Genesys Administrator Extension Help

Utilisation de la console de ligne de commande (CLC)

4/8/2025

Utilisation de la console de ligne de commande (CLC)

Sommaire

La console de ligne de commande (CLC) permet aux administrateurs d'utiliser la ligne de commande pour exécuter certaines fonctions GAX sur les définitions de solution (SPD) et les packages d'installation (IP). Par exemple, vous pouvez utiliser la CLC pour déployer les SPD silencieusement sur les hôtes distants.

Vous devez pouvoir accéder à l'interface de ligne de commande du système d'exploitation pour utiliser la CLC. Si vous n'utilisez pas l'ordinateur hôte GAX, l'outil CLC (**gaxclc.jar**) doit être disponible sur l'ordinateur local.

Pour accéder au fichier d'aide intégré de la CLC, exécutez l'une des commandes suivantes :

```
java –jar gaxclc.jar help

java –jar gaxclc.jar ?
```

> **Important**
>
> Lorsque vous exécutez les commandes avec la CLC, un fichier journal est généré à l'emplacement d'exécution de l'outil.

Structure

La CLC prend en charge les commandes qui utilisent la structure suivante :

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> <function> <operation> <args>
```

Dans l'exemple ci-dessus :

- `-u:user` correspond au nom de l'utilisateur utilisé pour se connecter à Configuration Server.
- `-p:password` correspond au mot de passe utilisé pour se connecter à Configuration Server. La CLC suppose qu'il n'y a pas de mot de passe si aucune valeur n'est précisée pour cet indicateur.
- `-s` demande à la CLC d'utiliser une connexion « https » sécurisée pour accéder au serveur GAX. Si cet indicateur n'est pas précisé, la CLC utilise « http ».
- `-h:<host>:<port>` spécifie l'hôte et le port du serveur GAX. Si cet indicateur n'est pas précisé, la CLC utilise la valeur suivante : `-h:localhost:8080`.
- `<function>` peut être `ip` ou `spd`.
- `<operation>` spécifie l'action à exécuter. Les valeurs valides de cet indicateur sont spécifiques à la fonction indiquée lors de l'étape précédente (`ip` ou `spd`).
- `<args>` spécifie les arguments d'action. Les valeurs valides de cet indicateur sont spécifiques aux paramètres `<function>` et `<operation>` définies lors des étapes précédentes.

Voici un exemple de commande CLC :

```
java -jar gaxclc.jar -u:default -p:password -h:localhost:8080 spd execute 10054 1 "C:/GAX/
input.txt"
```

Définitions de solution

La CLC prend en charge les actions suivantes concernant les définitions de solution :

- `add`
- `query`
- `querybyid`
- `execute`
- `delete`
- `encrypt` (voir l'onglet `execute`)

add

add

Présentation

Cette action ajoute une définition de solution à la base de données GAX. Si la définition de solution existe déjà, et se termine par le nom et la version dans le XML de la définition de solution, cette action remplace la définition de solution existante.

En cas de réussite, l'action renvoie l'ID de la définition de solution ajoutée.

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd add "file path"
```

- `"file path"` — Chemin d'accès au fichier XML.

Exemple

```
java -jar gaxclc.jar -u:default -p:password spd add "c:\GAX\newSpd.xml"
```

query

query

Présentation

Cette action interroge toutes les définitions de solution et affiche un tableau qui répertorie les détails suivants sur chaque définition de solution :

- Numéro d'ID
- Nom
- Version
- ID base de données du locataire

Voici un exemple :

```
10054 gvp 8.1.5 1
10060 genesysOne 8.1.5 1
10060 eServices 8.1.5 1
```

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd query
```

Exemple

```
java -jar gaxclc.jar -u:default -p:password -s -h:132.45.43.45:443 spd query
```

querybyid

querybyid

Présentation

Cette action interroge une définition de solution à l'aide de son ID. Si la définition de solution n'existe pas, l'action échoue.

En cas de réussite, l'action affiche un tableau qui répertorie les détails suivants sur la définition de solution :

- ID de profil
- Nom

Par exemple :

```
1 Install
```

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd query SPDID
```

- SPDID — ID de la définition de solution interrogée.

