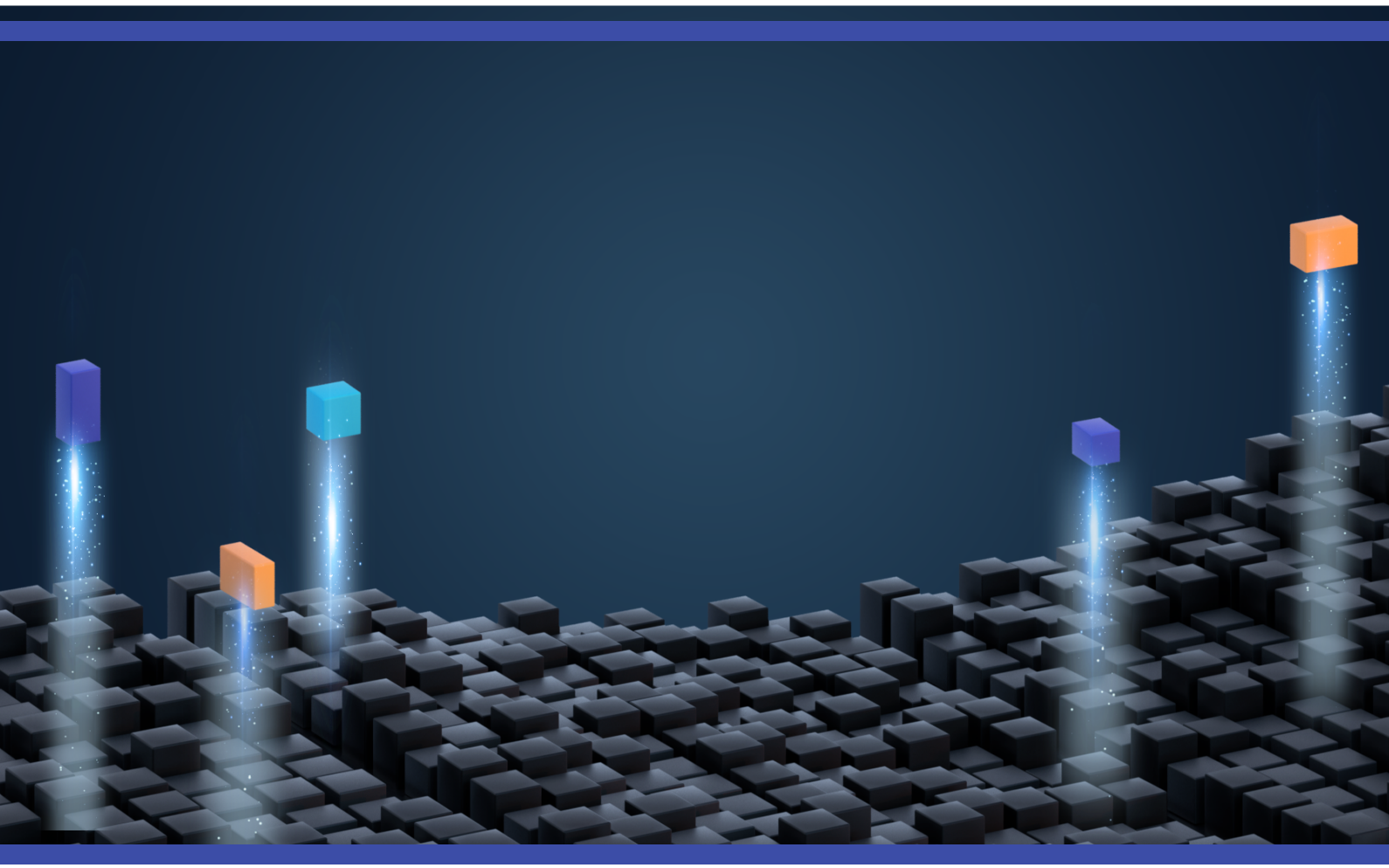
OpenLegacy
Digital to the Core

OpenLegacy / BigID

DATASET (VSAM) ADAPTER SETUP

Customer Success EMEA



Contents

Overview

Introduction

- The Hub is the OpenLegacy design tool. It's available as SaaS, full on-prem application (Hub Enterprise) and a light version (Light Hub).
- A Hub account is required by the OpenLegacy / BigID workflow.
- The Hub documentation is available here: https://docs.ol-hub.com/docs/getting-started-with-openlegacy-hub

A user API key will be needed to configure the OpenLegacy CLI.
Follow these instructions to generate one:
https://docs.ol-hub.com/docs/generating-api-keys

Prerequisites

Prerequisite / Task	Description
Mainframe FTP server setup	OpenLegacy leverage the mainframe FTP server to submit JCL jobs. The FTP server must be properly configured.
.fileparm library setup	For each file to be scanned, the Mainframe team must ensure a corresponding Member has been added .fileparm library describing the Name, DCB characteristic and space allocation of the file.
Mainframe side JCLs	The Mainframe team must prepare some JCL templates based on examples we provide.
Hub Account	Make sure you either have a OpenLegacy SaaS Hub, Enterprise Hub or Light Hub account.

