour chaque hôte.
- Pour résoudre ce problème, on effectue un remplacement de contenu.
- Dans Shinken Entreprise, on peut utiliser la Variable $HOSTADDRESS$ qui va contenir l'adresse de l'hôte courant.

Ainsi, en utilisant cette **Variable** dans notre **commande**, lorsque celle-ci sera utilisée lors de la vérification d'un hôte, le mécanisme de remplacement dynamique va automatiquement remplacer la **Variable** par l'adresse de l'hôte.

Commande avant remplacement

```
check_ping -H "$HOSTADDRESS$" (...autres paramètres)
```

donne au final

Commande après remplacement

```
check_ping -H "192.168.1.12" (...autres paramètres)
```

Utilisation du remplacement dynamique de contenu

Endroits où le remplacement dynamique de contenu est effectué

Le remplacement dynamique de contenu n'est pas effectué partout.

Voici la liste par type d'élément où est effectué le remplacement dynamique de contenu :

Les Hôtes et leurs modèles

- Vivant (*Commande de vérification*),
- Affichage des seuils,
- URL externe,
- Liste des URL externes,

- Données locales & héritées d'un modèle,
- Commande lancée par le gestionnaire d'événements.

Les Clusters et leurs modèles

- Définition,
- Affichage des seuils,
- URL externe,
- Liste des URL externes,
- Données locales & héritées d'un modèle,
- Commande lancée par le gestionnaire d'événements.

Les Checks et leur modèles

- Commande de vérification,
- Affichage des seuils,
- URL externe,
- Liste des URL externes,
- Données locales & héritées d'un modèle,
- Commande lancée par le gestionnaire d'événements.

Les Utilisateurs avec leurs modèles

- Données locales & héritées d'un modèle

Les Méthodes de notification

- Commande de notifications pour l'hôte/cluster,
- Commande de notifications pour les checks.

Les Commandes

- Ligne de Commande

Les Modulations de données

- Données locales & héritées d'un modèle

Remplacement récursif

Le remplacement dynamique de contenu est **récursif**.

- Ce qui veut dire que les Variables dans les Variables seront remplacées.
- Il est possible d'effectuer une boucle de remplacement sans le vouloir (*exemple : **VARIABLE_1** nécessite **VARIABLE_2** qui nécessite **VARIABLE_3** qui nécessite **VARIABLE_1***).
  Dans ce cas, une erreur est remontée, dans l'interface de Configuration et de Visualisation.

Exemple d'erreur de remplacement récursif dans l'onglet Checks de la page d'édition d'un hôte.

? Unknown Attachment

Limites lors du remplacement des Variables dans une ligne de commande

Il peut arriver que le résultat de certaines Variables nécessite l'évaluation d'autres Variables pour être obtenu.

Pour éviter tout emballement récursif (*exemple : **VARIABLE_1** nécessite **VARIABLE_2** qui nécessite **VARIABLE_3** qui nécessite **VARIABLE_1***), les limites suivantes sont appliquées lors de la résolution des Variables :

- Il ne peut pas y avoir plus de 32 niveaux d'imbrication de Variables, au-delà de ce niveau, les Variables ne sont plus résolues.
- Il ne peut pas y avoir plus de 255 Variables à résoudre sur la ligne de commande, au-delà de ce nombre, les Variables ne sont plus résolues.
- La ligne de commande générée après résolution des Variables ne peut pas excéder 65000 caractères.

Si un dépassement se produit, la résolution des Variables est interrompue, et la ligne de commande est tronquée pour être remplacée par une commande remontant cette information.

Comment est effectué le remplacement des Variables

Le remplacement est fait dans deux démons :

- Le **Synchronizer** pour **l'Interface de Configuration.**
- Le **Scheduler** pour **l'Interface de Visualisation**, les **notifications** et la **préparation des commandes de supervision** (*pour leur exécution par le Poller*).