Exemple

```
java -jar gaxclc.jar -u:default -p:password -h:132.45.43.45:8080 spd query 4374
```

execute

execute

Présentation

Cette action exécute une définition de solution.

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd execute SPDID profileName|
-profileID:profileID|-profileName:profileName -encrypted "input file"
```

- SPDID — ID de la définition de solution à exécuter.
- profileName|-profileID:profileID|-profileName:profileName—Profil de la définition de solution à exécuter.

> **Important**
> Si aucun indicateur n'est spécifié, profileName est considéré comme étant le profil de définition de solution à exécuter.

- -encrypted — Si spécifié, indique si le fichier d'entrée est chiffré.

[+] Afficher l'utilisation

La CLC fournit un support de chiffrement pour les fichiers d'entrée qui comprennent des données sensibles comme les mots de passe.

Format :

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd encrypt "input file path"
```

```
"encrypted output file path"
```

Le fichier d'entrée chiffré est conservé à l'emplacement spécifié par `"encrypted output file path"`. Si le fichier existe déjà à cet emplacement, il est écrasé.

Exemple :

```
java -jar gaxclc.jar -u:default -p:password spd -encrypted "c:\GAX\input.txt" "c:\GAX\
encrypted.txt"

java -jar gaxclc.jar -u:default -p:password spd -encrypted "input.txt" "encrypted.txt"
```

- `"input file"` — Spécifie le fichier d'entrée qui contient les paramètres de définition de solution. Si `-encrypted` est défini, le fichier d'entrée est chiffré.

Le fichier d'entrée doit être au format JSONObject et comprendre les paramètres de définition de solution d'un profil spécifique. Le fichier doit être codé au format UTF-8.

[+] Afficher l'utilisation

The input file must be in JSONObject format and include SPD parameters for a specific profile. The file must be encoded in UTF-8 format.

string

The input structure for a *string* type is described below:

```
{
        "Dialog name" : {
                "Input name" : "string"
        }
}
```

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="NAME_PARAM1" title="PERSON NAME"  default="birit" type="string"
required="true">
           <description>Please enter the person name</description>
        </input>
 </dialog>
 <dialog step="Step2">
        <input name="NAME_PARAM2" title="PERSON NAME"  default="birit" type="string"
required="true">
           <description>Please enter the person name</description>
        </input>
 </dialog>
 <execution>
```

```
        <script>
                log('string test' );
        </script>
 </execution>
</profile>
```

Input File for Install Profile

```
{
        "Step1" : {
                "NAME_PARAM1" : "Kate"
        },
        "Step2" : {
                "NAME_PARAM2" : "John"
        }
}
```

Boolean

The input structure for a *boolean* type is described below:

```
{
        "Dialog name" : {
                "Input name" : true/false
        }
}
```

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="STATUS" title="status"  type="boolean" required="true">
           <description>status field</description>
        </input>
 </dialog>
 <execution>
        <script>
                log('boolean test');
        </script>
 </execution>
</profile>
```

Input File for Install Profile

```
{
        "Step1" : {
                "STATUS" : true
        }
}
```

Integer

The input structure for an *integer* type is described below:

```
{
        "Dialog name" : {
                "Input name" : <integer>
        }
}
```

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="NUMBER" title="number"  type="integer" required="true">
           <description>number field</description>
        </input>
 </dialog>
 <execution>
        <script>
                log('number test');
        </script>
 </execution>
</profile>
```

Input File for Install Profile

```
{
        "Step1" : {
                "NUMBER" : 132
        }
}
```

Password

The input structure for a *password* type is described below:

```
{
        "Dialog name" : {
                "Input name" : "password"
        }
}
```

> **Important**
>
> Input files that include sensitive data such as passwords should be encrypted using the SPD encrypt operation.

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="PASSWORD" title="password"  type="password" required="true">
           <description>password field</description>
        </input>
 </dialog>
 <execution>
        <script>
                log('password test');
        </script>
 </execution>
</profile>
```