The OpenLegacy CLI

The OpenLegacy CLI (ol) is the command line interface used to communicate with the Hub.
The CLI is available both for Windows and Mac/Linux. The installation instructions are available here:

- **Windows:** https://docs.ol-Hub.com/docs/install-openlegacy-cli-windows
- **Mac/Linux:** https://docs.ol-Hub.com/docs/install-openlegacy-cli-mac

Connect the OL CLI to the Hub

Enter the following command to connect the CLI to the Hub:

```
$ ol login <enter>
api-key: <enter your Hub user api key here> <enter>
```

CLI Commands

Usage: **temp> ol [OPTIONS] COMMAND [ARGS]**

'

Command	Purpose
add	Add Asset to Module metadata
clone	Clone metadata
config	Configure the CLI or module
create	Create metadata
delete	Delete metadata from the OL-Hub
export	Export metadata from OL-Hub
import	Import metadata to OL Hub
generate	Generate Service source code based on a project and a generator
login	Login to OL-Hub
logout	Logout from OL-Hub
list	List modules, projects, connector, etc.
push	Push metadata to OL-Hub
templates	Template operations
test	Test an asset or connection
usage	Get current tenant objects counters

Supported Options (General)

- -v, -version, --version displays the OpenLegacy CLI version
- -h, --help General option for all commands which prints information of the command including the command usage, sub commands, and command options.
- --verbose General option for all commands which enables more informative mode.
  This mode will print info level logs and exceptions to the terminal. In case of exception, stack traces will be printed as well.

Supported Commands

Add

Adding an asset or multiple assets to a module by creating metadata from legacy sources.
The legacy source can be located locally and the metadata from this source is created by parsing the source, or

it can be located at the backend, and then metadata is created by fetching the source using connection properties and parsing the fetched source.
The **Add** command has dynamic options. The dynamic options are created from the connection properties that are inside the `module.json` file.
Each connection property will be an option for the command and assigning a value for that option will write its value in the `module.json` file.
When using a connector that needs to fetch the source from the backend/datasource, e.g., `SAP, STORED_PROC`, etc., we need to use the connection properties options to fetch the source, parse it and create metadata.
Important note: the **Add** command can be executed only from the context of the module (i.e., the directory contains the `module.json` file). Use the cd command to change directories

- Options:
  - -a | --add-args:
    specifying an argument the connector can use to add the correct source to the module.
    For example, in a cics-cobol connector, the argument value is the path to the COBOL source file, while in a SAP connector, the argument value is a BAPI name.
  - --profile
    Select a custom profile to be used for connection. The default profile will be used if omitted.
  - dynamic options created by module connection properties.
- Usage:
  - `temp> ol add --add-args ./itemde.cbl`
  - `temp> ol add -a bapi_get_customer --host 1.1.1.1 --router 800`
  - `temp> ol add -a bapi_get_customer --profile qa`

Clone Module

Clones a module from OL Hub into a newly created directory.
The cloned module will contain all assets (operations/entities) related to the module, together with their test cases.

- Rules:
  - user must be logged into the Hub.
  - module must exist in the Hub.
  - directory should not exist, and if it exists, it must be empty.
- Options:
  - --name: module name to be cloned from the Hub
- Usage:
  - `temp> ol clone module --name sap-module`

List Connectors

List available connectors that will be used for creating module context, creating metadata from sources, testing connection to a datasource, and testing operation.
Each connector is displayed with name and version. In some cases, we might have two connectors with same name but with different versions. In such cases, the two connectors are different and separate connectors as the higher version might contain improvements, bug fixes, etc.
Another special ability of connectors is that they are pluggable, i.e., we can add and update connectors without installing a new CLI. The connectors are loaded when executing a command.

- Usage:
  - `tmp> ol list connectors`

List Generators

List the generators that exist in OL Hub for the logged in user.
The result will be a table containing the following data on each existing generator:

- generator name

- Usage:
    - `tmp> ol list generators`

List Modules

List the modules that exist in OL Hub for the logged in user.
The result will be a table containing the following data on each existing module:

- module name
- description

When providing an `additional` option, the result will be a table containing the following data on each existing module:

- module ID
- module name
- description
- modified by
- last modified
- number of assets in the module
- Options:
    - -a, --additional Display additional information
- Usage:
    - `tmp> ol list modules`

List Projects

List the projects that exist in OL Hub for the logged in user.
The result will be a table containing the following data on each project:

- project name
- project description

When providing an `additional` option, the result will be a table containing the following data on each existing project:

- project ID
- project name
- description
- modified by
- last modified
- number of assets in the project.
- Rules:
    - Must be logged in to OL-Hub.
- Options:
    - -a, --additional Display additional information
- Usage:
    - `tmp> ol list projects`

List Profiles

List the profiles that exist in the Hub for the logged in user
The result will be a list of all defined profiles with content that can be viewed with ol config module

- Usage:
    - `tmp> ol list profiles`

List Tests

Print a table with data on test assets of type `Operation` .

- Rules:
  The command can run only in the Module context folder.
  When no asset name is supplied, the command will print all asset' data.

- Usage:
  `ol tests`

Asset Name	Test-Cases	Status	Last Tested
asset-1	case-1	Failed	YYYY-MM-DD HH:MM:SS
asset-2	case-1	Not Tested	N/A
asset-2	case-2	Success	YYYY-MM-DD HH:MM:SS

`ol list tests asset-1` (asset name can be auto completed)

Asset Name	Test-Cases	Status	Last Tested
asset-1	case-1	Failed	YYYY-MM-DD HH:MM:SS

List Templates

Get list of available templates that can be used when generating a service for the logged in user.

The result will be a table containing the following data for each existing template:

- template name
- status

If a custom directory does not exist, the user will get a warning in verbose mode

Template statuses:

- `DEFAULT` - template will be used as is from plugin
- `DRAFT` - template is saved to custom template directory, but not changed yet
- `CUSTOM` - template is changed by user
- Options:
  - `--path|-p`: Path to custom templates directory. Default value is `~/.ol/cli/templates`
- Usage:
  - `tmp> ol templates`
  - `tmp> ol templates --path=/my/custom/templates/path`

Create Module

Create an empty module. The module will be the primary context to work with the CLI. This command has two required options:

- `name` – defines the module name
- `connector` – defines a selected connector that will tie the module to a specific datasource.

