Document updater

Documentation

Contents

[1. Overview 2](#_Toc151624172)

[2. Installation 2](#_Toc151624173)

[3. Execution 2](#_Toc151624174)

[4. Directory and file structure 2](#_Toc151624175)

[5. Application design 3](#_Toc151624176)

[5.1 The “app.py” module 4](#_Toc151624177)

[5.2 The “controller.py” module 5](#_Toc151624178)

[5.3 The “mails.py” module 7](#_Toc151624179)

[5.4 The “excel.py” module 9](#_Toc151624180)

[5.5 The “processor” package 9](#_Toc151624181)

[5.6 The “app\_config.yaml” file 12](#_Toc151624182)

[5.7 The “log\_config.yaml” file 12](#_Toc151624183)

[5.8 The “rules.yaml” file 13](#_Toc151624184)

[6. Program flow 14](#_Toc151624185)

[7. User manual 16](#_Toc151624186)

[Revision 18](#_Toc151624187)

# 1. Overview

The “Document Updater” application automates appending case ID numbers to existing strings stored in the ‘Text’ field of line items (credit notes, invoices) posted on customer accounts. The application also automates closing of QM service notifications.

# 1. Requirements

The client part requires MS Excel 2016 installed on the user computer. The server part requires access to SAP P25 system and the appropriate user roles are mandatory for successful data processing in the SAP FB03, VA03, and QM02 transactions is needed. The server part requires python 3.7 or higher; older versions 3.x should work as well but have not been tested. I addition, third part libraries are to be installed.

# 2. Installation

Run the “install.bat” file contained in the “app” directory to install the application. Follow the instructions in the setup program to navigate through the installation process.

# 3. Execution

The server-bound part of the automation starts by running the “app.bat” file contained in the “app” directory. The batch file requires passing an email address of the sender to the %email\_id% parameter. The client part of the application starts by opening the “app.xlsm” file contained in the “client” directory.

# 4. Directory and file structure

The application is organized into the following directories and files:

|  |  |
| --- | --- |
| **Name** | **Description** |
| server | Root directory for the server part of the application. |
| server/engine | Contains the engine scripts of the server part. |
| server/engine/controller.py | Controls high-level operations of the application. |
| server/engine/mails.py | Fetches, creates, and sends user emails. |
| server/engine/report.py | Generates user report. |
| server/engine/utils.py | Utility procedures used in the application modules. |
| server/engine/sap.py | Manages connection to the SAP GUI Scripting engine. |
| server/engine/fb03.py | Performs operations in the FB03 SAP transaction. |
| server/engine/va03.py | Performs operations in the VA03 SAP transaction. |
| server/engine/qm02.py | Performs operations in the QM02 SAP transaction. |
| server/env | Contains a local python environment. |
| server/logs | Contains application runtime logs. |
| server/notification | Contains templates for user notifications. |
| server/notification/template\_error.html | Template for error-reporting notifications. |
| server/notification/template\_completed.html | Template for success-reporting notifications. |
| server/temp | Contains temporary files. |
| server/app\_config.yaml | Contains configurable application settings. |
| server/log\_config.yaml | Contains configurable logging settings. |
| server/app.py | Program entry point of the application. |
| server/app.bat | Batch file that runs the automation. |
| app/install.bat | Installs the server part of the application. |
| server/requirement s.txt | Contains a list of site-packages and their versions. |
| client | Root directory for the client part of the application. |
| client/app.xlsm | Main program of the client part. |
| client/LEDbot.ico | Icon of the main client program. |
| client/LEDbot.jpg | Icon of the “Upload” button in the client program. |
| doc | Root directory for project documentation. |
| doc/Documentation\_v1.0.docx | Administrator and user manual. |

# 5. Server component design

The automation was developed using Python 3.9. Older versions (3.7 - 3.8) should work as well but have not been tested. The individual Python modules are organized in a horizontal 3-tiered pattern, with each layer intended to be self-independent (Fig. 1).

