# CloudStorage

# CLI User Manual

Made by Neit Consulting, s.r.o.

Date of last update: 4. 4. 2017

Version: 2.0

# Summary

# A. Preface

Manual serves as a guideline for working with the client CLI that enables process data for archiving. The CLI client is controlled from the command line to allow its batch and automated processing and must be installed on each computer from which the data will be archived. We designate such a computer as a Client Station. The CLI client is written in Java 8 and is fully compatible with massively used systems.

The CLI client can back up any files and folders, connect to the database, and export the data to a CSV file that is then processed as a regular file using a user-defined query. Currently supported databases are Oracle, MSSQL, MySQL, PostgreSQL.

You can use recursive processing of all subfolders when processing folders. In data processing, CLI client data can also be compressed.

For the CLI client to process the data, it needs a configuration file called the "FTD property file", with the FTD abbreviation "File type definition", the Czech translation is "Definice Typu Souboru". This file contains all settings for the correct processing of archived data. The CLI client can download and validate these files from the Neit Metadata server.

In the case of a malfunction, the CLI client can send an error mail to a predetermined email, such as an administrator.

Each client station must be registered with Neit Metadata Server, which is implemented using IP Addresses. Communication between the CLI client and the Neit Metadata server is secured by SSL encryption with RSA certificates.

## A.1. Pre-installation requirements

- Oracle Java 1.8 a higher, JRE version also works
    - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Optimal allocated memory RAM: 256MB (default)

## A.2. Installation

- Upload the CLI client folder to the client station.
- There are certificates for encryption in the CLI folder. There are default, which can be replaced by the custom you create. A brief procedure for creating custom certificates for CLI:
    - Private KeyStore (cakeystore.jks) for keypair backup, not only for CLI, but also for Master a Slave Servers.
    - Own Keystore (csKeyStore.jks)
    - Own TrustStore (csTrustStore.jks) must contain certificate signed by certification authority.

- Generate self-signed RSA keypair (client_priv_publ_self-signed.p12) to private KeyStore. Afterwards export keypair for another import
- Import keypair to KeyStore (csKeyStore.jks)
- Export X.509 public certificate (client_publ_self-signed.cer) from keypair -> then import to TrustStore (csTrustStore.jks)
- While creating custom certificates, it is necessary to change configuration in file tomcat-users.xml.

- Content of configuration file (config.properties) is used for client configuration. For example, this configuration file includes default settings for maximum upload times and attempts, which are recommended not to change if your needs are not different. For proper CLI functionality, the following items must be set:

  - mtdURL – Neit Metadata server address
  - downloadTimeout – defines timeout and waiting time for server response, default value = 7200
  - uploadConfirmationTimeout – defines waiting time to confirm successful upload to Cloud, default value = 7200
  - downloadWaitTime – waiting time for server a message about download success, default value = 20
  - uploadWaitTime – waiting time for a message from server about upload success, default value = 20
  - maxDownloadAttempts – maximal number of attempts to download, default value = 3
  - maxUploadAttempts – maximal number of attempts to upload, default value = 3
  - keyPass – password for key decryption
  - loadKeyStoreFromResources, loadTrustStoreFromResources – If true, this is a confirmation that the CLI should retrieve certificates from the internal resource, and the items listed below will remain blank. If false is set, items must be set. These are external paths to certificates and passwords to them.
    - keyStorePath, trustStorePath – path to keyStore, trustStore, if loadFromResources is false
    - keyStorePass, trustStorePass – password to keyStore, trustStore

- The configuration file "sql.properties" is used to configure to work with databases and need to fill it only if you want to process data directly from the database. In this case, you need to fill in the type and address of the database, the port on which it is running, the database name, the user, the password, and the end of the SQL query that will be used to export the data from the

database. You can use system variables to parameterize the SQL query. E.g.:

- Set sys_ftd_id=6
- db.sqlquery = SELECT * FROM filecs INNER JOIN filetypedef ON filecs.ftd_id = %sys_ftd_id%CLI usage