The module is tied to a selected connector and cannot contain metadata of different datasources other than the selected connector.

The result of executing the command is a new directory with the specified name, and a metadata file that describes the module called `module.json`.

The module metadata file contains information such as module name, connector name and version, and connection properties of the connector.

The OpenLegacy CLI has a special characteristic – it is pluggable. This influences the **`create module`** command by creating the connection properties of the module dynamically according to the selected connector. This

means, for example, that if you choose to work with a **`cics-cobol`** connector for **module1** and a **`sap`** connector for **module2**, each module will contain different connection properties inside its `module.json` file.

- Options:
  - `--name`: provide the module name
  - `--connector`:
    Provide the connector that will tie the module to a datasource.
    This option can be written in two ways:
    - Specifying the connector name with connector version as follows:
      `connector-name:connector-version`
    - Specifying the connector name alone without connector version. This tells the CLI to find the **highest** version of the connector by given name.
- Usage:
  - tmp> ol create module --name mf-module --connector cics-cobol:1.0.1
  - tmp> ol create module --name sap-module --connector sap

Generate

Generates a project by given project name and generator name. The command defines what project to generate, and the generator with which to generate it.
When executing this command, the CLI first fetches the project metadata by the project name from the Hub, and then uses the selected generator with the fetched project metadata to create a project based on that generator. If the user has customized templates for this generator, they will be used as default templates, unless the user has explicitly indicated (`--skip` or `--skip-all`) not to use custom templates,
The project is generated into a new directory that is created with the project name and all the generated sources. The generated project should be compileable and runnable without any problems.

- Rules:
  - Project name must exist in the Hub for the user (best practice to use `ol list projects` command to get list of available projects).
  - Generator name must be available as a solution to the CLI (best practice to use `ol list generators` command to get list of available generators).
- Options:
  - `--project`: provide the project name
  - `--generator`: provide the solution name
  - `--templates` <path>: provide path to destination folder which contains custom templates. If path does not exist, an exception will be thrown
  - `--skip` <custom templates list>: skip given custom templates
  - `--skip-all`: skip all custom templates
- Usage:
  - `tmp> ol generate --project mf-api --generator springboot-java-rest-maven`
  - `tmp> ol generate --project mf-api --generator springboot-java-rest-maven --skip api-ol-config.yml.ftl`

Login

Perform login to OL Hub.
Upon successful login, stores a file containing the token located at `{user-home}/.ol/.dt-cli/ol-cli-config.json`.
If the user does not specify the **-u** or **-p** options, a command prompt will ask the user to provide the credentials. For a password command prompt, the provided value from the user will be hidden to ensure a “safe” experience.

- Options:
  - -u: provide the username
  - -p: provide the password

- Usage:
    - `tmp> ol -u myuser@gmail.con -p 12345`

Logout

Perform logout from OL Hub. After executing this command, the `ol-cli-config.json` that stores the user token will be deleted.

- Usage:
    - `tmp> ol logout`

Config Module

Modifies and prints connection properties profiles.
If a non-default profile selected, the provided changes will be set to the selected profile. Later profiles can be used in `ol add` and `ol test` commands
Important note: Non-default profiles may contain only part of the required configuration. When the selected profile is merged with the default profile, some values may be overridden.

- Arguments:
    - `PROFILE_NAME` Specifies profile to be changed, defaults to Default profile if omitted
- Options:
    - dynamic options by selected connector
- Usage:
    - `tmp> ol config module`
    - `tmp> ol config module dev`
    - `tmp> ol config module dev --baseUrl http://path.to.db/dev`

Delete Config

Deletes the specified connection properties profile.

- Arguments:
    - `PROFILE_NAME` Specifies profile to be deleted. Note that the default profile cannot be deleted
- Usage:
    - `tmp> ol delete profile custom`

Push

Parent command for module and operation sub commands. Does nothing other than groups logically commands that define things we can push to OL Hub.

- **Sub-commands:**
    - **Module**
      Creates or updates a module in OL Hub by sending the current module context, with all contained operations and their test cases.
      Creating a module in OL Hub is done by pushing a module where no other module with the same name exists.
      Updating a module is done by pushing a module where that module name already exists.
        - Rules:
            - Must be inside module context (module root directory).
        - Options:
            - `--omit-sensitive` : Sensitive connection properties named (name is equal to :user, username, password) will be omitted and not get pushed to the OL Hub. Validation works throughout the depth of the “connectionProperties” object hierarchy.
        - Usage:
            - `tmp> ol push module`
            - `tmp> ol push module --omit-sensitive`
    - **Operation**
      Adds or updates an operation to an existing module in OL Hub.
      The main purpose of this command is to push the operation with a **description** (to give a

business/logical meaning to the operation) where it cannot be done with the push module command.
Adding a new operation is done by pushing an operation that does not exist for the specified module by `-m` option.
Updating is done by pushing an operation with data that are different from the data that exist in the OL Hub for that operation. The differentiation can be through the operation metadata itself, change data in test case, or added test case.

- Rules:
  - Must be inside module context (module root directory).
  - Operation must exist locally inside the module.
- Options:
  - `-m` | --module: Provide the module name
  - `-f` | --file: provide relative path to operation metadata file
  - -d | --description: Provide a description that gives a business/logical meaning to the pushed operation.
- Usage:
  - `tmp> ol push operation -m my-module -f operations/itemde/itemde-operation.json`
  - `tmp> ol push operation -m sap-module -f bapi-transfer-operation.json -d "this asset defines contract for transferring money between two accounts"`

Test Connection

Test connectivity to the datasource.
This command has the same dynamic options as the `add` command. Please refer to the **`Add`** command above.

