Le mélange des sources & les clés de synchronisation (sync-key)

Sommaire

Le mélange des sources

Après la collecte des données, c'est l'étape de mélange des sources (rappel la description des étapes de l'import des sources se trouve dans la page Modules de Sources (imports) et de Taggers (qualification)).

- Le mélange des sources permet de créer un élément dont la définition est répartie entre plusieurs sources.
- Les **clés de synchronisation** permettent de choisir quelle définition va être fusionnée avec quelle autre définition.

> **La règle de mélange des sources**
>
> Si une définition possède au moins une clé de synchronisation en commun avec une autre définition alors ces 2 définitions fusionnent.

- Les propriétés utilisées pour récupérer les clés de synchronisation dépendent du type de l'élément et de la source.
- Les pages des sources listent les clés utilisées par défaut.

> **Important**
>
> Les éléments d'une même source possédant au moins une clé de synchronisation en commun doivent avoir toutes leurs clés de synchronisation identiques (Exemple pour un hôte de la source cfg-file : même nom et même adresses) sinon ces éléments seront en erreur.

Exemple d'un mélange de source

Exemple avec 2 sources :

- *Ma source d'import par fichier* : source de type cfg-file-import (*voir page* Module de source de type cfg-file-import).
- *Discovery* : source de type discovery-import (*voir page* Module de source discovery (de type discovery-import)).

Les 2 sources ont leurs clés de synchronisation configurées par défaut, c'est-à-dire :

Pour les hôtes de la source "*Ma source d'import par fichier*" :

- host_name (*le nom de l'hôte*)
- _SE_UUID
- address

Pour les hôtes de la source *Discovery* :

- host_name (*le nom de l'hôte*)
- address

Les 2 sources donnent les définitions suivantes :

Définition de : Ma source d'import par fichier

```
define host {
    host_name          Hote 1
    address            localhost
    propriété_commune  valeur1
    propriété1         valeur1
}
```

Définition de : Discovery

```
define host {
    host_name          Hote 2
    address            localhost
    propriété_commune  valeur2
    propriété2         valeur2
}
```

élément résultat du mélange des 2 définitions

```
define host {
    host_name          Hote 1
    address            localhost
    propriété_commune  valeur1
    propriété1         valeur1
    propriété2         valeur2
}
```

Lorsque la même propriété est définie dans plusieurs définitions, la valeur de l'hôte pris en compte sera celle dont la source a le plus petit numéro d'ordre.

Dans l'exemple, on voit que la propriété commune (*propriété_commune*) a pour valeur la première valeur définie ("*valeur1*" et que le nom de l'hôte est "*Hote* 1"), car la source "*Ma source d'import par fichier*" à un ordre plus petit que la source "*Discovery*".

Les clés de synchronisation

Définition

Une clé de synchronisation est la valeur d'une propriété d'une définition donnée par la source. Le choix des propriétés dont les valeurs vont servir de clé de synchronisation dépend de la source et du type d'élément.
Certaines sources permettent de choisir quelles propriétés seront utilisées comme clé de synchronisation. Le nom de l'élément et son _SE_UUID sont toujours utilisés comme clé de synchronisation si la source est capable de fournir ces propriétés.

Exemple de cas où la source ne donne pas de _SE_UUID :
La source Discovery n'est pas capable de fournir un _SE_UUID donc cette source ne donne pas cette clé de synchronisation. Il est conseillé de ne pas importer les _SE_UUID d'un autre Synchronizer avec la source synchronizer-collector-linker. Cela pourrait créer des conflits avec les éléments déjà importés.

Cas du _SE_UUID

Parmi les clés de synchronisation d'un objet, le _SE_UUID est traité d'une manière différente.

Lorsqu'un élément possède un _SE_UUID dans sa définition, il est considéré comme unique dans le mécanisme d'import des sources, et il ne sera pas fusionné avec un élément ayant un _SE_UUID différent. Il peut par contre toujours être fusionné avec des éléments sans _SE_UUID ou avec d'autres éléments ayant le même _SE_UUID.

