# 2 Macros utilisateur

## Aperçu

Les macros utilisateur sont prises en charge dans Zabbix pour une plus grande flexibilité, en plus des macros supportées prêtes à l'emploi.

Les macros utilisateur peuvent être définies aux niveaux : global, modèle et hôte. Ces macros ont une syntaxe spéciale :

```
{$MACRO}
```

Zabbix résout les macros selon la priorité suivante :

1. macros au niveau de l'hôte (vérifié en premier)
2. macros définies pour les modèles de premier niveau de l'hôte (c-à-d modèles liés directement à l'hôte), triées par ID de modèle
3. macros définies pour les modèles de second niveau de l'hôte, triées par ID de modèle
4. macros définies pour les modèles de troisième niveau de l'hôte, triées par ID de modèle, etc.
5. macros globales (vérifiées en dernier)

En d'autres termes, si une macro n'existe pas pour un hôte, Zabbix essaiera de la trouver dans les modèles d'hôte d'une profondeur croissante. S'il n'est toujours pas trouvé, une macro globale sera utilisée, si elle existe.

Si Zabbix ne parvient pas à trouver une macro, celle-ci ne sera pas résolue.

Les macros utilisateur peuvent être utilisées dans :

- les noms d'éléments
- les paramètres de clé d'élément
- l'intervalle de mise à jour des éléments et les intervalles flexibles
- les noms et descriptions des déclencheurs
- les paramètres et les constantes des expressions de déclencheurs (voir les exemples)
- beaucoup d'autres endroits - voir la liste complète

#### Cas d'utilisation courants des macros globales et d'hôtes

- Utiliser une macro globale à plusieurs endroits ; puis modifiez la valeur de la macro et appliquez les modifications de configuration à tous les emplacements en un seul clic
- Tirer parti des modèles dotés d'attributs spécifiques à l'hôte : mots de passe, numéros de port, noms de fichier, expressions régulières, etc.

## Configuration

Pour définir les macros utilisateur, accédez aux emplacements correspondants dans l'interface :

- pour les macros globales, visitez *Administration → Général → Macros*

- pour les macros d'hôte et de modèle, ouvrez les propriétés d'hôte ou de modèle et recherchez l'onglet *Macros*.

Si une macro utilisateur est utilisée dans des éléments ou des déclencheurs dans un modèle, il est suggéré d'ajouter cette macro au modèle même si elle est définie à un niveau global. De cette manière, l'exportation du modèle au format XML et son importation dans un autre système lui permettront toujours de fonctionner comme prévu.

Les caractères suivants sont autorisés dans les noms des macros : **A-Z** , **0-9** , **_** , **.**

## Exemples

#### Exemple 1

Utilisation de la macro au niveau de l'hôte dans la clé d'élément “Statut du démon SSH” :

```
net.tcp.service[ssh,,{$SSH_PORT}]
```

Cet élément peut être affecté à plusieurs hôtes, à condition que la valeur de **{$SSH_PORT}** soit définie sur ces hôtes.

#### Exemple 2

Utilisation de la macro au niveau de l’hôte dans le déclencheur “La charge CPU est trop importante” :

```
{ca_001:system.cpu.load[,avg1].last()}>{$MAX_CPULOAD}
```

Un tel déclencheur serait créé sur le modèle et non modifié dans des hôtes individuels.

Si vous souhaitez utiliser une quantité de valeurs comme paramètre de fonction (par exemple, **max(#3)**), incluez le signe dièse dans la définition de la macro, comme ceci : SOME_PERIOD ⇒ #3

#### Exemple 3

Utilisation de deux macros dans le déclencheur “La charge CPU est trop importante” :

```
{ca_001:system.cpu.load[,avg1].min({$CPULOAD_PERIOD})}>{$MAX_CPULOAD}
```

Notez qu'une macro peut être utilisée comme paramètre de fonction de déclencheur, dans cet exemple la fonction **min()**.

Dans les expressions de déclencheur, les macros utilisateur seront résolues si vous faites référence à un paramètre ou à une constante. Ils ne résoudront PAS si vous référencez l'hôte, la clé d'élément, la fonction, l'opérateur ou une autre expression de déclencheur.