Input File for Install Profile

```
{
        "Step1" : {
                "PASSWORD" : "xyz9846gdkjg"
        }
}
```

SelectOne

The input structure for a *selectOne* type with an **<objectselect>** tag is described bellow:

```
{
        "Dialog name" : {
                "Input name" : {
                        "objectselect" : {
                                "filter" : [{
                                                "value" : "filter value",
                                                "name"  : "filter name"
                                        }
                                ]
                        }
                }
        }
}
```

> **Important**
>
> CLC intersects (*AND*) filters defined in the SPD file and input file for a *selectOne* input. The filter criteria should be different in an SPD input file and filter names should differ in the same filter definition.

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="APP_OBJ_SELECT_ONE" title="Application Name" hidden="false"
type="selectOne" default="">
                                <description>select application</description>
                                <objectselect>
                                        <filter value="CfgApplication" name="type"/>
                                </objectselect>
        </input>
 </dialog>
 <execution>
        <script>
                log('test select one' );
        </script>

 </execution>
```

Input File for Install Profile

```
{
        "Step1" : {
                "APP_OBJ_SELECT_ONE" : {
                        "objectselect" : {
                                "filter" : [{
                                                "value" : "SIP_lrm26",
                                                "name"  : "name"
                                        }
                                ]
                        }
                }
        }
}
```

SelectMultiple

The input structure for a *selectMultiple* type with **<objectselect>** tag is described below:

```
{
        "Dialog name" : {
                "Input name" : {
                        "objectselect" : {
                                "filter" : [{
                                                "value" : "filter value",
                                                "name"  : "filter name"
                                        }
                                ]
                        }
                }
        }
}
```

Filters defined in an SPD input file are joined in union (*OR*) and then intersect (*AND*) with filters defined in an SPD file for a *selectMultiple* input.

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Step1">
        <input name="APP_OBJ_SELECT_MULTIPLE" title="Application Name" hidden="false"
type="selectMultiple" default="">
                                <description>select application</description>
                                <objectselect>
                                        <filter value="CfgApplication" name="type"/>
                                </objectselect>
        </input>
```

```
 </dialog>
 <execution>
        <script>
                log('test select multiple' );
        </script>

 </execution>
```

Input File for Install Profile

```
{
        "Step1" : {
                "APP_OBJ_SELECT_MULTIPLE" : {
                        "objectselect" : {
                                "filter" : [{
                                                "value" : "SIP_lrm26",
                                                "name"  : "name"
                                        },{
                                                "value" : "SIP_lrm27",
                                                "name"  : "name"
                                        }
                                ]
                        }
                }
        }
}
```

The operation returns two applications named **SIP_lrm26** and **SIP_lrm27**.

Selection Tag

The input structure for a *selectOne/selectMultiple/boolean* type with **<selection>** tag is described below:

```
{
        "Dialog name" : {
                "Input name" : {
                        "selection" : {
                                "option" : [{
                                                "value" : "option value assigned to the input
parameter",
                                                "name"  : "option name is displayed in UI"
                                        }
                                ]
                        }
                }
        }
}
```

CLC selects options defined in the SPD input file. Multiple options can be specified only for the *selectMultiple* input type.

Example

SPD Profile

```
<profile name="Install">
 <dialog step="Application Parameters">
        <input name="DATA_MODEL" title="Binary Version (32-bit or 64-bit)" default="64"
type="selectOne" required="true">
            <description>This parameter defines the 32-bit or the 64-bit version of the
binary to be deployed.  </description>
                <selection>
                    <option name="32" value="32"/>
                    <option name="64" value="64"/>
                </selection>
        </input>
 </dialog>
 <execution>
        <script>
                log('test selection support' );
        </script>

 </execution>
```

Input File for Install Profile

```
{
        "Application Parameters" : {
                "DATA_MODEL" : {
                        "selection" : {
                                "option" : [{
                                                "value" : "64",
                                                "name"  : "64"
                                        }
                                ]
                        }
                }
        }
}
```

> **Important**
>
> - If the input file does not specify a value for a SPD parameter, the value defined in the **default** attribute of the input element will be used.