- Options:
  - `--profile` Select custom profile to be used for connection, default profile will be used if omitted
  - dynamic options by selected connector
- Usage:
  - `ol test connection --baseUrl http://192.86.32.142 --port 12345 --uriMap oldist3 --codePage CP037 --live true`
  - `ol test connection`
  - `ol test connection --profile qa`

Test Asset

Test the ability to invoke an operation, program, or service in the datasource and get a response.
The test operation command uses the selected connector with the edited connection properties in the `module.json` file, and the `in.json` in the operation test data cases, to perform a real test against the datasource.
The `in.json` file is used for the input of the test.
The result of the test operation can be:

1. `status code 200` → Updates test `metadata.json` file with status 200 and an empty error message. Also updates the `out.json` file with the successful response.
2. `status code 500` → Updates test `metadata.json` file with status 500 and with error message. does not change the `out.json` file content.
3. failure

- Rules:
  - Must be inside module context (module root directory).
  - Operation must exist.
  - `File test_data/case-1/in.json` must exist.

- Options:
  - `-o | --operation-name`: Provide the operation name to test.
  - `--profile` Select custom profile to be used for connection, default profile will be used if omitted
- Usage:
  The case number can be specified but if no number is provided, then the first case will be tested.
  - `tmp> ol test-operation -o itemde`
  - `tmp> ol test-operation --operation-name bapi-getcustomers`
  - `tmp> ol test-operation --operation-name bapi-getcustomers. --profile dev`

Delete Test Case

Delete test case used to delete a single test case of an asset.

- Rules:
  The command can run only in the `Module` context folder.
  Must supply asset name and test case name.
- Usage:

```
ol delete test ASSET_NAME TEST_CASE
```

Example of usage: `ol delete test asset-1 case-1`

Templates Edit

Make specified templates custom by taking the default template from the plugin and saving it to the custom templates directory. If the template already exists in directory, an exception will be thrown, unless the --force option is used. Then the user can use their favorite text editor to edit the template.

- Options:
  - `--path|-p`: Path to custom templates directory. Default value is `~/.ol/cli/templates` , if path does not exist, it will be created automatically
  - `--force|-f`: Force override custom template
- Usage:
  - `tmp> ol templates edit api-ol-config.yml.ftl api-pom.ftl build-jar.launch.ftl`
  - `tmp> ol templates edit api-ol-config.yml.ftl --force -p /my/custom/templates/path`

Export Repository

Export repository metadata used to fetch all the repository metadata to a local machine to backup or share it between other tenants or environments.
The metadata include the following resources:

1. Projects - The projects with their modules, contracts, and methods.
2. Modules - The modules of the exported projects (or all of them if no specific project specified), their assets, test cases, and the connection properties (optional)

The exported file is generated with orz extension (openlegacy repository zip) and includes:

1. `.ol-Hub-header.json`: General information about the Hub:
   a. Hub URL
   b. Version (Hub version, core version, and dtf version)
   c. Timestamp
   d. User
2. `.ol-Hub-body.json`: The repository metadata

Rules:

- The user must be logged into the OL Hub.
- If one or more project is specified, the project must exist in the Hub with a valid name. (the name needs to be longer than 3 characters, start with a small letter, and include only small letters, numbers, dashes, or underscores.)
- The path must exist and must be a directory with `write` privileges.

Arguments:
List of project's names to be exported. If there is no project name, we will export all the project and module metadata.

Options:

- `--file, -f` (optional): Define the file name. The default name is `ol-Hub-export-timestamp.orz.`
- `--path, -p` (optional): Define different path to save the file. The default path is the current exception path.

Flags:
`--connection, -c` (optional): Include the connection properties for all modules if specified, otherwise not. The default for this flag is `false`

Usage:
```
ol export repository <LIST OF PROJECT'S NAME> --file exported-file-name --path ./
--connection
```

Import Repository

The Import repository command is used to copy or update existing repository metadata from a local machine to the Hub.
Matching of objects for comparison is done by ID, therefore it is important to use the corresponding ID of objects for updating. Otherwise, the system will create new objects.
Only those Projects, Modules, or Assets that have been added or modified will be imported and will be persisted in the database.
The CLI will report the result of the import to the console:
Response Example:
```
Successfully imported the repository to the Hub
ASSET <AssetName> CREATED
MODULE <ModuleName> UPDATED
PROJECT <ProjectName> CREATED
```
Repository metadata can be generated using the Export Repository command (see above) and includes the following resources:

1. `Projects` - The projects with their modules, contracts, and methods.
2. `Modules` - The modules of the exported projects (or all of them if no specific projects specified), their assets, test cases, and the connection properties (optional).

The imported file must have an "orz" extension (OpenLegacy Repository Archive file) that includes:

1. `.ol-Hub-header.json:` General information about the Hub like:
    a. Hub URL
    b. Version (Hub version, core version, and dtf version)
    c. Timestamp

    d. User
2. `.ol-Hub-body.json`: The repository metadata

Rules:

1. The user must be logged into the Hub.
2. The path must exist and must point to orz file with reading privileges.
3. Naming conflicts – you cannot import a resource (project/module/asset in a module/method in a contract) with the same name in the destination (origin) but with a different ID.
   e.g., - if we have project with name = “abc” and ID = “1111“, and we import a different project with name = “abc” and ID = “222” - we will get a conflict error.

Arguments:
The imported repository file path must exist and file name must include "orz" extension.
Flags:
`--connection, -c` (optional): Include the connection properties for all modules if specified, otherwise not. The default for this flag is `false`
Usage:

```
ol import repository </PATH/TO/OL-REPOSITORY-ZIP.orz> --connection
```

Usage

Get current tenant objects counters vs license limits

- Usage:
  - `tmp> ol usage`
- Response example:

```
Your OL Hub account usage:
3 Project/s
2 Module/s
2 Asset/s
3 Contract/s
3 Method/s out of 40 licensed
0 Generated service/s
```

Currently, the **usage** command considers only methods usage. This response shows that tenant used three (3) methods from 40 methods limited by license.