The top layer consists of the “app.py” module, which is the program entry point. The responsibility of this layer is to initialize the application, drive the overall processing of the business logic by invoking the controller layer, and perform the final cleanup. The application starts by running the “app.bat” executable which invokes the execution of the module’s code by the local “root/server/engine/env/python/Scripts/python.exe” interpreter.

The middle layer is represented by the “controller.py” module, which contains processing logic that models the established business process. This layer also bridges the high-level operations performed in the top layer and the low-level operations performed in the bottom layer.

The bottom layer consists of the "mails.py", "report.py", “qm02.py”, “sap.py”, “fb03.py”, and “va03.py” modules. These modules perform all low-level operations, including retrieving, preparing, and processing of the user data in dedicated SAP transactions, generating the user report from the processing output, and sending the notification along with an Except report to the user.

The main application configuration, customer-specific data processing parameters, and logging parameters are stored in the “app\_config.yaml”, and “log\_config.yaml” files, from where they are loaded during application initialization.

**request**

**document**

**updating**

**send**

**report**

**to user**

**request**

**data**

**writing**

**fetch**

**input**

**from**

**email**

**request**

**operation**

**provide**

**response**

read text

convert text

Files

app\_config.yaml

log\_config.yaml

Bottom layer

Top layer

Middle layer

**return**

**report**

**path**

create

report

attach

report

**request**

**notification**

**closing**

return

processing output

**provide session**

**Fig. 1**: **The layered design of the application.**

**load configurations**

## **5.1 The “app.py” module**

The module contains the “main()” procedure that represents the entry point of the program:

def **main**( **args** : *dict* ) -> *int*:

**Description:**

Controls the overall execution of the program.

**Parameters:**

**args:** Arguments passed from the calling environment:

* “email\_id”: The string ID of the user message that has triggered the application.

**Returns:**

Program completion state:

* 0 : Program successfully completes.
* 1 : Program fails during logger configuration.
* 2 : Program fails during the initialization phase.
* 3 : Program fails during the processing phase.
* 4 : Program fails during the reporting phase.

## **5.2 The “controller.py” module**

The public interface of the controlling module consists of the following procedures:

def **configure\_logger**( **log\_dir** : *str*, **cfg\_path** : *str*, **\*header** : *str* ) -> *None:*

**Description:**

Configures application logging system.

**Parameters:**

**log\_dir:** Path to the directory to store the log file.

**cfg\_path:** Path to a yaml/yml file that contains application configuration parameters.

**header:** A sequence of lines to print into the log header.

**Returns:**

The procedure does not return an explicit value.

def **load\_app\_config**( **cfg\_path** : *str* ) -> *dict:*

**Description:**

Reads application configuration parameters from a file.

**Parameters:**

**cfg\_path:** Path to a yaml/yml file that contains application configuration parameters.

**Returns:**

Application configuration parameters.

def **load\_processing\_rules**( **account\_maps\_dir** : *str*, **file\_path**: *str* ) -> dict:

**Description:**

Loads customer-specified data processing parameters.

**Parameters:**

**accounts\_map\_dir:** Path to the directory where account maps are stored.

**file\_path:** Path to the file containing the processing rules.

**Returns:**

Data processing parameters.

def **connect\_to\_sap**( **system** : *str*) -> *CDispatch:*

**Description:**

Creates connection to the SAP GUI scripting engine.

**Parameters:**

**system:** The SAP system to use for connecting to the scripting engine.

**Returns:**

An SAP *GuiSession* object that represents active user session.

def **disconnect\_from\_sap**( **sess** : *CDispatch*) -> *None:*

**Description:**

Closes connection to the SAP GUI scripting engine.

**Parameters:**

**sess:** An SAP *GuiSession* object (wrapped in the *win32:CDispatch* class) that represents an active SAP GUI session.

**Returns:**

The procedure does not return an explicit value.

def **fetch\_user\_input**( **msg\_cfg** : *dict*, **email\_id** : *str*) -> *dict:*

**Description:**

Fetches the processing parameters and data provided by the user.