- If an SPD input element has the **required** attribute set to `true`, but there is no corresponding input value that is supplied in either the SPD (as a default) or in the input file, then the SPD execution fails.
- If an SPD input element has the **readonly** attribute value set to `true`, then the value in the **default** attribute value is used for the execution, if defined. If the **readonly** attribute value is set to true, **required** is set to `false`, and the **default** attribute is not defined, then the following logic is used for input value determination:
  1. For the *boolean* input type, the input value is set to `false`.
  2. For the *string* and *password* input types, the input value is set to `""`.
  3. For the *integer* input type, the input is not propagated.
- If a dialog **cond** attribute value evaluates to false, the dialog is skipped by the CLC tool. Example:

```
<dialog step="Role input" cond="false">
  <input name="ROLE" title="Role" hidden="false" type="selectOne"
required="true">
        <description>Please indicate the role</description>
                <objectselect>
                        <filter value="CfgRole" name="type"/>
                </objectselect>
        </input>
 </dialog>
```

Exemple

```
java -jar gaxclc.jar -u:default -p:password -s -h:localhost:8080 spd execute 10054
-profileID:1 "C:/GAX/input.txt"
```

```
java -jar gaxclc.jar -u:default -p:password -h:localhost:8080 spd execute 10054
-profileName:"Install profile" "C:/GAX/input.txt"
```

```
java -jar gaxclc.jar -u:default -p:password -s -h:localhost:8080 spd execute 10054 1
-encrypted "C:/GAX/encryptedinput.txt"
```

delete

delete

Présentation

Cette action supprime une définition de solution. Si la définition de solution n'existe pas, l'action échoue.

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> spd delete SPDID
```

- `SPDID` — ID de la définition de solution à supprimer.

Exemple

```
java -jar gaxclc.jar -u:default -p:password spd delete 5436
```

Packages d'installation

La CLC prend en charge les actions suivantes concernant la fonction `ip` :

- `add`
- `query`
- `querybyid`
- `delete`

add

add

Présentation

Cette action ajoute un package d'installation (sous la forme d'un fichier .zip) à la base de données GAX. Si le package d'installation existe déjà, il est remplacé.

En cas de réussite, l'action affiche l'ID du package d'installation.

> **Important**

> Le fichier .zip doit contenir le package d'installation et le dossier de modèles du package d'installation.

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> ip add "path to IP zip file"
```

Exemple

```
java -jar gaxclc.jar -u:default -p:password ip add "C:\GAX\TESTS\zippedIpUpload\PRODUCTION\
IP_TSrvSIP64_18100079b1_ENU_windows.zip"
```

query

query

Présentation

Cette action interroge tous les packages d'installation et affiche un tableau qui répertorie les détails suivants sur chaque package d'installation :

- Numéro d'ID
- Nom
- Version
- SE
- Paramètres régionaux
- Etat

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> ip query
```

Exemple

```
java -jar gaxclc.jar -u:default -p:password -s -h:132.45.43.45:443 ip query
```

querybyid

querybyid

Présentation

Cette action interroge un package d'installation à l'aide de son ID et affiche un tableau qui répertorie les détails suivants :

- Numéro d'ID
- Nom
- Version
- SE
- Paramètres régionaux
- Etat

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> ip query IPID
```

- `IPID` — ID du package d'installation à supprimer.

Exemple

```
java -jar gaxclc.jar -u:default -p:password -h:132.45.43.45:8080 ip query 543
```

delete

delete

Présentation

Cette action supprime un package d'installation.

Format

```
java –jar gaxclc.jar -u:user -p:password -s -h:<host>:<port> ip delete IPID
```

- `IPID` — ID du package d'installation à supprimer.

Exemple

```
java -jar gaxclc.jar -u:default -p:password ip delete 547
```