As has been said, the IP address of the client station must be registered and enabled on the Neit Metadata server. The registration request can be done by simply executing the command "java -jar cs.jar -g log". This operation fails, but you tell the server that they are trying to connect to an address they do not know and the server adds them between the IP addresses pending approval. Once your IP address is approved by your system administrator, you can start working normally. In practice, it may look like this:

```
C:\cloudstorage\out\cs_jar>java -jar cs.jar -g log
C:\cloudstorage\out\cs_jar\jssecacerts

Configuration Initialized
Default REST URL set to: https://localhost:443/api

REST API WEB Path set to: https://localhost:443/api/filedata/get/log
Error, your client IP address is not whitelisted on the server.
```

After adding your address to the server, it is recommended that you run help, which will make it easy for you to work with the client at first. For each parameter, examples are given for its use. Command „**java -jar cs.jar -h**“

-g / --getdf: Download the appropriate definition file type by its name (write without extension .cfg !).

-a / --alldf: Download all definition file types from the server.

-s / --save: Defines the final path where the files or file are to be stored.

-c / --defcfg: The path to the definition file when uploading to the server.

-u / --upf: File or folder for upload to server.

-r / --recurs: Parameter for upload, recursive search of folder tree structure.

-z / --zip: The upload parameter adds the files to the folder and skips it.

-q / --sqlcfg: The path to the SQL configuration file.

-n / --skipv: It skips checking to upload the file to the storage.

-t / --transc: Overrides the compression settings during transfer.

-h / --help: View help.

## Examples:

**Download all file type definitions from server:**

java -jar cs.jar -a

java -jar cs.jar -a -s D:\

**Download the appropriate definition file type by its name:**

java -jar cs.jar -g log

java -jar cs.jar -g log -s D:\

**Verify that the definition file is correct:**

java -jar cs.jar -c log.cfg

java -jar cs.jar -c /etc/cs/log.cfg

**Upload files or folders**

java -jar cs.jar -c log.cfg -u access.log

java -jar cs.jar -c log.cfg -u access\logs\

java -jar cs.jar -c D:\log.cfg -u D:\access.log

**Upload files or folders in .zip**

java -jar cs.jar -c log.cfg -z -u access\

java -jar cs.jar -c D:\log.cfg -z -u D:\access\

**Upload files from folder using recursive browse**

java -jar cs.jar -c log.cfg -r -u access\

java -jar cs.jar -c D:\log.cfg -r -u D:\access\

**Upload files from folder using recursive browse in .zip archive**

java -jar cs.jar -c log.cfg -z -r -u access\

java -jar cs.jar -c D:\log.cfg -z -r -u D:\access\

**Export SQL data to CSV file**

java -jar cs.jar -q sql.properties -s data.csv

java -jar cs.jar -q D:\sql.properties -s D:\data.csv

**Export SQL data to CSV file and it's subsequent upload on server**

By using parameter -u and path to specified file, file is not removed from disk after upload.

java -jar cs.jar -q sql.properties -c log.cfg

java -jar cs.jar -q D:\sql.properties -c D:\log.cfg -u D:\data.csv

## File type definition

File type definition (FTD) is stored on Neit metadata server and describes each one group of data defined by FTD Administrator. The FTD contains a name, a list of extensions that can be archived by the FTD, the indexing of archived data by their extensions, the authorized IP address of the client stations for the given FTD, the storage location to be archived, the list of TAGs allowed for the given FTD, and so on.

For CLI client purposes, it is only necessary to remember the name of the FTD by which we obtain the "FTD property file" from the Neit Metadata server. Which, as already mentioned, is essential for data processing. On the following lines, you will find an example of one FTD property file named CSVFile.cfg. There are examples of two tag definitions in this FTD property file. Tag values can be changed. The standard value for a tag can also include a system variable, in the form: %variable_name%.

