**CS DMS Closing**

**Administrator Guide**

**Release 1.0.20220114**

**Written by Dusan Paal**

**LEDVANCE GmbH**

**March 04, 2022**

Contents

[Overview 3](#_Toc97736178)

[Requirements, installation and use 4](#_Toc97736179)

[Directory and file structure 5](#_Toc97736180)

[Application design and workflow 7](#_Toc97736181)

[Testing and error handling 10](#_Toc97736182)

[Revision 11](#_Toc97736183)

# Overview

The DMS Closing represents a cloud-based application written in python that automates updating of case parameters in “UDM\_DISPUTE” SAP transaction for those disputed cases where credit note was issued by the CS team. In brief, the process consists of the following steps:

1. inserting the credit note number into the “Status Sales” field of disputed cases if not already contained,
2. changing the value of the “Root Cause Code” field of disputed cases to “L06” where applicable,
3. changing the value of the “Status” field to “Solved” (status code 2) where applicable,
4. generating an Excel user report with the processing outcome and sending a notification to users.

Upon application start, active countries are detected. Inactive countries if any, are excluded from processing. For each active country, the accounting data containing credit notes is exported from customer accounts in the FBL5N SAP transaction. The data is then parsed and converted. Next, case IDs are extracted from the strings stored in the data “Text” field. These ID values refer to disputed cases recorded originally in UDM\_DIPSUTE. If no such ID is found during the extraction process, then the processing is stopped for the entire country. However, a report is generated from the original data for further user reference.

If any case IDs are successfully extracted, the application then searches the corresponding cases in UDM\_DISPUTE transaction. If the search returns no results, this may indicate invalid case IDs (e. g. due to typos) or a missing access to the country’s company code in UDM\_DISPUTE. Otherwise, the returned dispute data is exported and merged with the accounting data on case ID values serving as the merging key. The merged data is then evaluated, and new case parameters are generated where applicable. If there are any cases to modify found, the application enters UDM\_DISPUTE, opens each case to modify, enters the new params into the corresponding fields and saves the case settings.

Finally, an Excel report capturing the processing outcome along with the original data is generated and uploaded to a network destination folder specified by the CS team.

# Requirements, installation and use

The CS DMS Closing application requires Python 3.9.9 to run. Other 3.x.x python versions should work as well but have not been tested. Older 2.x.x python versions are not supported. The application also requires the following additional external libraries:

Pandas 1.3.4 – parsing of textual data from exported files, extraction of relevant information form the parsed data, evaluation of relevant data parameters, generating new case parameters based on the evaluation outcome

PyWin32 3.00 – connection to SAP GUI scripting engine

PyYaml 6.0 – reading application configuration and data processing parameters stored in the “yaml” file format

XlsxWriter 3.0.2 – writing processing output and raw data to reports in the Excel “xlsx” file format

Copy application files to a location of your choice. It is strongly recommended to create a new python virtual environment “env” in the application “root\app” subdirectory using the requirements contained in the “root\app\reqs.txt” file. However, if the decision is made to use an external python environment, care must be taken to ensure that all the above-stated requirements are satisfied, otherwise, the program may fail to execute properly. The path to the python interpreter specified in the application “app.bat” file must be modified, accordingly.

Open the “root\app\app.bat” file in order to launch the application. During application initialization, the program will load application configuration parameters and attempt to connect to the SAP GUI scripting engine. Should any SAP pop-up window(s) appear asking to grant access to the scripting engine, these need to be confirmed by the user manually in order to continue program execution. Otherwise, the application will be terminated with the corresponding return code (see “Error handling” section). Therefore, it is advised that the scripting option be allowed in the main SAP GUI settings before running the application.

All application events occurring at runtime are recorded in the “root\app\log.log“ file. The purpose of the log file is to record any important application runtime events to track data flow, debugging or performance profiling of the program. Each event record contains information about the occurrence time, event type and message.

The application execution can be controlled by maintaining configuration and data processing parameters stored in “appconfig.yaml” and “rules.yaml” file, respectively. Both are located to the “root\app” directory. Application configuration parameters define application folder and file paths, SAP transaction and system settings, report layout and notification settings. Data processing rules detail country-specific rules applied during data processing. Should the need of adding a new country arise, this can be easily achieved by inserting a new record with specified parameter values into the rules-containing file.