OpenLegacy BigID Dataset API

Pre-Requisites

- BigID Scanner
- OpenLegacy Hub Tenant - SaaS/On-Prem
- Relevant OpenLegacy Hub Mainframe-Dataset Modules
- Relevant OpenLegacy Hub Project containing the Modules mentioned

Specifications

The MFDS connector receives a data request from the data consumer, converts it into JCL statements, and transfers them to the mainframe data source via FTP. The JCL statements trigger the MF job and return the Job ID via FTP.
Then, the connector, using FTP commands, polls the job log in predefined intervals and checks if the job ended successfully. If it does, the connector retrieves the output file via FTP, processes the data, and sends it back to the data consumer.
The submitted Job will look as follows:

```
//OL5TEST1  JOB  A123,'CICS COBOL',CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID
//*
// SET HLQ=BATCON
// SET VERSION=V0000001
// SET CUSTID=CN8HD631
// SET FILEID=A2300001
// SET INCLUDES=&HLQ..&VERSION..INCLUDES
// SET FILEPARM=&HLQ..&VERSION..&CUSTID..FILEPARM
// SET PREPROC=NULLPROC
// SET POSTPROC=NULLPROC
//*
// EXPORT SYMLIST=(V00)
// SET V00=''
//*
//INCS JCLLIB ORDER=(&INCLUDES,&FILEPARM)
//          INCLUDE MEMBER=&FILEID
//          INCLUDE MEMBER=&PREPROC
//          INCLUDE MEMBER=MODLSORT
//*
//SYSIN     DD   *,SYMBOLS=EXECSYS
    OPTION COPY,SKIPREC=0,STOPAFT=1
    OUTFIL FNAMES=(OUT1)
/*
//          INCLUDE MEMBER=&POSTPROC
//
---
```

The given JCL (Job Control Language) syntax represents a mainframe job that performs various tasks. Let's break down the components of this JCL:
`//OL5TEST1 JOB A123,'CICS COBOL',CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID` is the Job Card:

- OL5TEST1 is the job name.
- A123 is the account number or job owner.
- 'CICS COBOL' is a job description or comment.
- CLASS=A specifies the job class (priority).
- MSGCLASS=H sets the output message class to H.
- NOTIFY=&SYSUID sends job completion notifications to the user's ID.

Then we have the Symbolic Parameters:

```
// SET HLQ=BATCON
// SET VERSION=V0000001
// SET CUSTID=CN8HD631
// SET FILEID=A2300001
// SET INCLUDES=&HLQ..&VERSION..INCLUDES
// SET FILEPARM=&HLQ..&VERSION..&CUSTID..FILEPARM
// SET PREPROC=NULLPROC
// SET POSTPROC=NULLPROC
```

- HLQ - stands for "High-Level Qualifier." In mainframe systems, the High-Level Qualifier is a user-defined prefix used to identify datasets or libraries. It is commonly used to distinguish datasets or resources belonging to a particular system or application. The specific value assigned to HLQ, in this case, is "BATCON," which would be used as a prefix for dataset or library names in subsequent JCL statements.
- VERSION, CUSTID and FILEID are values used as dataset names for INCLUDES and FILEPARM
- INCLUDES and FILEPARM are derived from other parameters using concatenation.
  -- In our Example INCLUDES value is BATCON. V0000001.INCLUDES which is a dataset name containing chunks of JCL code to include when constructing the final JCL to be submitted to the JES2 internal reader, those are included in the line "//INCS JCLLIB ORDER=(&INCLUDES,&FILEPARM)":

In our Example FILEPARM value is BATCON.V0000001.CN8HD631.FILEPARM which is the dataset name containing one member for each file, for example for FILEID=A2300001 is expected to have a member named A2300001 which is included below the line "//INCS JCLLIB ORDER=(&INCLUDES,&FILEPARM)" and what does is setting variables used when doing the SORT, below the contents of our tests dataset and one example member:

F00 thought F.09 are used in chunks of JCL included, for example:

And the sort output:

- PREPROC and POSTPROC represent additional member names, this is used when the file needs a pre-processing or post-processing steps.

Then we have the Include Statements:

```
//INCS JCLLIB ORDER=(&INCLUDES,&FILEPARM)
//           INCLUDE MEMBER=&FILEID
//           INCLUDE MEMBER=&PREPROC
//           INCLUDE MEMBER=MODLSORT
```

- INCS JCLLIB specifies the JCL library concatenation.
- ORDER=(&INCLUDES,&FILEPARM) defines the order in which libraries are searched for INCLUDE statements.
- INCLUDE MEMBER=&FILEID includes the member specified by FILEID.
- INCLUDE MEMBER=&PREPROC includes the member specified by PREPROC for pre-processing.
- INCLUDE MEMBER=MODLSORT includes the member named "MODLSORT".

Next is the SYSIN (Input) Data:

```
//SYSIN     DD   *,SYMBOLS=EXECSYS
OPTION COPY,SKIPREC=0,STOPAFT=1
OUTFIL FNAMES=(OUT1)
/*
```

- SYSIN is a DD statement specifying input data.

- *,SYMBOLS=EXECSYS indicates that the data will be provided inline (in the JCL itself) and also requests execution-time symbol translation.
- OPTION COPY,SKIPREC=0,STOPAFT=1 specifies a data manipulation operation.
- OUTFIL FNAMES=(OUT1) directs the output of the operation to a dataset named OUT1.