The value of the system variable so defined is added when the CLI client is started. It is therefore possible to run the CLI client repeatedly in sequence, always changing the system variables as needed before starting. FTD administrator can make some tag value prescribe a regular expression that must comply with the value you entered. For example, if FTD administrator defines a regular expression that requires 4 digits, you cannot enter anything else in that tag.

**#DO NOT CHANGE!** - Do not change, it contains the name of ftd and the extension format of the requested file to be uploaded to the server

| | |
|---|---|
| **type=CSVFile** | File type definition name |
| **extensions=csv** | List of allowed extensions for a given FTD |
| **anyExtensionAllowed=false** | A switch that allows you to also upload file with not specified extension in FTD. |

**#FILE TYPE DATA SETTINGS** - Define tags where individual tags representing filename filters can be edited, and the file can match data types, such as For text, date, or IP address. Lines beginning with "#" are informative only.

Example 1:

| | |
|---|---|
| **#Required=true** | The switch determines whether TAG is mandatory or not. (True / false) |
| **#Regex=[a-z].*** | Regular expression that must fill in the value you fill in. FTD Administrator should provide you with more detailed information on the permitted values. |

| | |
|---|---|
| **#Description=tag description** | It is used to describe the tag. There may be information from the FTD Administrator for CLI users. For example, the allowed regular expression values. |
| **TagDefinitionIdentifier=standard value** | Tag identifier, ie its name + its default value. If after = is nothing, the tag does not have a standard value. |

Example 2:

| | |
|---|---|
| **#Required=false** | Tag is optional. |
| **#Regex=.*** | Any number of characters corresponds to it. |
| **#Description=hidden system tag** | Tag description is „hidden system tag“. |
| **NOTDESTROY=** | Tag name is „NOTDESTROY“ and has no standard value. |

## File type definition download

Downloading a FTD property file is a basic requirement for further file processing. By default, it is downloaded to the folder from which the CLI client is running. You can use the "-s ..." parameter to define a different path for the CLI Client to be stored. The FTD property file can be downloaded in two ways. It is downloading only one specific property file by name or all available property files for the client station from which the CLI client is running and save it to the "Templates" folder.

```
C:\cloudstorage\out\cs_jar>java -jar cs.jar -g typeZip

REST API WEB Path set to: https://localhost:443/api/client/filedata/typeZip
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\typeZip.cfg


C:\cloudstorage\out\cs_jar>java -jar cs.jar -a

REST API WEB Path set to: https://localhost:443/api/client/filedata/
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\Templates\typeZip.cfg
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\Templates\CSVFile.cfg
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\Templates\OracleStorageTest.cfg
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\Templates\TestWithoutIndex.cfg
Definition file downloaded and saved to: C:\cloudstorage\out\cs_jar\Templates\LocalStorageTest.cfg
```

## File upload

To upload the file successfully, use -u parameter, with a combination of the correct FTD. The -u parameter can be further combined with the -z parameters for archiving and -r for the recursive behavior of the tree structure file. There are several combinations of these parameters that are described here:

**Single file upload**– uploading single file, e.g. in csv format

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\CSVFile.cfg -u C:\csv_test\output_10K_1.csv

**Folder upload** – The client scans the folder and uploads all the files it finds. However, it does not search for folders.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\CSVFile.cfg -u C:\csv_test

**Upload folder recursively**– The client searches the folder and all of its subfolders. The found files are then uploaded to the server.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\CSVFile.cfg -r -u C:\csv_test

**Upload single file in archive** – Uploading the archive to the server. The client compresses the selected file and then sends it to the server. After uploading successfully, the temporary archive is deleted on the disk.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\typeZip.cfg -z -u C:\csv_test\output_10K_1.csv

**Upload folder in archive**– The client searches for that folder and compresses all the files into the archive, which it then uploads to the server. After a successful upload, the temporary archive will be deleted from the disk. In this folder, the client does not undergo subfolders.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\typeZip.cfg -z -u C:\csv_test