Le **Synchronizer** ne gérant que la configuration des éléments,

- il n'a pas accès aux états des éléments supervisés, comme le statut d'un hôte. **Il ne peut donc pas remplacer toutes les Variables.**
- La liste des Variables non résolues par le **Synchronizer** est disponible dans le chapitre ci-dessous (*voir Les Variables issues des Propriétés ($HOST...$, $SERVICE...$, $CONTACT...$)*)

Il y a une autre différence sur les remplacements effectués par les deux démons :

- le **Scheduler**, gère le nombre de Variables présentes sur la ligne de commande, à chaque étape de substitution.
- le **Synchronizer**, lors de l'essai de check,
  - **ne gère pas** le nombre de Variables présentes sur la ligne de commande lors des substitutions,
  - et **il n'applique pas** la règle limitant leur nombre à 255 à chaque étape de substitution.

Les différents types de Variables

Il existe trois types de Variables :

- Les Variables globales
- Les Variables d'élément
- Les Variables génératives

Les Variables globales

Il est possible dans Shinken Entreprise de définir des Variables globales accessibles partout dans Shinken et qui ne dépendent pas d'un élément particulier.

- Ces **Variables globales** se définissent dans des fichiers de configuration.
- Une globale nommée **MAGLOBALE** sera accessible avec la notation **$MAGLOBALE$**.

Remplacement des données globales

? Unknown Attachment

Définir des Variables globales

Les **Variables globales peuvent être définies UNIQUEMENT par fichiers de configuration**.

- Par défaut, un certain de nombre de Variables globales sont prédéfinies dans le dossier **/etc/shinken/resource.d**,
- Ce dossier contient tous les fichiers qui déclarent les Variables globales.
- Au démarrage du **Synchroniser**, ces fichiers sont chargés et les Variables globales qui y sont définies sont disponibles dans l'UI de Configuration.
- Au démarrage de **l'Arbiter,** ces fichiers sont chargés et les Variables globales qui y sont définies sont disponibles pour tous les autres démons.

La syntaxe pour la déclaration des Variables globales est la suivante :

Syntaxe de déclaration des globales

```
# Commentaire: les lignes commençant par # seront ignorées
# Les noms de globales doivent être entourés de $

$NOMDELAGLOBALE$=valeur
```

Les noms de **Variables globales** ne peuvent contenir **que** des caractères alphanumériques (*A-Z 0-9*), des tirets (-) et des soulignés (_).

- Le nom d'une Variable globale sera toujours en majuscules.
- Pour permettre à l'utilisateur de faire ses propres packs et faciliter l'import d'une configuration écrite au format cfg (*voir la page Collecteur de type (cfg-file-import) - Import depuis des fichiers au format .cfg),*
  - il est **possible** de déclarer des **Variables globales dans une source**.
  - Pour cela, il faut placer les fichiers .cfg dans un dossier **global-data** de la source.
    - Les fichiers de déclaration de Variables globales seront copiés par le Synchronizer dans **/etc/shinken/resource.d/** pour rendre leur contenu disponible comme pour les autres Variables globales.
      - Voir plus précisément le chapitre sur les données globales dans la page Collecteur de type cfg-file-import (format Shinken ou nagios)

Variables globales prédéfinies

Les Variables globales ci-dessous sont définies par Shinken (EXPLICATION)

Syntaxe	Description
$LONGDATETIME$	Heure/date courante au format long (*par exemple : Fri Oct 13 00:30:28 CDT 2000*)
$SHORTDATETIME$	Heure/date courante au format court (*par exemple : 10-13-2000 00:30:28*)
$DATE$	Date courante (*par exemple : 10-13-2000*)
$TIME$	Heure courante (*par exemple : 00:30:28*)
$TIMET$	Heure courante au format timestamp (*par exemple : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Utiliser des Variables globales

Les Variables globales sont accessibles **en entourant le nom** de la **Variable globale** par des dollars **"$"**.

- La Variable globale "MAGLOBALE" est donc accessible avec la notation $MAGLOBALE$.

