AR Account Items Updater

Documentation

Contents

1. Overview

The “AR Accounting Items Updater” application updates the 'Text' and ‘Assignment’ fields of items posted on customer or G/L accounts:

1. The accountant enters data into the dedicated worksheet in the client part of the application.
2. Then, the accountant uploads the filled in data form to the processing server.
3. The server part of the application then updates the items in FBL3N or FBL5N, respectively.
4. An Excel report is generated and sent back to the accountant.

2. Installation

The server part of the application is contained in the “app” directory, while the client part of the application is contained in the “client” directory. Both parts are installed separately by running the “install.bat” files stored in their root directories.

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/fbl3n.py	Automates text updating in the FBL3N SAP transaction.
server/engine/fbl5n.py	Automates text updating in the FBL5N SAP transaction.
server/engine/mails.py	Fetches, creates, and sends user emails.
server/engine/report.py	Creates user excel reports.
server/engine/sap.py	Mediates connection to the SAP GUI Scripting Engine.
server/env	Contains a local python environment.
server/logs	Contains application runtime logs.
server/notifications	Contains templates for user notifications.
server/notifications/template_error.html	Template for error-reporting notifications.
server/notifications/template_general.html	Template for success-reporting notifications.

server/temp	Contains temporary files.
server/app.py	Program entry point of the application.
server/app_config.yaml	Contains configurable application settings.
server/log_config.yaml	Contains configurable logging settings.
server/requirements.txt	Contains a list of site-packages and their versions.
server/install.bat	Installs the server part of the application.
client	Root directory for the client part of the application.
client/app.xlsm	Main program of the client part.
client/install.bat	Installs the client part of the application.
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_v1.1.docx	Administrator and user manual.

5. Server component design

The server-bound part of 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 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 modules "sap.py", "fbl3n.py", "fbl5n.py", "mails.py" and "report.py". 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 and logging parameters are stored in “app_config.yaml” and “log_config.yaml” files, from where they are loaded during application initialization.