C'est utile dans le cas où on veut définir 2 hôtes avec la même adresse par exemple. Le _SE_UUID permet de dire qu'il s'agit de 2 hôtes distincts et ils ne seront pas fusionnés.

```
define host {

   # Shinken Enterprise. Lines added by import core. Do not remove it, it's used by Shinken Enterprise to
update your objects if you re-import them.
      _SE_UUID             core-hosts-60f9373a47d511e8a3c1080027a3ae8c
      _SE_UUID_HASH        7883f1c2c623f7001d70c02503ce198d
   # End of Shinken Enterprise part

    host_name               Host 1
    address                 host_addr
}

define host {

   # Shinken Enterprise. Lines added by import core. Do not remove it, it's used by Shinken Enterprise to
update your objects if you re-import them.
      _SE_UUID             core-hosts-6454ba8a47d511e897f8080027a3ae8c
      _SE_UUID_HASH        7b991a6b08e648842e1a7c7bc28ee140
   # End of Shinken Enterprise part


    host_name               Host 2
    address                 host_addr
    retry_interval          4
    check_running_timeout   4
}
```

Les 2 hôtes ne seront pas fusionnés à cause de la présence du _SE_UUID.

La data _SE_UUID_HASH permet de simplement vérifier que la data _SE_UUID n'a été générée correctement et n'a pas été corrompue par une typo d'un utilisateur en éditant son fichier.

A noter: elle n'a qu'une valeur d'information pour l'auteur du fichier, le Synchronizer ne refusera pas l'objet si la valeur est vide ou incorrecte.

Un couple SE_UUID / _SE_UUID_HASH à la forme suivante:

_SE_UUID core-**type_element**-**identifiant_uuid1_unique**

_SE_UUID_HASH **hash_md5_du_se_uuid**

- **type_element**: Le type de l'élément sur lequel on veut spécifier ce _SE_UUID. Les différentes valeurs possibles sont les différents types définissables par fichier de configuration:
  - clusters
  - clustertpls
  - hosts
  - hosttpls
  - hostgroups
  - serviceshosts
  - serviceshosttpls
  - servicesclusters
  - servicesclustertpls
  - servicetpls
  - contacts
  - contacttpls
  - contactgroups
  - escalations
  - notificationways
  - commands
  - businessimpactmodulations
  - macromodulations
  - resultmodulations
  - timeperiods
- **identifiant_uuid1_unique**: Chaine de caractères (*alphanumériques*) au format **UUID1** permettant d'identifier de manière unique l'élément.
- **hash_md5_du_se_uuid**: Hash MD5 de la valeur du _SE_UUID
  - par exemple:
    - pour un hôte, en ayant généré un uuid1="60f9373a47d511e8a3c1080027a3ae8c"
      - on calcule donc se_uuid_hash = md5("core-hosts-60f9373a47d511e8a3c1080027a3ae8c") => "7883f1c2c623f7001d70c02503ce198d"

```
_SE_UUID            core-hosts-60f9373a47d511e8a3c1080027a3ae8c
_SE_UUID_HASH       7883f1c2c623f7001d70c02503ce198d
```

Exemple de génération du couple _SE_UUID / _SE_UUID_HASH par code

L'**UUID1** et **MD5** sont normés, et vont donc donner des résultats similaires suivants les langages utilisés. Par exemple en python on peux les générer de cette manière:

Exemple de génération des UUID1 et MD5 en python pour un hôte

```
import uuid
import hashlib

# uuid1 generation as string
s_uuid = uuid.uuid1().hex

# generate se_uuid as string
se_uuid = u'core-hosts-%s' % s_uuid

# generate the se_uuid_hash as string, based on the se_uuid string
se_uuid_hash = hashlib.md5(se_uuid).hexdigest()
```

Voir les clés de synchronisation

Les clés de synchronisation sont visible depuis l'onglet Détail de la dernière exécution dans la page de configuration des sources