And lastly Post-Processing Include:

```
//          INCLUDE MEMBER=&POSTPROC
```

INCLUDE MEMBER=&POSTPROC includes the member specified by POSTPROC for post-processing.

How It Works

The solution can be divided into the following parts:

- Creating the OpenLegacy Hub Project
- OpenLegacy BigID Dataset API
- Run the OpenLegacy BigID Dataset API

Creating the OpenLegacy Hub Project

For each Dataset file, you would like to Scan using the BigID Scanner. You will need to do the following steps:

In the Mainframe Side:

Ensure with your Mainframe team that they have created a Member in the `.fileparm` library describing the Name, `DCB characteristic` and `space allocation` of the relevant file you need to scan.

Make sure to ask your Mainframe team for the following details:

- The member name and set it as the FileID.
- The File type: Fixed Block/ Variable Block
- In case it is a Fixed Block, as for the record length

Creating the Module

You will need a module for each Dataset file you would like to Scan using the BigID Scanner.

The steps for creating the module are as follows:

- Create the module using the OpenLegacy CLI: `ol create module <MODULE_NAME> --connector mainframe-dataset` where `<MODULE_NAME>` is the module name you decided.
- Make sure to have the following information:
  - Member Name
  - File Type
  - In case it is a Fixed Block, ask for the record length
  - HLQ - the High-Level Qualifier is a user-defined prefix used to identify datasets or libraries
  - CUSTID - is being used as part of the created fileparam
  - JCL Job Card - in case of a multiline Job, use ;; between each new line
- Inside the newly created module directory, run the following command:
  - If the file is Fixed Block:

    ```
    ol test connection --host <YOUR_MAINFRAME_SERVER_ADDRESS> --user <FTP_USER>
    --password <FTP_PASSWORD> --file-name <FileID> --output-file-prefix
    <HLQ_Value> --output-file-suffix <CUSTID>.<FileID>.OL --default-job-card <JCL
    Job Card> --records-have-fixed-length true --record-length <RECORD_LENGTH>
    --record-id-position 0 --record-id-length 16
    ```

  - If the file is Variable Block:

    ```
    ol test connection --host <YOUR_MAINFRAME_SERVER_ADDRESS> --user <FTP_USER>
    --password <FTP_PASSWORD> --file-name <FileID> --output-file-prefix
    ```

```
<HLQ_Value> --output-file-suffix <CUSTID>.<FileID>.OL --default-job-card <JCL
Job Card> --record-id-position 0 --record-id-length 16 --record-offset
<THE_OFFSET_WHERE_THE_RECORD_IDENTIFIER>  --record-read-length
<THE_RECORD_IDENTIFIER_LENGTH>
```

- In case your file is Variable Block or it has more than one record type returning, you must specify the `--record-offset <THE_OFFSET_WHERE_THE_RECORD_IDENTIFIER> --record-read-length <THE_RECORD_IDENTIFIER_LENGTH>` properties where the TYPE INDICATOR is a value in the buffer that is unique for each record type.
- The following JCL Parameters are also configurable using the `ol test connection`:
  - HLQ: `--hlq TEXT` - the High-Level Qualifier is a user-defined prefix used to identify datasets or libraries
  - VERSION: `--version TEXT`
  - CUSTID: `-cust-id TEXT`
  - FILEID: `--file-id TEXT` - Specify the file name
  - INC: `--inc TEXT` - the submitted include statement

- Parse the relevant copybooks using the following:
  - In case of a Fixed Block File with only one record type use:
    `ol add --source-path <PATH_TO_THE_COPYBOOK>`
  - Else you will need to set the Hex Value of the Record Identifier:
    `ol add --source-path <PATH_TO_THE_COPYBOOK> --indicator-value <HEX VALUE`

Testing the Module

It would be best to test the module before pushing it to the OpenLegacy Hub, to do so, you should follow the next steps:

- Change sort2-operation json - Open assets/sort2-operation/sort2-operation.json:
  - Delete Line Number 8
- Change sort2-operation test case input Open assets/sort2-operation/test_date/case-1/in.json change it to be:

```
{
  "var0" : "",
  "fileName" : "OUT1",
  "conditions" : [ ],
  "operator" : "AND",
  "pagination" : {
    "offset" : 0,
    "limit" : 10
  },
  "hlq" : null,
  "version" : null,
  "custId" : "<CUSTID>",
  "fileId" : "<FILE_ID>",
  "inc" : null,
  "fileParam" : null
      }
```

- Run the test using: ol test asset sort2-operation

Pushing the Module

To Push the Module into the OpenLegacy Hub do the following:

- Make sure you are inside the module dirctory, run: ol push module

Creating the Project

Open the OpenLegcy Hub Application, do the following:

- On the left bar click on `Projects`
- Click on `Create New Project` on the top right corner
- Set a name for the project
- In the Bottom part where it is written Backend Modules select the module you pushed in the previous step
- Click on `Create Project`
- In case you already have a project and you just need to add a new module to it, do the following:
    - On the left bar click on Modules
    - Select the new module
    - Click on 'Add To Project'
    - Select `Add Modules to Existing Project'
    - Select the relevnt project

Configure the OpenLegacy BigID Dataset API

The OpenLegacy BigID API reads the JCL Job properties in the following order:
Global Properties that will override any other parameters You can set in the docker-compose YAML:

- GLOBAL_USER - In case you would like to use a global user property rather then using the once defined in the modules
- GLOBAL_PASSWORD - In case you would like to use a global password property rather then using the once defined in the modules
- SSL_ENABLE
- SSL_PROTOCOL
- SSL_IMPLICIT
- JOB_CARD - In case you would like to use a global Job Card rather then using the once in the modules