Parce que les Variables globales sont définies dans les fichiers de configuration, l'ajout ou la modification d'une Variable globale dans ces fichiers nécessite un redémarrage du Synchronizer et de l'Arbiter pour que les changements soient pris en compte.

- C'est un point qui sera amélioré dans une prochaine version de Shinken.

Les Variables d'élément (Hôte, Cluster, Check, Utilisateur)

Les Variables d'élément sont évalués à partir des propriétés (*de définition ou d'exécution*) ou des données d'un élément

- Pour information, le propriétés concernent

Les Variables issues des Propriétés ($HOST...$, $SERVICE...$, $CONTACT...$)

Parmi tous les éléments de Shinken Entreprise, il est possible d'accéder à certaines propriétés des **Hôtes,** des **Clusters**, des **Checks** et des **Utilisateurs**.

- Pour accéder à une propriété d'un élément, il faut bien sûr le $, puis le type de l'élément et le nom de la propriété
  - Pour les Types :
    - **HOST =>** pour les **Hôtes** et **Clusters** (*c'est le même mot clé de type*)
    - **SERVICE** => pour les **Checks**
    - **CONTACT =>** pour les **Utilisateurs**
  - On accède a la propriété en rajoutant son nom aprés le type

Propriétés de l'élément

Variable	Fonction
$HOSTPROPRIETE$	Accède une propriété de l'hôte
$SERVICEPROPRIETE$	Accède à une propriété du check
$CONTACTPROPRIETE$	Accède à une propriété de l'utilisateur

  - Les propriétés disponibles par éléments différents en fonction de l'élément.
    - Les noms des propriétés accessibles par le mécanisme de variable sont listés dans les chapitres ci-dessous décrivant les éléments (*Hôtes, Clusters, Checks, Utilisateurs*).

- Par exemple, pour qu'une commande de vérification d'un check accède à **l'adresse d'un hôte**, il faut utiliser **$HOSTADDRESS$**.

Remplacement de propriétés

? Unknown Attachment

Propriétés des hôtes

Syntaxe	Description	Remplacé dans le Synchronizer	Remplacé dans le Scheduler
$HOSTNAME$	Nom de l'hôte (*propriété host_name*)	✓	✓
$HOSTDISPLAYNAME$	Nom d'affichage de l'hôte (*propriété display_name*)	✓	✓
$HOSTADDRESS$	Adresse de l'hôte (*propriété address*)	✓	✓
$HOSTSTATE$	Statut courant de l'hôte (*UP, DOWN, ou UNREACHABLE*)	✗	✓
$HOSTSTATETYPE$	Confirmation du statut d'un hôte (*SOFT ou HARD*)	✗	✓
$HOSTSTATEID$	Numéro correspondant au statut courant de l'hôte (*0=UP, 1=DOWN, ou 2=UNREACHABLE*)	✗	✓
$LASTHOSTSTATE$	Statut précédent de l'hôte (*UP, DOWN, ou UNREACHABLE*)	✗	✓
$LASTHOSTSTATEID$	Numéro correspondant au statut précédent de l'hôte (*0=UP, 1=DOWN, ou 2=UNREACHABLE*)	✗	✓
$HOSTGROUPNAME$	Nom du groupe d'hôte auquel appartient l'hôte. Si il appartient à plusieurs groupes d'hôtes, un seul sera retourné	✗	✓
$HOSTGROUPNAMES$	Liste des groupes d'hôtes auxquels appartient l'hôte, séparés par des virgules	✓	✓
$LASTHOSTCHECK$	Date au format timestamp de la dernière vérification de l'hôte	✗	✓
$LASTHOSTSTATECHANGE$	Date au format timestamp du dernier changement de statut de l'hôte	✗	✓
$LASTHOSTUP$	Date au format timestamp du dernier statut UP de l'hôte	✗	✓
$LASTHOSTDOWN$	Date au format timestamp du dernier statut DOWN de l'hôte	✗	✓
$LASTHOSTUNREACHABLE$	Date au format timestamp du dernier statut UNREACHABLE de l'hôte	✗	✓
$HOSTOUTPUT$	Résultat de la dernière vérification de l'hôte	✗	✓
$LONGHOSTOUTPUT$	Résultat long de la dernière vérification de l'hôte	✗	✓
$HOSTPERFDATA$	Données de performances renvoyées par la dernière vérification de l'hôte	✗	✓
$HOSTCHECKCOMMAND$	Nom de la commande utilisée pour la vérification de l'hôte (*avec les paramètres*)	✓	✓
$HOSTNOTESURL$	URL externe de l'hôte (*propriété notes_url*)	✓	✓
$HOSTBUSINESSIMPACT$	Nombre entre 0 et 5 indiquant l'impact métier de l'hôte	✓	✓
$HOSTFIRSTNOTIFICATIONDELAY$	Nombre de minutes à attendre avant d'envoyer la **première** notification pour un hôte	✓	✓
$HOSTNOTIFICATIONNUMBER$	Nombre de notifications consécutives envoyées pour un statut différent de UP	✓	✓
$HOSTTHRESHOLDSDISPLAY$	Affichage des seuils, tel qu'il est paramétré sur l'hôte	✓	✓