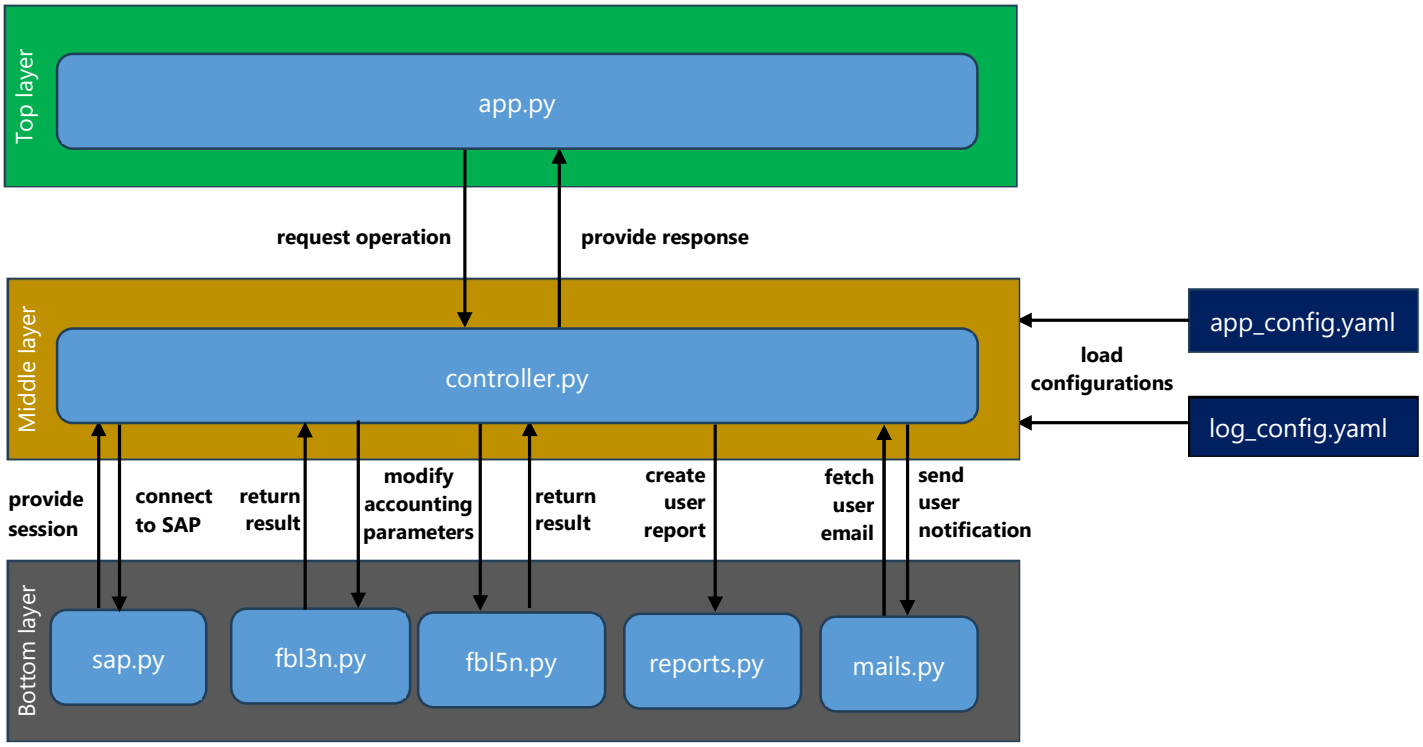


Fig. 1: **The layered design of the server part of the application.**

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 **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 **get_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": A detailed error message if an exception occurs.
- "email": Email address of the sender.

- "company_code": Company code provided by the sender.
- "data": An *pandas.DataFrame* object containing converted attachment data.
- "account_type": Type of the accounts:
- "customer": The accounts contained in the data are customer accounts.
- "general_ledger": The accounts contained in the data are G/L accounts.
- "attachment":
    - "name": attachments[0]["name"],
    - "content": converted

def **modify_accounting_parameters** (
sess : *CDispatch*, **data** : *DataFrame*, **acc_type** : *str*, **cocd** : *str*, **data_cfg** : *dict*
) -> *dict*:

Description:

Modifies accounting item parameters in FBL3N/FBL5N according to the data supplied by the user.
As of the current version, only the 'Text' and 'Assignment' fields can be modified.

Parameters:

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

data: Message attachment data.

acc_type: Type of the processed accounts.

cocd: Company code of the processed accounts.

data_cfg: Application 'data' configuration params.

Returns:

The processing result, with the following keys and values:
- “data”: (*pandas.DataFrame*) The original user data and the processing status stored in its 'message' field.
- “error_message”: (*str, None*) An error message if an exception occurs, otherwise *None*.

def **create_report**(**temp_dir** : *str*, **data_cfg** : *dict*, **data** : *DataFrame*) -> *str*:

Description:

Creates a user report from the processing outcome.

Parameters:

temp_dir: Path to the directory where temporary files are stored.

data_cfg: Application 'data' configuration parameters.

data: The processing outcome from which the 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.

template_dir: Path to the application directory that contains notification templates.

user_mail: Email address of the user who requested processing.

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 "sap.py" module

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

DEFAULT_EXE_PATH : *str* = "C:\Program Files (x86)\SAP\FrontEnd\SAPgui\saplogon.exe"
The default path where the local SAP GUI is expected to be installed.

system_code : *str* = ""
Variable that stores the code of the system that was logged in when the *connect()* procedure was called.

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 that represents an active session where transactions run.

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

Description:
Disconnects from the SAP GUI Scripting Engine.

Parameters:
sess: An SAP *GuiSession* object.

Returns:
The procedure does not return an explicit value.

5.4 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:

Sends 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.

5.5 The “flb5n.py” module

The application controller uses only the ‘change_document_parameters()’ procedure to modify the 'Text' and 'Assignment' fields of the line items. The FBL5N must be started by calling the `start()` procedure. Attempt to use an exclusive procedure when FBL5N 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 FBL5N transaction.