# Directory and file structure

Application scripts and data is organized into a tree-like structure of folders and files, where the root directory represents the entire project folder (see Figure 1).

**Figure 1:** The tree structure of the application files and folders.

root

doc

app

env (optional)

reports

temp

templates

scripts

biaController.py

biaDMS.py

biaFBL5N.py

biaMail.py

biaProcessor.py

biaReport.py

app.py

app.bat

appconfig.yaml

rules.yaml

logging.yaml

log.log

**Table 1:** List of application directories and files and their purpose.

|  |  |
| --- | --- |
| Name | Description |
| doc | Contains application documentation files. |
| app | Contains application execution and configuration files. |
| env | Contains python virtual environment. |
| reports | Temporarily stores generated reports. |
| templates | Contains user notification HTML templates. |
| requirements.txt | Contains a list of external python library names and versions. |
| logging.yaml | Contains event logger parameters. |
| appconfig.yaml | Contains application configuration parameters. |
| rules.yaml | Contains country-specific data processing rules. |
| app.bat | Application execution file. |
| app.py | Application main script file. |
| biaController.py | Mediates data transfer and control flow between SAP-automating modules, data processor and messenger. |
| biaSAP.py | Connects to and disconnects from SAP GUI. |
| biaDMS.py | Exports case data and updates case parameters. |
| biaFBL5N.py | Exports data from customer accounts. |
| biaProcessor.py | Performs data parsing, conversion and evaluation. Generates new case parameters. |
| biaReport.py | Generates and uploads Excel reports. |
| biaMail.py | Sends a notification to users. |

# Application design and workflow

On top of the program modular design sits the “app.py” module that represents the main execution script called form the “app.bat” file. The module hosts the “main()” procedure representing program entry point. By entering this procedure, application execution starts. Statements are executed by calling the controlling procedures:

1. biaController. initialize\_log(): loads and applies logging configuration parameters
2. biaController.load\_app\_config(): loads application configuration parameters
3. biaController.load\_processing\_rules(): loads data processing rules
4. biaController.get\_active\_entities(): returns a list of countries to process
5. biaController.export\_fbl5n\_data(): exports accounting data from FBL5N transaction
6. biaController.preprocess\_fbl5n\_data(): parses and converts data, then and extracts case IDs from data
7. biaController.export\_dms\_data(): exports disputed data from UDM\_DISPUTE transaction
8. biaController.preprocess\_dms\_data(): preprocesses accounting and disputed data
9. biaController.process\_disputes(): updates case parameters in DISPUTE transaction
10. biaController.report\_output(): generates and uploads report files to user-defined network directory
11. biaController.release(): releases all application-allocated resources and logs out from SAP GUI.

Once the program exits the “main()” procedure, a return code is received by the “app.py” script. This code is further passed onto the “sys.exit()” procedure, which allows capturing the return state by the script caller (see section “Error handling” for a complete list of return codes).

**Figure 2**: Scheme of the application design with indicated inter-modular data and control flows.

calls

app.py

- main execution script

app.bat

- application launcher

provides

control

handles response

controls report creation and upload

biaController.py

biaSAP.py

- logs in to SAP GUI application

biaReport.py

- creates and uploads Excel reports

requests SAP logon

provides session

biaFBL5N.py

- exports accounting data to file

controls data export

biaMail.py

- sends email notification to users

controls sending

of notifications

netwwork repository

upload reports

writes data

reads data

reads notification template

creates reports

notification template

temporary

report files

temporary

text data files

provides processing output

requests data processing

controls data export or case parameter update

biaDMS.py

- exports dispute data to file

- modifies dispute parameters

biaProcessor.py

- performs data-related operations

**Figure 3:** A detailed list of application module interfaces and fields ( - for private, + for public items )

**biaController.py**

+ bool init\_logger( str app\_name, str app\_ver )

+ dict init\_load\_app\_config( void )

+ dict load\_processing\_rules( dict cfg )

+ dict get\_active\_countries( dict rules )

+ GuiSession init\_sap( dict cfg )

+ void release( dict data\_cfg = None, GuiSession sess = None )

+ bool export\_fbl5n\_data( dict data\_cfg, dict sap\_cfg, dict entits, GuiSession sess )

+ bool export\_dms\_data( dict data\_cfg, str sap\_cfg, DataFrame fbl5n\_data, GuiSession sess )