Other properties are taken from the module properties. If they are missing there, it should use the relevant properties from the docker-compose env. If missing there, it will use the default values, the properties we are referring to:

- JOB_CARD default value is `OL5TEST1 JOB A123,'CICS COBOL',CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID`
- VAR0 default value is ``
- FILE_NAME default value is `OUT1`
- HLQ default value is `BATCON`
- CUSTID default value is `CN8HD631`
- VERSION default value is `V0000001`
- INC default value is `&HLQ..&VERSION..INCLUDES`
- FILE_PARAM default value is `&HLQ..&VERSION..&CUSTID..FILEPARM`

Configure the OpenLegacy Hub Connection:

- If you are working against the OpenLegacy SaaS, set:
    - OL_HUB_API_KEY=<API_KEY>
    - OL_HUB_PROJECT_NAME=<PROJECT_NAME>
    - OL_CORE_LICENSE_KEY=<LICENSE_KEY>
- If you are working against OpenLegacy On-Prem, add the following:

    - OL_HUB_URL=<SERVER_ADDRESS>
- If you are using OPZ, Make sure to configure the following:
    - OL_SOURCE_PROVIDER=OL_PROJECT_ZIP
    - OL_PROJECT_ZIP_PATH=<PATH_TO_OPZ_FILE>
    - In the docker-compose, make sure to mount the OPZ file:
        - <PATH_TO_OPZ_FILE_LOCAL>:<PATH_TO_OPZ_FILE_REMOTE>

Run the OpenLegacy BigID Dataset API

To run the API, Run `docker-compose up -d`
The swagger-ui is available under: /swagger-ui.html
Available methods:

- objects:
  `curl -X 'GET' '<http://localhost:8080/objects'> -H 'accept: application/json'`
- objects describe:
  `curl -X 'GET' '<http://localhost:8080/objects/<FILE_NAME>/describe'> -H 'accept: application/json'`
- records:
  `curl -X 'GET' '<http://localhost:8080/objects/<FILE_NAME>/records?Offset=0&Count=4'> -H 'accept: application/json'`
  Offset and Count are optional
- sar:
- `example request:`
- `curl -X 'POST' \`
- `'<http://localhost:8080/objects/<FILE_NAME>/sar'> \`
- `-H 'accept: application/json' \`
- `-H 'Content-Type: application/json' \`
- `-d '[`
- `{`
- `"fieldName": "aKey.aTypeIndexField.customer",`
- `"fieldValue": "11111",`
- `"isFullMatch": true`
  `}]'`

TL;DR

Supported Environment Variables and their default values:

- ENV DEBUG_LEVEL 'info'

Hub Params

- ENV OL_HUB_API_KEY
- ENV OL_HUB_PROJECT_NAME
- ENV OL_CORE_LICENSE_KEY
- ENV OL_HUB_URL 'https://api.ol-hub.com'
- ENV OL_SOURCE_PROVIDER 'hub'

JCL Params

- ENV JOB_CARD 'OL5TEST1 JOB A123,'CICS COBOL',CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID'
- ENV VAR0 ''
- ENV FILE_NAME 'OUT1'
- ENV HLQ 'BATCON'
- ENV CUSTID 'CN8HD631'
- ENV VERSION 'V0000001'

- ENV INC '&HLQ..&VERSION..INCLUDES'
- ENV FILE_PARAM '&HLQ..&VERSION..&CUSTID..FILEPARM'

FTP Params

- ENV SSL_ENABLE 'false'
- ENV SSL_PROTOCOL 'TLS'
- ENV SSL_IMPLICIT 'false'
- ENV GLOBAL_USER ''
- ENV GLOBAL_PASSWORD ''

Record ID

- ENV RECORD_ID_START_POS '0'
- ENV RECORD_ID_LENGTH '16'

Workflow

The following are the steps required to setup a new OpenLegacy BigID project:

1. Create a new module
2. Configure the module connection profile
3. Parse the VSAM file copybook
4. Test the module
5. Upload the module definition to the Hub
6. Create a Hub project (Only required the first time)
7. Associate an additional module to the project (Only required if more then one module is used)
8. Export the project metadata from the Hub (OPZ file)
9. Update the OPZ file in the OpenLegacy nocode-bigid container

Repeat steps 1 to 7 (minus step 6) for each additional VSAM file.

Step 1 – Create a new module

```
ol create module <MODULE_NAME> --connector mainframe-dataset
```

Replace <MODULE_NAME> with an actual module name (lowercase-letters/numbers/hyphens)
This command will create a new directory named <MODULE_NAME> to contain the module metadata.
CD into the module directory.

```
cd <MODULE_NAME>
```

Step 2 – Configure the module connection profile

Each module must be configured with proper connection properties.
To view the list of available properties type:

```
ol test connection -h
```

A typical connection configuration will provide at least the following properties:

- **--host** The mainframe FTP server address
- **--port** The FTP port
- **--user** The FTP user
- **--password** The FTP password
- **--record-id-position** BigID record identifier position in bytes
- **--record-id-length** BigID record identifier length in bytes

The BigID scanner needs each record provided by OpenLegacy to be identified by a unique id.
The **record-id-position** and **record-id-length** are used to extract a portion of the record sequence to be used as a BigID identifier.