**Parameters:**

**msg\_cfg:** Application 'messages' configuration parameters.

**email\_id:** The string ID of the message.

**Returns:**

Names of the processing parameters and their values:

- "error\_message": (*str*) A detailed error message if an exception occurs.

- "email": (*str*) Email address of the sender.

- "attachment\_paths": (*list[str]*) List of paths to downloaded attachments.

def **assign\_credit\_note\_numbers**( **sess** : *CDispatch*, **in\_data** : *DataFrame* ) -> *DataFrame*:

**Description:**

Retrieves credit note numbers for the order numbers contained in data input using VA03 transaction.

**Parameters:**

**sess:** An SAP *GuiSession* object.

**data**: Original user data containing order numbers for which credit note numbers will be retrieved.

**Returns:**

Original user data with added 'Credit\_Note' field containing credit note numbers for each order where the credit note has already been created. If no credit note is found for a given order, then the default 'Credit\_Note' field value is left in the input data.

def **update\_accounting\_documents**( **sess** : *CDispatch*, **in\_data** : *DataFrame* ) -> *DataFrame*:

**Description:**

Updates document (credit note/invoice) text on case ID in FB03.

**Parameters:**

**sess:** An SAP *GuiSession* object.

**data**: Original user data containing numbers of documents to update.

**Returns:**

Original user data updated on 'Message' strings that inform the user of the document processing result.

def **close\_service\_notifications**( **sess** : *CDispatch*, **in\_data** : *DataFrame* ) -> *DataFrame*:

**Description:**

Closes service notifications in QM02.

**Parameters:**

**sess:** An SAP *GuiSession* object.

**data**: Original user data that contains numbers of notifications to close.

**Returns:**

Original user data updated on 'Message' strings that inform the user of the document processing result.

def **create\_report**( **temp\_dir** : *str*, **report\_cfg** : *dict*, **data** : *DataFrame* ) -> *DataFrame*:

**Description:**

Creates user report from the processing result.

**Parameters:**

**temp\_dir:** Path to the directory where temporary files are stored.

**data\_cfg**: Application 'data' configuration parameters.

**data**: The processing result from which report will be generated.

**Returns:**

Path to the report file.

def **send\_notification**(

**msg\_cfg** : *dict*, **user\_mail** : *str*, **template\_dir** : *str*,

**attachment** : Union[*dict*, *str*]= None, **error\_msg** : *str* = ""

)-> *None*:

**Description:**

Sends a notification with processing result to the user.

**Parameters:**

**msg\_cfg:** Application 'messages' configuration parameters.

**user\_mail:** Email address of the user who requested processing.

**template\_dir:** Path to the application directory that contains notification templates.

**attachment:** Attachment name and data, or a file path.

**error\_msg:** Error message that will be included in the user notification.

By default, no error message is included.

**Returns:**

The procedure does not return an explicit value.

def **delete\_temp\_files**( **temp\_dir** : *str*)-> *None*:

**Description:**

Removes all temporary files.

**Parameters:**

**temp\_dir:** Path to the directory where temporary files are stored.

**Returns:**

The procedure does not return an explicit value.

## **5.3 The “mails.py” module**

The module provides a simplified interface for managing emails for a specific account that exists on an Exchange Web Services (EWS) server. Most of the procedures depend on the *exchangelib* package, which must be installed before using the module.

def **get\_account**( **mailbox** : *str*, **name** : *str*, **x\_server** :*str* ) -> *Account:*

**Description:**

Models an MS Exchange server user account.

**Parameters:**

**mailbox:** Name of the shared mailbox.

**name:** Name of the user account.

**x\_server:** Name of the MS Exchange server.

**Returns:**

The user account object.