**Upload folder in archive recursively** – The client searches the folder and all its subfolders. All files will be compressed into one archive, which will then be uploaded to the server. After uploading successfully, the temporary archive is deleted from the disk.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -c ftd_templates\typeZip.cfg -z -r -u C:\csv_test

## File type definition validity verification

Verifying FTD before using it will guarantee its correctness. Control is achieved using parameter -c help nameFTD.cfg.

```
C:\cloudstorage\out\cs_jar>java -jar cs.jar -c Templates\typeZip.cfg

REST API WEB Path set to: https://localhost:443/api/client/filedata
Uploaded definition file is valid.
```

## Export SQL data to CSV file

The client also allows you to export SQL data to a CSV file. You can also export directly to the server. To do this, you must have attached database configuration information. Only -q and -s are used to select just export and save. After -q the path to the configuration file for the database is written and the name of the file to export the

csv file is written to -s. If you do not specify a path or a .csv name for the -s parameter, the name of the saved file is generated by the database name, day and time, and then saved to the current folder.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -q sql.properties -s data.csv

If the file is being exported from the database and then uploaded to the server, the -s parameter for the location is replaced by -c with the config for FTD. The file will be deleted from the disk after it has been uploaded to the server. Basically, the file is uploaded to the server in .csv format. By adding the -z parameter, the file will be compressed and sent to the server in this form.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -q sql.properties -c ftd_templates\CSVFile.cfg

C:\cloudstorage\out\cs_jar>java -jar cs.jar -q sql.properties -c ftd_templates\CSVFile.cfg -z

Using parameter -u and path to specified file, the file is not removed from disk after upload.

C:\cloudstorage\out\cs_jar>java -jar cs.jar -q sql.properties -c ftd_templates\CSVFile.cfg -u data.csv

# B. Return codes

Possible return codes when working with a CLI client. The CLI client returns only the numeric codes represented by the longer description. For more information, check the CLI client log. The following return codes are generally described in the table below.

| | | |
|---|---|---|
| **0** | **Success** | The requested operation was successful. |
| **1** | **Unspecified error from Exception** | Unexpected Application Error.<br>Consider submitting feedback. |
| **2** | **Configuration missing or empty error** | The configuration file is empty or missing.<br>Check CLI Client Configuration. |
| **3** | **Params parse error** | Failed to load specified parameters.<br>Check the entered parameters. |
| **4** | **Unspecified params entered** | Unrecognizable parameters have been entered.<br>Check the entered parameters. |
| **5** | **Invalid combination of params entered** | Incompatible combinations of parameters were entered. Correct the specified parameters. |
| **6** | **RestClient initialization error** | Client could not load SSL configuration.<br>Check CLI Client Configuration. |
| **7** | **Configuration Master Server URL invalid or empty** | Unable to connect to Master Server URL. Check the client's CLI configuration to see if the URL is incorrect or missing. |
| **8** | **KeyStore or TrustStore path not set** | There are no paths for individual SSL certificates. Set paths to SSL certificates in client CLI configuration. |
| **9** | **Configuration Exception** | Error in configuration file.<br>Check CLI Client Configuration. |
| **10** | **Definition config file required params missing or invalid** | Mandatory parameters are not set in the configuration file.<br>Check CLI Client Configuration. |
| **11** | **MD5 validation error** | File upload failed.<br>Repeat the action. |
| **12** | **Received invalid data from server** | Error receiving data.<br>Consider submitting feedback. |
| **13** | **File extension mismatch with File data extension** | The file cannot be sent because its extension is not allowed on the server.<br>Enable the extension on the server or use another FTD. |

| | | |
|---|---|---|
| **14** | **Maximum attempts reached for job** | The maximum number of attempts to transfer data has been exceeded.<br>Check the error messages in the log. |
| **15** | **Error server response** | The action did not happen as expected. Check the error messages in the log, and consider submitting feedback. |
| **16** | **Maximum waiting time reached for job** | Check the error messages in the log. |