⚠ On accède en général aux propriétés de l'hôte avec la notation entre dollars commençant par HOST (*exemple : $HOSTADDRESS$*). Dans le tableau, certaines entrées ne commençant pas par HOST sont présentes, mais elles font quand même référence à une propriété de l'hôte.

Propriétés des checks

Syntaxe	Description	Remplacé dans le Synchronizer	Remplacé dans le Scheduler
$SERVICEDESC$	Nom/description du check	✓	✓
$SERVICEDISPLAYNAME$	Nom d'affichage du check (*propriété display_name*)	✓	✓
$SERVICESTATE$	Statut courant du check (*OK, WARNING, UNKNOWN, CRITICAL*)	✗	✓
$SERVICESTATETYPE$	Confirmation du statut d'un check (*SOFT ou HARD*)	✗	✓
$SERVICESTATEID$	Numéro correspondant au statut courant du check (*0=UP, 1=DOWN, ou 2=UNREACHABLE*)	✗	✓
$LASTSERVICESTATE$	Statut précédent du check (*OK, WARNING, UNKNOWN, CRITICAL*)	✗	✓
$LASTSERVICESTATEID$	Numéro correspondant au statut précédent du check (*0=UP, 1=DOWN, ou 2=UNREACHABLE*)	✗	✓
$SERVICEISVOLATILE$	Booléen indiquant si le check est volatile (*0=Non volatile, 1=Volatile*)	✓	✓
$LASTSERVICECHECK$	Date au format timestamp de la dernière exécution du check	✗	✓
$LASTSERVICESTATECHANGE$	Date au format timestamp du dernier changement de statut du check	✗	✓
$LASTSERVICEOK$	Date au format timestamp du dernier statut OK du check	✗	✓
$LASTSERVICEWARNING$	Date au format timestamp du dernier statut WARNING du check	✗	✓
$LASTSERVICEUNKNOWN$	Date au format timestamp du dernier statut UNKNOWN du check	✗	✓
$LASTSERVICECRITICAL$	Date au format timestamp du dernier statut CRITICAL du check	✗	✓
$SERVICEOUTPUT$	Résultat de la dernière vérification du check	✗	✓
$LONGSERVICEOUTPUT$	Résultat long de la dernière vérification du check	✗	✓
$SERVICEPERFDATA$	Données de performances renvoyées par la dernière exécution du check	✗	✓
$SERVICECHECKCOMMAND$	Nom de la commande utilisée pour l'exécution du check (*avec les paramètres*)	✓	✓
$SERVICENOTESURL$	URL externe du check (*propriété notes_url*)	✓	✓
$SERVICEBUSINESSIMPACT$	Nombre entre 0 et 5 indiquant l'impact métier du check	✓	✓
$SERVICEFIRSTNOTIFICATIONDELAY$	Nombre de minutes à attendre avant d'envoyer la **première** notification pour un service	✓	✓
$SERVICENOTIFICATIONNUMBER$	Nombre de notifications consécutives envoyées pour un statut différent de OK	✗	✓
$SERVICETHRESHOLDSDIPLAY$	Affichage des seuils, tel qu'il est paramétré sur le check	✓	✓

⚠ On accède en général aux propriétés du check avec la notation entre dollars commençant par SERVICE (*exemple : $SERVICEDESC$*). Dans le tableau, certaines entrées ne commençant pas par SERVICE sont présentes, mais elles font quand même référence à une propriété du check.