If the FBL5N has already been started, then the running transaction will be restarted.

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

Description:

Closes a running FBL5N transaction.

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

Parameters:

The procedure takes no parameters.

Returns:

The procedure does not return an explicit value.

def **change_document_parameters**(
accounts : *list*, **company_code** : *str*, **parameters** : *dict*, **status** : *str* = "open", **layout** : *str* = ""
) -> *dict*:

Description:

Replaces the 'Text' and 'Assignment' parameters of accounting items.

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

If loading of accounting items completely fails, then an *ItemsLoadingError* is raised.

If no items are found using the specified company code and customer accounts, then a *NoItemsFoundWarning* warning is raised.

If no items are found when filtering the table on the 'Text' field using the specified text values, then an *NoItemsFoundError* exception is raised.

Parameters:

accounts: Numbers of customer accounts containing the line items to change.
company_code: The company code for which the accounting data will be changed.
parameters: Original item text values mapped to their new 'Text' and 'Assignment' values stored as `str`.
The data is structured as follows:

```
{
    "old_text_value_1": {
        "new_text": "value"
        "new_assignment": "value"
    },
    "old_text_value_2": {
        "new_text": "value"
        "new_assignment": "value"
    },
    ...
    "old_text_value_n": {
```

```
                    "new_text": "value"
                    "new_assignment": "value"
                }
            }
```

status: Item status to consider for selection:

- "open": Open items will be exported (default).
- "cleared": Cleared items will be exported.
- "all": All items will be exported.

layout: The name of the layout that defines the format of the loaded data.

By default, no specific layout name is used, and a the predefined FLB5N layout is used.

Returns:

The processing results with the following structure:

```
{
    "old_text_value_1": {
            "new_text": "value"
            "new_assignment": "value"
            "message": "value"
    },
    "old_text_value_2": {
            "new_text": "value"
            "new_assignment": "value"
            message": "value"
    },
    ...
    "old_text_value_n": {
            "new_text": "value"
            "new_assignment": "value"
            message": "value"
    }
}
```

5.6 The "flb3n.py" module

The application controller uses only the 'change_document_parameters()' procedure to modify the 'Text' and 'Assignment' fields of the line items. The FBL3N must be started by calling the `start()` procedure. Attempt to use an exclusive procedure when FBL3N 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 FBL3N transaction.

If the FBL3N has already been started, then the running transaction will be restarted.

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

Description:

Closes a running FBL3N transaction.

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

Parameters:

The procedure takes no parameters.

Returns:

The procedure does not return an explicit value.

def **change_document_parameters**(
accounts : *list*, **company_code** : *str*, **parameters** : *dict,* **status** : *str* = "open", **layout** : *str* = ""
) -> *dict*:

Description:

Replaces the 'Text' and 'Assignment' parameters of accounting items.

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

If loading of accounting items completely fails, then an *ItemsLoadingError* is raised.

If no items are found using the specified company code and GL accounts, then a *NoItemsFoundWarning* warning is raised.

If no items are found when filtering the table on the 'Text' field using the specified text values, then an *NoItemsFoundError* exception is raised.

Parameters:

accounts: Numbers of G/L accounts containing the line items to change.

company_code: The company code for which the accounting data will be changed.

parameters: Original item text values mapped to their new 'Text' and 'Assignment' values stored as *str*.

The data is structured as follows: {

```
"old_text_value_1": {
        "new_text": "value"
        "new_assignment": "value"
},
"old_text_value_2": {
        "new_text": "value"
        "new_assignment": "value"
},
...
"old_text_value_n": {
        "new_text": "value"
        "new_assignment": "value"
}
```

}

status: Item status to consider for selection:

- "open": Open items will be exported (default).
- "cleared": Cleared items will be exported.
- "all": All items will be exported.

layout: The name of the layout that defines the format of the loaded data.

By default, no specific layout name is used, and a the predefined FLB3N layout is used.