**Raises:**

*CredentialsNotFoundError*:

When the file with the account credentials parameters is not found at the path specified.

*CredentialsParameterMissingError*:

When a credential parameter is not found in the content of the file where credentials are stored.

def **create\_smtp\_message**(

**sender** : *str*, **recipient** : *Union[str, list]*, **subject** : *str*, **body** : *str*, **attachment** : *Union[FilePath, list, dict]* = None

)-> *SmtpMessage:*

**Description:**

Creates an SMTP-compatible message.

**Parameters:**

**sender:** Email address of the sender.

**recipient:** Email address or addresses of the recipient.

**subject:** Message subject.

**body:** Message body in HTML format.

**attachment:**

- *None* : The message will be created without any attachment.

- *FilePath* : Path to the file to attach.

- *list* [*FilePath*]: Paths to the files to attach.

- *dict* {*str* : *FilePath*} : file names and paths to attach.

Attachment type is inferred from the file type.

The file names will be used as attachment names.

An invalid file path raises `FileNotFoundError` exception.

- *dict* {*str* : *bytes*} : file names and `byte-like` objects to attach

Attachment type is inferred from the file name.

If the data type cannot be inferred, then a raw binary

object is attached. The file names will be used as attachment names.

**Returns:** The constructed message.

def **send\_smtp\_message**( **msg** : *SmtpMessage*, **host** : *str*, **port** : *int,* **timeout**: *int* = 30*,* **debug** *: int* = 0)-> *None:*

**Description:**

Send an SMTP message.

An *UndeliveredError* exception is raised if the message is not delivered to all recipients.

**Parameters:**

**msg:** Message to send.

**host:** Name of the SMTP host server used for message sending.

**port:** Number of the SMTP server port.

**timeout:** Number of seconds to wait for the message to be sent (default: 30).

Exceeding this limit will raise an *TimeoutError* exception.

**debug:** Whether debug messages for connection and for all messages

sent to and received from the server should be captured:

- 0: "off" (default)

- 1: "verbose"

- 2: "timestamped"

**Returns:**

The procedure does not return an explicit value.

def **get\_messages**( **acc** : *Account*, **email\_id** : *str* ) -> *list:*

**Description:**

Fetches messages with a specific message ID from an inbox.

**Parameters:**

**acc:** Account to access the inbox where the messages are stored.

**email\_id:** ID of the message to fetch (the "Message.message\_id" property).

**Returns:**

A list of *exchangelib:Message* objects that represent the retrieved messages.

If no messages with the specified ID are found, then an empty list is returned.

This may happen when the message ID is incorrect, or the message has been deleted.

def **get\_attachments**( **msg** : *Message*, **ext** : *str* = ".\*" ) -> *list*:

**Description:**

Fetches message attachments and their names.

**Parameters:**

**msg:** Message from which attachments are fetched.

**ext:** File extension, that filters the attachment file types to fetch.

By default, any file attachments are fetched. If an extension (e. g. ".pdf")

is used, then only attachments with that file type are fetched.

**Returns:**

A *list* of *dict* objects, each containing attachment parameters:

- "name" (*str*): Name of the attachment.

- "data" (*bytes*): Attachment binary data.

def **save\_attachments**( **msg** : *Message*, **dst** : *DirPath*, **ext** : *str* = ".\*" ) -> *list*:

**Description:**

Saves message attachments to a local folder.

**Parameters:**

**msg:** An *exchangelib:Message* object that represents the email with attachments to download.

**ext:** File extension to filter the attachments to be downloaded.

By default, all attached files are downloaded. If a file extension (e.g.: '.pdf') is used,

then only attachments of the specified file type are downloaded.

**Returns:**

A *list[FilePath]* of file paths to the stored attachments.

## **5.4 The “report.py” module**

The module creates Excel file from data extracted from payment advice documents.

def **generate\_excel\_report**( **data** : *DataFrame*, **file** : *FilePath*, **sheet\_name** : *str* ) -> *None:*

**Description:**

Creates a user Excel report from the processing result.