#### Exemple 4

Synchroniser la condition d'indisponibilité de l'agent avec l'intervalle de mise à jour de l'élément :

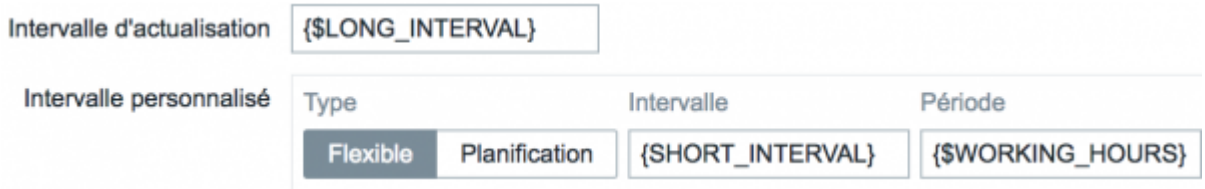
- définir la macro {$INTERVAL} et l'utiliser dans l'intervalle de mise à jour des éléments ;
- utiliser {$INTERVAL} comme paramètre du déclencheur d'indisponibilité de l'agent :

```
{ca_001:agent.ping.nodata({$INTERVAL})}=1
```

**Exemple 5**

Centraliser la configuration des heures de travail :

- créez une macro globale {$WORKING_HOURS} égale à `1-5,09:00-18:00` ;
- utilisez-la dans *Administration → Général → Heures ouvrées* ;
- utilisez-la dans *Utilisateurs → Média → Lorsque actif* ;
- utilisez-la pour configurer des interrogations plus fréquentes au cours des heures ouvrées :




- utilisez-la dans la condition d'action de *Période* ;
- ajustez les heures ouvrées *Administration → Général → Macros*, si nécessaire.

### Contexte de macro utilisateur

Un contexte facultatif peut être utilisé dans les macros utilisateur, permettant de remplacer la valeur par défaut par une valeur spécifique au contexte.

Les macros utilisateur avec contexte ont une syntaxe similaire :

```
{$MACRO:context}
```

Le contexte de macro est une valeur de texte simple. Le cas d'utilisation courant pour les contextes de macro serait d'utiliser une valeur de macro de découverte de bas niveau en tant que contexte de macro d'utilisateur. Par exemple, un prototype de déclencheur peut être défini pour que la découverte du système de fichiers monté utilise une limite d'espace disque faible différente en fonction des points de montage ou des types de système de fichiers.

Seules les macros de découverte de bas niveau sont prises en charge dans les contextes de macro. Toutes les autres macros sont ignorées et traitées comme du texte brut.

Techniquement, le contexte de macro est spécifié à l'aide de règles similaires aux paramètres de clé d'élément, à l'exception du contexte de macro qui n'est pas analysé comme plusieurs paramètres s'il existe un caractère `,` :

- Le contexte de macro doit être encadré avec `"` si le contexte contient un caractère `}` ou commence par un `"`. Les guillemets dans le contexte doivent être échappés avec le caractère `\`. Le caractère `\` lui-même n'est pas échappé, ce qui signifie qu'il est impossible d'avoir un contexte cité se terminant par le caractère `\` - la macro `{$MACRO:"a:\b\c\"}` n'est pas valide.

- Les espaces de début dans le contexte sont ignorés, les espaces de fin ne le sont pas. Par exemple,`{$MACRO:A}` est identique à `{$MACRO: A}`, mais pas à `{$MACRO:A }`.
- Tous les espaces avant les guillemets et après les guillemets sont ignorés, mais pas tous les espaces entre guillemets. Les macros `{$MACRO:"A"}`, `{$MACRO: "A"}`, `{$MACRO:"A" }` et `{$MACRO: "A" }` sont les mêmes, mais les macros `{$MACRO:"A"}` et `{$MACRO:" A "}` ne le sont pas.

Les macros suivantes sont toutes équivalentes, car elles ont le même contexte : `{$MACRO:A}`, `{$MACRO: A}` et `{$MACRO:"A"}`. Ceci est en contraste avec les clés d'élément, où `key[a]`, `key[ a]` et `key["a"]` sont les mêmes sémantiquement, mais différentes pour des raisons d'unicité.

Lorsque les macros de contexte sont traitées, Zabbix recherche la macro avec son contexte. Si une macro avec ce contexte n'est pas définie par les modèles hôte ou les modèles liés et si elle n'est pas définie en tant que macro globale avec contexte, la macro sans contexte est recherchée.

Voir l'exemple d'utilisation du contexte de macro dans un prototype de déclencheur d'espace disque et prenez en compte la clause de limitation.