Returns:

The processing results with the following structure: {

```
"old_text_value_1": {
        "new_text": "value"
        "new_assignment": "value"
```

```
                "message": "value"
        },
        "old_text_value_2": {
                "new_text": "value"
                "new_assignment": "value"
                message": "value"
        },
        ...
        "old_text_value_n": {
                "new_text": "value"
                "new_assignment": "value"
                message": "value"
        }
}
```

5.7 The "report.py" module

The module creates user reports from the processing output.

def **generate_excel_report**(**file** : *FilePath*, **data** : *DataFrame*, **sht_name** : *str*) -> *None:*

Description:

Creates an excel report from the processing outcome.

Parameters:

file: Path to the report file.

data: The processing outcome containing records to write.

sht_name: Name of the report sheet.

Returns:

The procedure does not return an explicit value.

5.8 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 (e. g.: 'P25').

data:

Parameters related to data processing.

fbl3n_layout: *str, null*

Name of the layout used for column formatting in FBL3N data (e. g.: 'TEXT_UPDATER').
Any name can be used, but the corresponding layout must contain the "Text" field.
If *null* is used, then the "Layout" field on the FBL3N initial mask is unfilled, and the default SAP layout is applied.

fbl5n_layout: *str, null*

Name of the layout used for column formatting in FBL5N data (e. g.: 'TEXT_UPDATER').
Any name can be used, but the corresponding layout must contain the "Text" field.
If *null* is used, then the "Layout" field on FBL5N initial mask is unfilled, and the default SAP layout is applied.

report_name: *str*

Name of the user report file. Any non-empty string is valid (e. g.: 'report').

sheet_name: *str*
Name of the data sheet of the user report. Any non-empty string is valid (e. g.: 'Data').

messages:
Parameters related to message processing.

requests:
Parameters related to processing incoming user requests.

account: *str*
Name of the account used to log into the mailbox with the user request emails (e. g.: 'lbs.robot@ledvance.com').
mailbox:
Name of the mailbox with the user request emails (e. g.: 'lbs.robot@ledvance.com').
server: *str*
Name of the mailbox with the user request emails (e. g.: 'outlook.office365.com').

notifications:
Parameters related to sending notifications to users.

send: *bool*
If *true*, then notifications are sent to users.
sender: *str*
Email address of the notification sender (e. g.: 'notifications@ledvance.com').
subject: *str*
Subject of the user notification (e. g.: 'AR Account Item Text Updater').
host: *str*
Name of the server hosting the SMTP service (e. g.: 'intrelay.ledvance.com').
port: *int*
Number of the port used to connect to the host (e. g.: 25).

5.9 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. 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 the user input such as the company code and the attached data, which is then extracted from the message. The input is validated, and the type of accounts is identified - either customer or G/L accounts. Mixing of account types in the same data file is not allowed. The attached data represent an Excel file with the following fields (columns):

- *Account*: Number of the account where the item to change has been posted.
- *Old text*: Current *Text* value of the posted item.
- *New text*: Value to replace the current *Text* value of the posted item.
- *New Assignment*: Value to replace the *Assignment* value of the posted item.

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 report 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.