In case your mainframe side JCL properties are different than those in our example files, you'll also need one or more of the following connection properties:

- **--file-name** FileID (the member name from the .fileparm library)
- **--output-file-prefix** HLQ Value
- **--output-file-suffix** <CUSTID>.<FileID>.OL

- **`--default-job-card`** `<JCL Job Card> (For job cards that span multiple lines, use a double semicolon ;; to separate each line)`

Example:

```
ol test connection \
    --host <YOUR_MAINFRAME_SERVER_ADDRESS> \
    --port <FTP_PORT> \
    --user <FTP_USER> \
    --password <FTP_PASSWORD> \
    --record-id-position 0 \
    --record-id-length 16 \
    --file-name <FileID> \
    --output-file-prefix <HLQ_Value> \
    --output-file-suffix <CUSTID>.<FileID>.OL \
    --default-job-card "<JCL Job Card>"
```

Depending on the VSAM file type, additional configuration properties have to be set.
If the VSAM file type is **Fixed Block and has a single record type** then the following properties are required:

- **`--records-have-fixed-length`** `A flag, must be set to` **`true`**
- **`--record-length`** `The record length in bytes`

Example:

```
ol test connection \
    --host <YOUR_MAINFRAME_SERVER_ADDRESS> \
    --port <FTP_PORT> \
    --user <FTP_USER> \
    --password <FTP_PASSWORD> \
    --record-id-position 0 \
    --record-id-length 16 \
    --file-name <FileID> \
    --output-file-prefix <HLQ_Value> \
    --output-file-suffix <CUSTID>.<FileID>.OL \
    --default-job-card "<JCL Job Card>" \
    --records-have-fixed-length true \
    --record-length <RECORD_LENGTH>
```

If the VSAM file type is **Variable Block or has multiple record types** then a *discriminator* value has to be identified. The discriminator is a sequence of bytes (not necessarily a copybook field) identified by an offset and a length (in bytes) representing a value that uniquely identifies the record type.
To do so, the following properties have to be used:

- **`--record-offset`** `The offset position of the discriminator sequence`
- **`--record-read-length`** `The discriminator sequence length`

Example:

```
ol test connection \
    --host <YOUR_MAINFRAME_SERVER_ADDRESS> \
    --port <FTP_PORT> \
    --user <FTP_USER> \
    --password <FTP_PASSWORD> \
    --record-id-position 0 \
    --record-id-length 16 \
    --file-name <FileID> \
    --output-file-prefix <HLQ_Value> \
    --output-file-suffix <CUSTID>.<FileID>.OL \
    --default-job-card "<JCL Job Card>" \
    --records-offset <DISCRIMINATOR_OFFSET> \
    --record-read-length <DISCRIMINATOR_LENGTH>
```

Step 3 - Parse the VSAM file copybook

In order to generate a model representation of the VSAM file records, one or more copybooks describing the structures are required.

The OpenLegacy CLI "add" command should be used to parse the copybooks and extract the data.
*NB: The copybook file extension **must** be "cpy".*

```
ol add –source-path <PATH_TO_THE_COPYBOOK>
```

Step 4 – Test the module

The module properties may need fine tuning so it's good practice to test it before pushing the definition to the Hub. The OpenLegacy **test connection** command can be used to adjust property values.
The module directory tree should look like this:

```
<module-dir>/
    module.json
    assets/
        <your-copybook>/...
        get-by-id-operation/...
        search-operation/...
        vsam-get-by-id/...
        sort2-operation/
            sort2-operation.json
            sources.json
            test_data/
                case-1/
                    in.json
                    out.json
                    metadata.json
```

Using a text editor, edit the **<module-dir>/assets/sort2-operation/test_data/case-1/in.json** file and swap the contents with the following (be sure to replace the placeholders):

```
{
  "var0" : "",
  "fileName" : "OUT1",
  "conditions" : [ ],
  "operator" : "AND",
  "pagination" : {
    "offset" : 0,
    "limit" : 10
  },
  "hlq" : null,
  "version" : null,
  "custId" : "<CUST_ID>",
  "fileId" : "<FILE_ID> ",
  "inc" : null,
  "fileParam" : null
}
```

Enter the following command to run the test:

```
ol test asset --verbose sort2-operation
```

The command will establish a connection to the mainframe via FTP while sending debug information to the standard output, including FTP messages exchange and the generated JCL.
The JCL should be accepted and eventually executed on the mainframe. If everything is configured correctly, no error messages will be displayed and the VSAM file data will be presented in the JSON format required by BigID.

Step 5 - Upload the module to the Hub

Enter the following command to upload the module definition to the Hub:

```
ol push module
```

Step 6 - Create a Hub project

This step is only needed once. Multiple modules can be pushed to the Hub and then added to the same project using the Hub web UI.
Enter the following command to create a new project in the Hub and associate it to the first module:

```
ol create project <project-name> -m <module-name>
```

Step 7 - Associate an additional module to the project

This step is only needed in case more than one module have to be associated to the same project.
The additional modules have to be associated to the project by using the Hub web UI (Hub studio).

1. Log into the Hub studio with your account
2. Open the **Modules** section
3. Use the check boxes to select each module you want to associate to a project
4. Click on the **Add to Project** then select **Add modules to existing project**

5. Click on the target project to select it then click the **Add To Project** button

Step 8 - Export the project metadata from the Hub

The information currently stored in the Hub have to be provided to the OpenLegacy BigID application.
OpenLegacy supports both on-line metadata fetching and metadata export for off-line access.
The metadata is exported as a zip archive containing JSON files. The file extension is **opz** for **O**penLegacy **P**roject **Z**ip file so it's commonly called OPZ file.

Enter the following command to export an OPZ file:

```
ol export project -license -connection <PROJECT_NAME>
```

A file named <PROJECT_NAME>.opz will be created in the current directory.

Step 9 - Update the OPZ file in the OpenLegacy nocode-bigid container

The OpenLegacy nocode-bigid application needs to load the OPZ file by path.
How to make the OPZ available to the application depends on the deployment environment.
In case a Kubernetes orchestrator is used, the OPZ file can be stored in a persistent volume mounted in the pod.