**Parameters:**

**data:** Data to print to the report sheet.

**file:** Path to the .xlsx report file to create.

If the destination folder doesn't exist, then an `FolderNotFoundError` exception is raised.

**sheet\_name**: Name of the sheet with printed data.

**Returns:**

The procedure does not return an explicit value.

## **5.5 The “fb03.py” module**

The module automates appending case ID numbers to an existing 'Text' string of accounting documents and removing file attachments from accounting documents in the FB03 SAP transaction. The FB03 must be started by calling the `start()` procedure. Attempt to use an exclusive procedure when FB03 has not been started results in the *UninitializedModuleError* exception. After using the module, the transaction should be closed, and the resources released by calling the `close()` procedure.

def **start**( **sess**: *CDispatch*) -> *None:*

**Description:**

Starts the FB03 transaction.

If the FB03 has already been started, then the running transaction will be restarted.

**Parameters:**

**sess:** An SAP `GuiSession` object.

**Returns:**

The procedure does not return an explicit value.

def **close**() -> *None:*

**Description:**

Closes a running FB03 transaction.

Attempt to close the transaction that has not been started by the `start()` procedure is ignored.

**Parameters:**

The procedure does not take any parameters.

**Returns:**

The procedure does not return an explicit value.

def **append\_case\_id**( **document** : *int*, **fiscal\_year** : *int*, **company\_code** : *str***, case\_id** : *int*) -> *None:*

**Description:**

Appends case ID to an existing document text.

The case is appended to the original text so that in a specific format: "$original\_text$ D $case\_id$".

If the document text already contains the case ID, then a *CaseIdContainedWarning* exception is raised.

When the connection to SAP is lost due to an error, then a *SapConnectionLostError* exception is raised.

A *DocumentProcessingError* exception is raised, when:

- attempting to modify a cleared document that is no longer editable

- attempting to edit a document result in an SAP error

- attempting to save changes to a document generates an SAP error

- text length limit is exceeded by inserting a case ID

- a dialog window appears, for which no appropriate handler exists

**Example:**

>>> original text: "RET711884319"

>>> append\_case\_id(544411698, 2024, "0010", 400081469)

>>> updated text: "RET711884319 D 400081469"

**Parameters:**

**document:** Number of the accounting document to modify.

**fiscal\_year:** Fiscal year in which the document was created.

**company\_code**: Company code for which the document was created.

**case\_id**: Identification number under which the case is stored in DMS.

**Returns:**

The procedure does not return an explicit value.

def **remove\_attachments**( **document** : *int*, **fiscal\_year** : *int*, **company\_code** : *str*) -> *None:*

**Description:**

Removes all files attached to an accounting document.

If no document is found using the specified searching criteria,

then a *DocumentNotFoundError* exception is raised.

When the connection to SAP is lost due to an error,

then a *SapConnectionLostError* exception is raised.

**Parameters:**

**document:** Number of the accounting document.

**fiscal\_year:** Fiscal year in which the document was created.

**company\_code**: Company code for which the document was created.

**case\_id**: Identification number under which the case is stored in DMS.

**Returns:**

The procedure does not return an explicit value.

## **5.6 The “qm02.py” module**

The module automates completion of a service notification in the QM02 SAP transaction. The QM02 must be started by calling the `start()` procedure. Attempt to use an exclusive procedure when QM02 has not been started results in the *UninitializedModuleError* exception. After using the module, the transaction should be closed, and the resources released by calling the `close()` procedure.

def **start**( **sess**: *CDispatch*) -> *None:*

**Description:**

Starts the QM02 transaction.

If the QM02 has already been started, then the running transaction will be restarted.

**Parameters:**

**sess:** An SAP `GuiSession` object.

**Returns:**

The procedure does not return an explicit value.

def **close**() -> *None:*

**Description:**

Closes a running QM02 transaction.