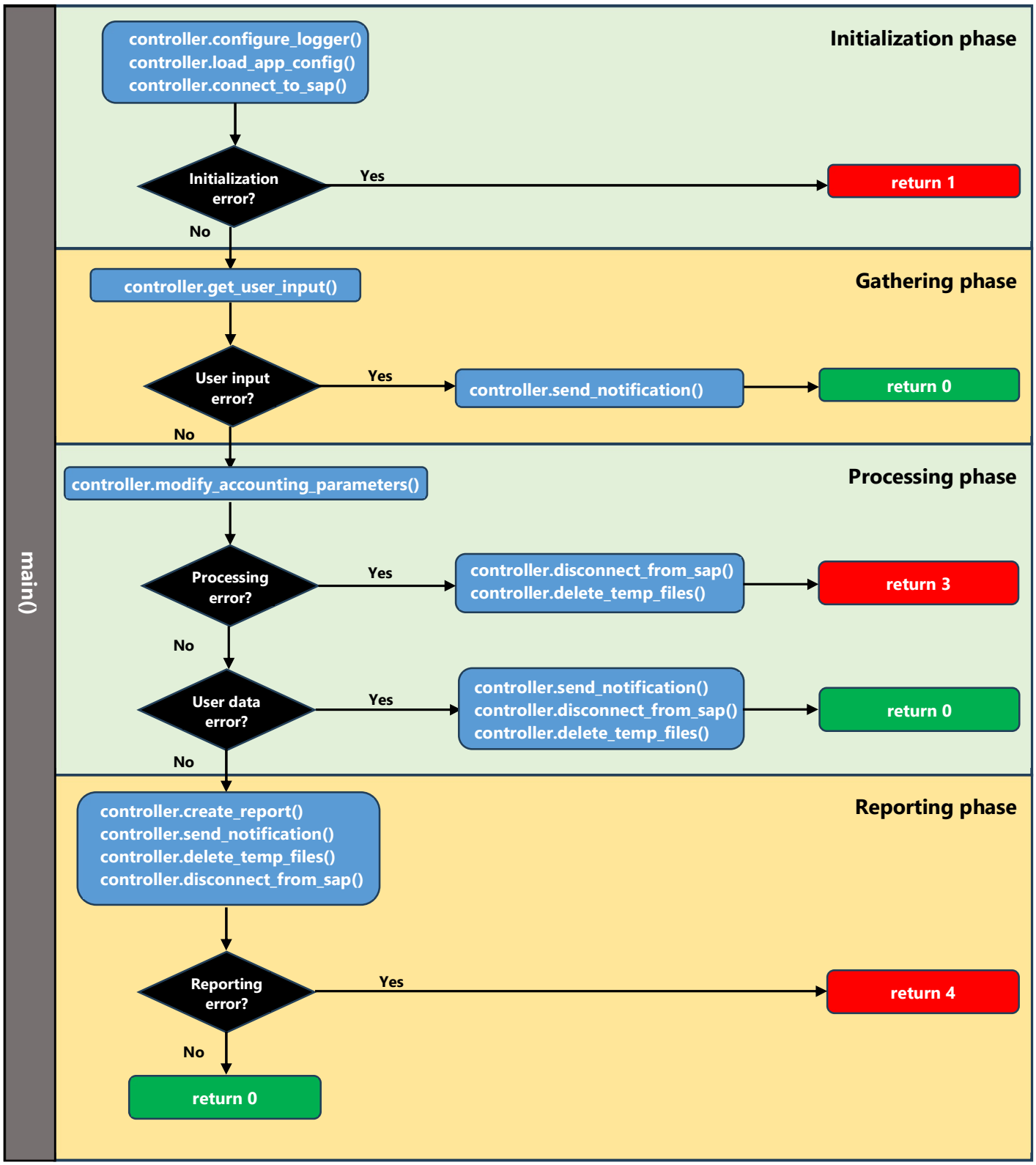


Fig. 2: Program flow from the perspective of the top layer. The execution starts by calling the "main()" procedure. The program then continues with initialization, data gathering, item processing, and finishes by reporting of the outcome to the user. If no internal error occurs, the program finishes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

7. The client component design

The client-bound part of the automation represents an Excel-based VBA application, that contains the data input template, and the procedures for data upload to the processing server.

7.1 The "app.xlsm" file

Contains the data input template as an Excel table with the following columns:

Account: Either G/L or customer account numbers. Combining both types of accounts is not permitted.
Old text: The current 'Text' values of open line items.
New text: The new 'Text' values to replace the existing 'Text' values of open line items.
New Assignment: The new 'Assignment' values to replace the existing 'Assignment' values of open line items.

The file hosts VBA modules that provide procedures for data upload to the processing server:

7.1.1. The "MD_Controller" module

Sub **Send**()

Description:

Sends the filled-in worksheet and the company code to the processing server.