Propriétés des utilisateurs

Syntaxe	Description	Remplacé dans le Synchronizer	Remplacé dans le Scheduler
$CONTACTNAME$	Nom de l'utilisateur (*propriété contact_name*)	✓	✓
$CONTACTEMAIL$	Adresse mail de l'utilisateur (*propriété email*)	✓	✓
$CONTACTPAGER$	Numéro de téléphone de l'utilisateur (*propriété pager*)	✓	✓

Les Variables pour les Données ($_HOST...$, $_SERVICE...$, $_CONTACT...$)

Shinken Entreprise permet d'ajouter des données personnalisées sur certains éléments, comme les hôtes, les checks, les utilisateurs, et bien sûr les modèles d'hôtes, de checks et d'utilisateurs. Ces données permettent de compléter la définition d'un élément lorsque les propriétés par défaut de l'objet ne suffisent pas à le décrire complètement.

Par exemple, si un hôte possède deux interfaces réseau, les propriétés par défaut de l'objet ne permettent pas de spécifier deux adresses. Par contre, il est possible d'ajouter une donnée personnalisée qu'on appelle par exemple "ADDRESS_2" qui pourra être utilisée lorsqu'on aura besoin d'avoir la deuxième adresse de l'hôte dans un check.

Ces données sont accessibles dans un objet en utilisant la notation suivante :

- $_**HOST**NOMDELADONNEPERSONNALISEE$.
- $_**SERVICE**NOMDELADONNEPERSONNALISEE$
- $_**CONTACT**NOMDELADONNEPERSONNALISEE$

Remarque

On remarque la présence d'un underscore (_) avant HOST, SERVICE ou CONTACT, ce qui permet de différencier l'accès à une propriété de l'élément et l'accès à une donnée personnalisée.

Données personnalisées

Variable	Fonction
$_HOSTDONNEE$	Accède à la donnée personnalisée "DONNEE" de l'hôte
$_SERVICEDONNEE$	Accède à la donnée personnalisée "DONNEE" du check
$_CONTACTDONNEE$	Accède à la donnée personnalisée "DONNEE" de l'utilisateur

Remplacement des données personnalisées

? Unknown Attachment

Définir des données personnalisées

Les données locales peuvent être définies sur les hôtes, checks, utilisateurs et leurs modèles respectifs de 2 manières:

- Par fichier de configuration
- Par l'interface de configuration

Dans un fichier de configuration, les données sont définies en préfixant un _ à leur nom. Le nom d'une donnée peut contenir seulement des caractères alphanumériques (A-Z0-9), des tirets (-) ou underscore (_). Aussi, le nom d'une donnée sera toujours en majuscules.

Exemple d'une objet définissant la donnée DONNEE_PERSONNALISEE

```
define host {
    host_name    mon_hote
    address                192.168.0.12


    _DONNEE_PERSONNALISEE      valeur_de_la_donnée
}
```

Dans l'interface de configuration, l'ajout et la modification de données personnalisées s'effectuent grâce à l'onglet "Données".

Ajout d'une donnée dans l'interface de Configuration

? Unknown Attachment

Cette capture d'écran montre l'édition de données personnalisées dans le cas d'un hôte. Les mêmes manipulations sur les données personnalisées sont possibles pour les modèles d'hôtes, clusters, modèles de clusters, checks, modèles de check, utilisateurs et modèles d'utilisateurs.

Dans un fichier de configuration, une donnée personnalisée est définie en précédant son nom d'un underscore (_).