Attempt to close the transaction that has not been started by the `start()` procedure is ignored.

**Parameters:**

The procedure does not take any parameters.

**Returns:**

The procedure does not return an explicit value.

def **complete\_notification**( **notif\_id** : *int*) -> *None:*

**Description:**

Completes an opened service notification.

If the notification is already completed, then a *NotificationCompletionWarning* warning is raised.

If the attempt to complete a notification fails, then a *NotificationCompletionError* exception is raised.

When the connection to SAP is lost due to an error, then a *SapConnectionLostError* exception is raised.

**Parameters:**

**notif\_id**: Identification number of a service notification.

**Returns:**

The procedure does not return an explicit value.

## **5.7 The “va03.py” module**

The module automates the retrieval of credit note numbers for credit note order requests using the SAP transaction VA03. The VA03 must be started by calling the `start()` procedure. Attempt to use an exclusive procedure when VA03 has not been started results in the *UninitializedModuleError* exception. After using the module, the transaction should be closed, and the resources released by calling the `close()` procedure.

def **start**( **sess**: *CDispatch*) -> *None:*

**Description:**

Starts the VA03 transaction.

If the VA03 has already been started, then the running transaction will be restarted.

**Parameters:**

**sess:** An SAP `GuiSession` object.

**Returns:**

The procedure does not return an explicit value.

def **close**() -> *None:*

**Description:**

Closes a running VA03 transaction.

Attempt to close the transaction that has not been started by the `start()` procedure is ignored.

**Parameters:**

The procedure does not take any parameters.

**Returns:**

The procedure does not return an explicit value.

def **get\_creditnote\_number**( **order** : *int* ) -> *None:*

**Description:**

Retrieves credit note number related to an order.

If a credit note does not exist yet in the system, then a *CreditNoteNotFoundWarning* warning is raised.

If the order is blocked, then a *DocumentProcessingWarning* warning is raised.

If no document is found for a particular order number, or an unhandled error occurs,

then a *DocumentProcessingError* exception is raised.

When the connection to SAP is lost due to an error, then a *SapConnectionLostError* exception is raised.

**Parameters:**

**order**: The number of the credited order.

**Returns:**

The number of the credit note created for an order.

## **5.8 The “sap.py” module**

The module provides an interface for managing connection to the SAP GUI Scripting Engine.

def **connect**( **system** : *str*, **exe** : *FilePath*="") -> *CDispatch:*

**Description:**

Connects to the SAP GUI Scripting Engine.

A *SapConnectionError* exception is raised when logging into the SAP GUI Scripting Engine fails.

**Parameters:**

**system**: SAP system with the GUI Scripting Engine to connect:

- "P25": Productive SSO.

- "Q25": Quality Assurance SSO.

**exe**: Path to the local SAP GUI executable.

If the path is not specified, then the default local SAP installation directory is searched for the executable file.

If the executable is not found, then a *FileNotFoundError* exception is raised.

**Returns:**

An SAP *GuiSession* context object representing an active session for running transactions.

def **disconnect**( **sess** : *CDispatch* ) -> *None:*

**Description:**

Disconnects from the SAP GUI Scripting Engine.

**Parameters:**

**sess**: An SAP *GuiSession* object.

**Returns:**

The procedure does not return an explicit value.

## **5.9 The “utils.py” module**

The module contains utility functions for common tasks across applications.

def **print\_section\_break**(

**log** : *Logger*, **n\_chars** : *int* = 20, **tag** : str = "", **char** : *str* = "-", **end** : *str* = "", **sides** : *str* = "both"

) -> *None*:

**Description:**

Connects to the SAP GUI Scripting Engine.

A *SapConnectionError* exception is raised when logging into the SAP GUI Scripting Engine fails.