+ dict preprocess\_fbl5n\_data( dict data\_cfg, dict rules, dict entits )

+ dict preprocess\_dms\_data( dict data\_cfg )

+ tuple process\_data( DataFrame fbl5n\_data, DataFrame dms\_data, dict entits, dict rules )

+ bool process\_disputes( list closing\_input, DataFrame data, GuiSession sess )

+ void report\_output( DataFrame data, dict rep\_cfg, dict notif\_cfg, dict rules )

- str \_TXT\_FILE\_ENCODING = "utf-8"

- str \_EMPTY\_STRING = ""

- Logger \_logger = logging.getLogger( name )

Reponsibilities:

Represents the main communication channel that manages data and control flow between connected highly specialized modules.

**app.py**

+int main()

Reponsibilities:

Represents the main script of the application that contains program entry procedure called at the script import.

**biaMail.py**

+ send\_message( str sender, str subj,

str body, list recips, list cc = [],

list att\_paths: = [] )

- Logger \_logger = logging.getLogger( name )

Reponsibilities:

creates and sends of emails directly

via SMTP server.

**biaSAP.py**

+ GuiSession login( str gui\_exe\_path, str sys\_name )

+ GuiSession logout( GuiSession sess )

- bool \_window\_exists( str name )

+ SYS\_P25 = "OG ERP: P25 Productive SSO"

+ SYS\_Q25 = "OG ERP: Q25 Quality Assurance SSO"

- Logger \_logger = logging.getLogger( name )

Reponsibilities:

Mediates connection to SAP GUI scripting engine.

**biaFBL5N.py**

+ bool initialize( GuiSession sess )

+ void release( void )

+ bool export( str file\_path, list cmp\_cds, str layout, str doc\_type\_filter = "DG" )

- void \_copy\_to\_clipboard( str text )

- void \_clear\_clipboard( void )

- void \_close\_popup\_dialog( bool confirm )

- void \_toggle\_worklist( bool activate )

- void \_open\_selection\_list( void )

- void \_set\_customer\_account( str val )

- void \_set\_export\_params( str path, str name, str enc = "4120" )

- void \_select\_data\_format( int idx )

- bool \_set\_layout( str name )

- GuiSession \_sess = None

- GuiFrameWindow \_main\_wnd = None

- GuiStatusBar\_main\_wnd = None

- GuiStatusBar \_stat\_bar = None

- Logger \_logger = logging.getLogger( name )

- str \_TYPE\_ERROR = "E"

- str \_TYPE\_WARNING = "W"

- str \_EMPTY\_STRING = ""

- dict \_vkeys = { str key\_names : int sap\_vkey\_codes }

Reponsibilities:

Loads and exports data located on customer accounts into a file.

**biaReport.py**

+ bool upload( str name, str src\_dir, str dst\_dir,

str dst\_subdir )

+ bool create\_report( DataFrame data, str file\_path,

str sht\_name, list field\_order )

+ str create\_notification( str templ\_path,

str summ\_path, str dst\_rep\_dir, str dst\_rep\_subdir, str summ )

+ str summarize( DataFrame data, str cmp\_cd,

str cntry )

- int \_get\_col\_width( Series vals, str fld\_name )

- str \_col\_to\_rng( DataFrame data, str first\_col,

str last\_col = None, int row = -1):

- int \_to\_excel\_serial( datetime date )

- str \_EMPTY\_STRING = ""

- Logger \_logger = logging.getLogger( name )

Reponsibilities:

Mediates connection to SAP GUI scripting engine.

**biaDMS.py**

- void \_copy\_to\_clipboard( str text )

- void \_clear\_clipboard( void )

- bool \_is\_popup\_dialog( void )

- void \_close\_popup\_dialog( bool confirm )

- void \_execute\_query( void )

- GuiGridView \_get\_grid\_view( void )

- GuiGridView \_get\_param\_mask( void )

- GuiGridView \_get\_search\_mask( void )

- GuiToolbarControl \_get\_control\_toolbar( void )

- str \_get\_case\_param( GuiGridView param\_mask, int cell\_idx, str col\_type)

- void \_change\_case\_param( GuiGridView param\_mask, int cell\_idx,

str col\_type, str val )

- bool \_find\_and\_click\_node( GuiTree tree, object node, str node\_id )

- void \_save\_changes( void )