Dans l'interface de configuration, cet underscore ne doit pas être spécifié, car il s'agit seulement d'un moyen dans les fichiers de différencier une donnée personnalisée d'une propriété. La déclaration d'une donnée personnalisée depuis l'interface se fait seulement en précisant le nom de la donnée et sa valeur.

Les Variables génératives

Nous avons vu que lorsque l'on remplace dynamiquement du contenu, il est possible de faire référence aux propriétés, aux données locales ainsi qu'aux Variables globales. Il existe également des Variables spéciales qui ne récupèrent ni des Variables globales ni des données locales ou des propriétés d'un élément. Elles permettent de transférer des données, notamment des arguments de commande.

Ces Variables spéciales sont accessibles avec les notations $ARGn$ et $VALUEn$.

Les Variables générées par l'utilisation d'une commande ($ARGn$, $VALUEn$)

Shinken Entreprise permet aux hôtes et aux checks de préciser les commandes qui seront utilisées pour la vérification de l'élément. Dans l'optique de rendre ces commandes le plus générique possible, et de permettre de factoriser la configuration, des arguments peuvent être passés à ces commandes.

Ces arguments sont séparés par des points d'exclamation '**!**'.

Dans l'exemple, un check utilise la commande "ma_commande" et lui passe 3 arguments.

? Unknown Attachment

Dans la définition de la commande, on veut pouvoir récupérer la valeur de ces arguments pour pouvoir les utiliser sur la ligne de commande.

Pour cela, on utilise la notation $ARGn$. La Variable $ARGn$ permet simplement d'accéder à la valeur du n-ième argument.

Dans l'exemple, on utilise $ARG1$, $ARG2$ et $ARG3$ pour récupérer les valeurs du premier, deuxième et troisième argument.

? Unknown Attachment

Il est possible d'utiliser des Variables comme argument.

- Exemple : 200!$_HOSTWARNING_LEVEL$
- Elle seront remplacées lors de l'évaluation de la commande.

Pour une utilisation avancée, et avec un certain niveau de maîtrise nécessaire, il est même possible d'utiliser une Variable pour définir plusieurs arguments.

- Il faut utiliser pour cela le séparateur : |-|
- Exemple :
  - Dans les arguments : VERBOSE!$_HOSTLEVELS$
  - Dans l'hôte la donnée LEVELS vaut 200|-|400
  - La commande script.py $ARG1$ $ARG2$ $ARG3$ deviendra script.py VERBOSE 200 400

Récupération des arguments dans une commande

? Unknown Attachment

Comme dans Nagios, il est possible d'utiliser jusqu'à 32 arguments.

Ainsi, les Variables $ARG1$ à $ARG32$ sont utilisables.

Les Variables générées par l'utilisation de la Duplication de check (Duplicate Foreach) - (KEY, $VALUE1$)

La fonctionnalité avancée Dupliquer des checks en fonction d'une liste de valeurs présentes dans la Donnée d'un hôte (duplicate_foreach) permet d'appliquer plusieurs fois le même check sur un hôte avec des paramètres différents, selon la valeur d'une donnée personnalisée sur l'hôte.

Sur chaque check utilisant la fonctionnalité Duplicate Foreach, une clé est affecté, et optionnellement des paramètres.

Exemple de donnée Duplicate Foreach

```
check1$(valeur1)$$(valeur2)$$(valeur3)$
```

La valeur de la clé est accessible avec la Variable KEY, et les arguments sont accessible grâce aux Variables $VALUEn$.

Le tableau suivant récapitule les Variables permettant d'accéder aux valeurs de la donnée Duplicate Foreach:

Variable	Valeur
KEY	check1
$VALUE1$	valeur1
$VALUE2$	valeur2
$VALUE3$	valeur3

Il possible d'utiliser 16 valeurs différentes. Ainsi, les Variables $VALUE1$ jusqu'à $VALUE16$ sont disponibles.

Exemple d'utilisation des données Duplicate Foreach

? Unknown Attachment