Genesys Administrator Extension Help

Using the Command Line Console (CLC)

4/6/2026

Using the Command Line Console (CLC)

Contents

The Command Line Console (CLC) enables administrators to use the command line to execute certain GAX functions on solution definitions (SPDs) and installation packages (IPs). For example, you might use the CLC to silently deploy SPDs onto remote hosts.

You must be able to access the operating system's command-line interface to use the CLC. If you are not on the GAX host machine, you must have the CLC tool (**gaxclc.jar**) available on the local machine.

To access CLC's embedded Help file, execute one of the following commands:

```
java –jar gaxclc.jar help

java –jar gaxclc.jar ?
```

> **Important**
>
> As you execute commands with CLC, a log file is generated in the same location as where the tool is executed.

Structure

CLC supports commands that use the following structure:

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

In the above example:

- `-u:user` is the user name to log in to Configuration Server.
- `-p:password` is the password to log in to Configuration Server. CLC assumes there is no password if this flag does not specify a value.
- `-s` instructs CLC to use a secure *https* connection to the GAX server. If this flag is not specified, CLC uses *http*.
- `-h:<host>:<port>` specifies the host and port of the GAX server. If this flag is not specified, CLC uses the following value: `-h:localhost:8080`.
- `<function>` can be either `ip` or `spd`.
- `<operation>` specifies the operation to be executed. The valid values for this flag are specific to the function you specified in the previous step (`ip` or `spd`).
- `<args>` specifies the operation arguments. The valid values for this flag are specific to the `<function>` and `<operation>` parameters you specified in the previous steps.

The following is an example of a CLC command:

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

SPDs

CLC supports the following operations for SPDs:

- `add`
- `query`
- `querybyid`
- `execute`
- `delete`
- `encrypt` (see `execute` tab)

add

add

Overview

This operation adds an SPD to the GAX database. If the SPD already exists, as determined by the name and version in the SPD XML, this operation replaces the existing SPD.

If successful, the operation returns the ID of the added SPD.

Format

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

- `"file path"`—The path to the XML file.

Example

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

query

query

Overview

This operation queries all SPDs and displays a table that lists the following for each SPD:

- ID number
- Name
- Version
- Tenant DBID

The following is an example:

```
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
```

Example

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

querybyid

querybyid

Overview

This operation queries an SPD by its ID. If the SPD does not exist, the operation fails.

If successful, the operation displays a table that lists the following details about the SPD:

- Profile ID
- Name

For example:

```
1 Install
```

Format

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

- SPDID—The ID of the SPD that is being queried.

Example

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

execute

execute

Overview

This operation executes a SPD.

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—The ID of the SPD to be executed.
- profileName|-profileID:profileID|-profileName:profileName—The SPD profile to be executed.

> **Important**
> If no flag is specified, then profileName is assumed as the SPD profile to be executed.

- -encrypted—If specified, indicates if the input file is encrypted.

[+] Show Usage

CLC provides encryption support for input files that include sensitive data such as passwords.

Format:

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

The encrypted input file is stored in the location specified by "encrypted output file path". If the file already exists at this location, it is overwritten.

Example:

```
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"`—Specifies the input file that contains SPD parameters. If `-encrypted` is set, the input file is encrypted.

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.

[+] Show usage

STRING Type

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 Type

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 Type

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 Type

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 Type

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 Type

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**.

SELECTONE Type

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>
```

Example

```
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

Overview

This operation deletes an SPD. If SPD does not exist, the operation fails.

Format

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

- `SPDID`—The ID of the SPD to be deleted.

Example

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

IPs

CLC supports the following operations for the `ip` function:

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

add

add

Overview

This operation adds an IP (packaged as a .zip file) to the GAX database. If the IP already exists, it is replaced.

If successful, the operation displays the ID of the IP.

> **Important**
> The .zip file must contain the IP and the templates folder for the IP.

Format

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

Example

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

query

query

Overview

This operation queries all IPs and displays a table that lists the following details for each IP:

- ID number
- Name
- Version
- OS
- Locale

- Status

Format

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

Example

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

querybyid

querybyid

Overview

This operation queries an IP by its ID and displays a table that lists the following details:

- ID number
- Name
- Version
- OS
- Locale
- Status

Format

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

- `IPID`—The ID of the IP to query.

Example

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

delete

delete

Overview

This operation deletes an IP.

Format

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

- IPID—The ID of the IP to delete.

Example

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