**Parameters:**

**log**: Logger used to print the section break.

**n\_chars**: Number of characters used for the indentation from both sides (default: 20).

**tag**: A text to insert before the counter of the section line (default: "").

**char**: The character used to create the indentation sequence (default: "-").

**end**: Ending character of the line (default: "").

**sides**: Sides to indent:

- "both": Both sides are indented (default behavior).

- "left": Only the left side is indented.

- "right": Only the right side is only indented.

**Returns:**

The procedure does not return an explicit value.

## **5.6 The “app\_config.yaml” file**

This file contains the main application configuration:

**sap**:

SAP configuration parameters.

**system**: *str*

Code of the SAP GUI system to connect (default: ‘P25’).

**report**:

Parameters related to Excel report generation.

**file\_name**: *str,* default: ‘output’

Name of the report file.

**datasheet\_name**: *str,* default: ‘Data‘

Name of the sheet where processing output is written.

**messages**:

Parameters related to message processing.

**requests**:

Parameters related to processing incoming user requests.

**account**: *str*, default: ‘lbs.robot@ledvance.com’

Name of the account used to log into the mailbox with the user request emails.

**mailbox**: *str*, default: ‘lbs.robot@ledvance.com’

Name of the mailbox with the user request emails.

**server**: *str*, default: ‘outlook.office365.com’

Name of the mailbox with the user request emails.

**notifications**:

Parameters related to sending notifications to users.

**send**: *bool*, default: true

Whether notifications with processing result are sent to users.

**sender**: *str,* default: ‘notifications@ledvance.com‘

Email address of the notification sender.

**subject**: *str*, default: ‘Notification of payment advice conversion’

Subject of the user notification.

**host**: *str*, default: ‘intrelay.ledvance.com’

Name of the server hosting the SMTP service.

**port**: *int*, default: 25

Number of the port used to connect to the host.

## **5.7 The “log\_config.yaml” file**

This file contains configuration for the application logging system. A detailed description of the standard parameters and their use is available in the official python [documentation](https://docs.python.org/3.9/library/logging.html). The configuration includes a custom parameter “retain\_logs\_days” that specifies the number of days that old log files will be retained.

# 6. Program flow

The program starts by calling the *main()* procedure contained in the *app.py* module. First, the application is initialized by configuring the logging system, loading the application configuration, and connecting to the SAP GUI Scripting Engine. If no errors occur, the program then proceeds to the processing phase.

In the user input fetching phase, the user message that has triggered the application is retrieved. The message contains attached PDF files, which are then extracted from the message.

In the processing phase, the open items are loaded into FBL5N or FBL3N using the specified company code and accounts. The table of loaded items is formatted by applying a layout, that must include the *Text* field. Once the data is loaded, the items are filtered on the *Text* field by entering the original text values contained in the user data. If no item is found using the filtered values, then the user receives a warning notification that the provided text values to change were not found in the customer accounts. If an item is found, then the existing text of the item is replaced with the corresponding *New text* value. If the account supports *Assignment* field and the user data contains the *New assignment* value, then the existing field will be replaced with that value.

Once all items are processed, the reporting phase begins. A completion notification is sent back to the user with an attached Excel file(s) containing the original data and the item processing status written in the *Message* field.

Finally, a cleanup phase is performed to remove all application temporary files and close the connection to the SAP GUI Scripting Engine.

**Fig. 2:** **Program flow from the perspective of the top layer.** The execution starts by calling the “main()” procedure. The program then continues with initialization, data gathering, item processing, and finishes by reporting of the outcome to the user. If no internal error occurs, the program finishes with return code 0. If any internal error occurs, then the program exits with a non-zero return code.