- void \_change\_case\_status( GuiGridView param\_mask, int val )

- bool \_apply\_layout( GuiGridView grid\_view, str name)

- void \_set\_export\_params( str path, str name, str enc = "4120")

- void \_select\_data\_format( GuiGridView grid\_view, int idx )

- void \_toggle\_display\_change( void )

+ GuiGridView initialize( GuiSession sess )

+ void release( void )

+ tuple search\_dispute( GuiGridView srch\_mask, int case\_id )

+ tuple search\_disputes( GuiGridView srch\_mask, tuple case\_IDs )

+ void modify\_case\_parameters( GuiGridView grid\_view, str root\_cause = None,

str stat\_sl = None, CaseStates stat: = Original )

+ bool export( GuiGridView grid\_view, str file\_path, str layout )

- GuiSession \_sess = None

- GuiFrameWindow \_main\_wnd = None

- GuiStatusBar\_main\_wnd = None

- GuiStatusBar \_stat\_bar = None

- bool \_edit\_mode\_on = False

- Logger \_logger = logging.getLogger( name )

+ int enum CaseStates

+ str enum RootCauses

- int enum \_SearchFieldIndexes

- str \_TYPE\_ERROR = "E"

- str \_TYPE\_WARNING = "W"

- str \_EMPTY\_STRING = ""

- dict \_vkeys = { str key\_names : int sap\_vkey\_codes }

Reponsibilities:

Mediates connection to SAP GUI scripting engine.

**biaProcessor.py**

+ DataFrame convert\_fbl5n\_data( str file\_path, str enc )

+ DataFrame convert\_dms\_data( str file\_path, str enc )

+ DataFrame assign\_country( DataFrame data, dict mapper )

+ DataFrame extract\_case\_ids( DataFrame data,

dict id\_to\_rx )

+ DataFrame compact\_data( DataFrame acc\_data,

DataFrame disp\_data )

+ DataFrame check\_consistency( DataFrame data )

- DataFrame \_generate\_params( DataFrame data )

+ DataFrame search\_matches( DataFrame data,

dict cntry\_rules )

+ list create\_closing\_input( DataFrame data )

- str \_generate\_status\_sales( str old\_val, int credit\_note )

- Series \_parse\_amounts( Series vals )

+ namedtuple Record = namedtuple( name, [ vals ] )

- \_logger: Logger = getLogger("master")

- str \_PRECREDIT\_NOTE\_RX = r"0?(50)\d{7}"

- str \_EMPTY\_STRING = ""

- int \_STATUS\_OPEN = 1

- int \_STATUS\_SOLVED = 2

- int \_STATUS\_CLOSED = 3

- int \_STATUS\_DEVALUATED = 4

- str \_RC\_UNUSED = \_EMPTY\_STRING

- str \_RC\_CREDIT\_NOTE\_ISSUED = "L06"

- str \_RC\_PAYMENT\_AGREEMENT = "L01"

- str \_RC\_DISPUTE\_UNJUSTIFIED = "L00"

- str \_RC\_CHARGE\_OFF = "L08"

- str \_RC\_BELOW\_THRESHOLD = "L14"

Reponsibilities:

Mediates connection to SAP GUI scripting engine.

**load application configuration**

**program start**

**load processing rules**

# Testing and error handling

The CS DMS Closing has been thoroughly unit-tested during the development phase using a python built-in “unittest“ framework. For each functionality, a separate test suite was created. Once each of the test suites passed, an integration test of the entire application was performed against a sample data.

If any fatal error occurs during runtime, leading to application crash, the program execution will be resumed on application restart from the last successfully processed country. Any exceptional state will be reflected into a return code specifying the nature of the error (see Table 2).

**Table 2:** List of return codes and their interpretation.

|  |  |
| --- | --- |
| Code | Description |
| 0 | Program finishes without error. |
| 1 | Unable to read logging configuration file, load file data or parse loaded data. |
| 2 | Unable to connect to the SAP GUI scripting engine. |
| 3 | No active country detected at application start. |
| 4 | Data export from FBL5N transaction fails. |
| 5 | FBL5N data preprocessing fails. |
| 6 | Data export from UDM\_DISPUTE transaction fails. |
| 7 | Critical error during dispute processing in UDM\_DISPUTE transaction occurs. |

# Revision

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 1.0 | 09.03.2022 | Dušan Paál | Initial version |