**Cleanup phase**

**Processing phase**

**Initialization phase**

**Gathering phase**

**Reporting phase**

**controller.send\_notification()**

**controller.disconnect\_from\_sap()**

**controller.delete\_temp\_files()**

**controller.fetch\_user\_input()**

**User input error?**

**controller.configure\_logger()**

**controller.load\_app\_config()**

**controller.connect\_to\_sap()**

**Initialization**

**error?**

**return 1**

**controller.assign\_credit\_note\_numbers()**

**controller.update\_accounting\_documents()**

**controller.close\_service\_notifications()**

**Processing error?**

**controller.send\_notification()**

**controller.delete\_temp\_files()**

**controller.delete\_temp\_files()**

**controller.send\_notification()**

**Reporting error?**

**return 0**

**return 4**

**main()**

**Input fetching**

**error?**

**controller.delete\_temp\_files()**

**controller.disconnect\_from\_sap()**

**Yes**

**No**

**No**

**Yes**

**return 0**

**return 0**

**Yes**

**No**

**return 3**

**Yes**

**No**

**Yes**

**No**

# 7. User manual

# 

Copy the content of the „client.zip“ file to a folder of your choice. **WARNING**: The folder structure must be retained, otherwise it may result in an unexpected program behavior. Double-click the file „app.xlsm“ file to open the application. Make sure macros are enabled on your computer. To do that, navigate to: “File” -> “Options” -> “Trust Center” -> “Trust Center Settings …” and tick all the options as seen on the picture right.

The user is mandated to fill in all the required fields on the „Data“ worksheet with valid entries. **Warning**: Entering an invalid value will result in skipping the entire row during processing! A detailed description of the columns:

A screenshot of a computer

Description automatically generated**Commapy Code**:

A 4-digit code of the country related to the document.

**Document Number**:

Number of the document as stored in SAP.

**Document year**:

Fiscal year, in which the document was created.

**Case ID**:

ID number of the corresponding case in DMS.

**Notification**:

ID number of service notification

Fill in the columns with the desired data as follows:

* If you wish to add case ID number to the text of an accounting document, provide the company code, the document number and the docment year. The document number can be an invoice number, credit note number or a credit note request number (e.g. 501827545).
* If you wish to close a service notification, provide the number of the notification.

A screenshot of a computer

Description automatically generatedThe above actions can also be used in combination, if desired. An example of correctly filled data sheet with entries to process and actions taken are illustrated below:

**Row 2:** Text of the credit note 557810191 will be updated on case ID: 400040573 in format: „original\_text D 400040573“ and notification 1001297987 wil be completed in QM02.

**Row 3:** Text of the credit note 557810120 will be updated on case ID: 400039119 in format: „original\_text D 400039119“.

**Row 4**: Text of the invoice 414420121 will be updated on case ID: 400039119 in format: „original\_text D 10219672“.

**Row 5:** Notification 1001458984 will be completed.

**Row 6**: Credit note request number will be used to detect the corresponding credit note number, the text of which will then be updated on case ID: 400039128 in format: „original\_text D 400039128“.

Once the table is filled, the data needs to be sent to the server by navigating to te tab „Add-ins“ and clicking the „Upload Data“ button.

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer error

Description automatically generatedThe data will be checked, and if no inconsistencies are found, the user is prompted to confirm data sending to the server. Click „Yes“ to continue, „No“ to abort data sending. The processing will start as soon as the server receives the user mail with the attached data.

# Email notification

Once the data processing has finished, the server sends a notification of completion to the user with attached „output.xlsx“ report detailing the processing output. The column „Message“ contains processing status for each document. If you’ve used **credit note request numbers** as document numbers, the corresponding **credit note numbers** will be displayed in column „Credit Note“.

A screenshot of a computer

Description automatically generated A table with numbers and letters

Description automatically generated

# Testing and error handling

A series of unit and integration tests was performed in a close cooperation with end users to ensure program correctness and stability. All program runtime events are recorded in a log file which is part of the server application. Any errors or unexpected behavior will be handled by the developer by reviewing the log file and/or a user error report. Users are therefore, strongly advised to report any bugs or inconsistencies in this documentation by sending an issue request to: support.bia@ledvance.com.

# Revision

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 1.0 | 20.11.2023 | Dusan Paal | Initial version |