7.1.1. The "MD_Messenger" module

Sub MD_Messenger.**SendOutlookMessage**(
ByVal **Subject** As *String*, ByVal **Body** As *String*, ByRef **Recipients** As *Collection*,
Optional ByRef **CcRecipients** As *Collection*, Optional ByRef **BccRecipients** As *Collection*,
Optional ByRef **Attachments** As *Collection*
)

Description:

Sends a message from the active Outlook account.

Parameters:

Subject: Subject of the email in HTML format.
Body: Body of the email.
Recipients: Email address(es) of the recipient(s)
CcRecipients: Email carbon copy address(es) of the recipient(s)
BccRecipients: Email blind carbon copy address(es) of the recipient(s)
Attachments: Path(s) to the file(s) to attach.

8. User manual

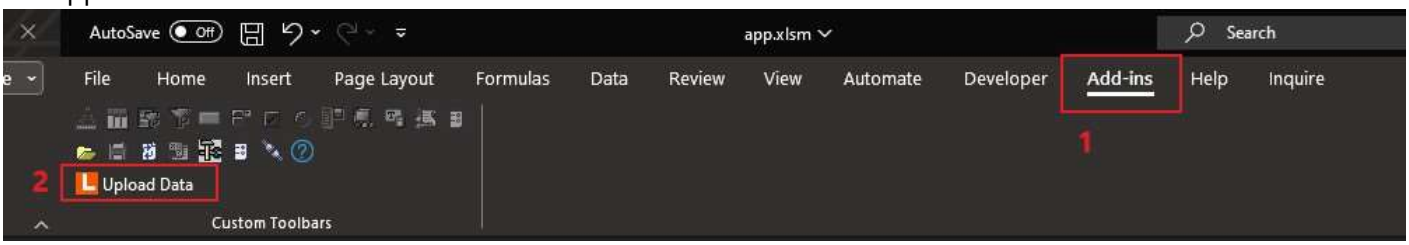
Unzip the *client.zip* archive and install the client application by running the *install.bat* file contained in the root directory. Follow the steps in the installation application. Once the installation is done, a desktop shortcut *Account Updater* is created.

Double-click the shortcut to open the application. An Excel file with pre-defined columns opens. A row represents one open line item (example below). Here, you can enter the input data into the columns as follows:

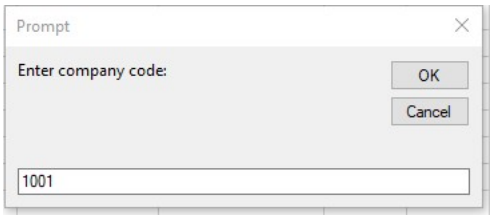
- **Account**: Enter either G/L or customer accounts that contain the open line items to update.
  Warning: Combining both types of accounts in one file is not allowed.
- **Old text**: Enter the current 'Text' values of open line items to change.
- **New text**: Enter new values to replace the existing item 'Text' values of open line items.
- **New Assignment**: Enter new values to replace the existing 'Assignment' values of open line items.

Account	Old Text	New Text	New Assignment
66791580	RE0000634417 D ??? (4928908)	RE0000634417 D 10326857 (4928908)	10326857
66791580	RE0000489944 D ??? (4928847)	RE0000489944 D 10326848 (4928847)	10326848
66791580	RE0000516723 D ??? (4928675)	RE0000516723 D 10326853 (4928675)	10326853
66791580	RE0000196888 D ??? (4928988)	RE0000196888 D 10326852 (4928988)	10326852
66791580	RE0000378116 D ??? (4928910)	RE0000378116 D 10326866 (4928910)	10326866
66791580	RE0000501631 D ??? (4928802)	RE0000501631 D 10326862 (4928802)	10326862

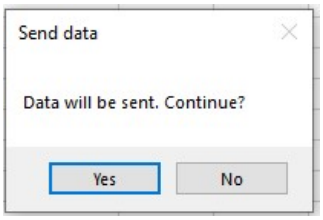
Upon entering the data into the table, navigate to the "Add-ins" tab (1) and click the "Upload Data" button (2) to send the data to the server for processing. If the button is missing, then close all open Excel files and restart the application.



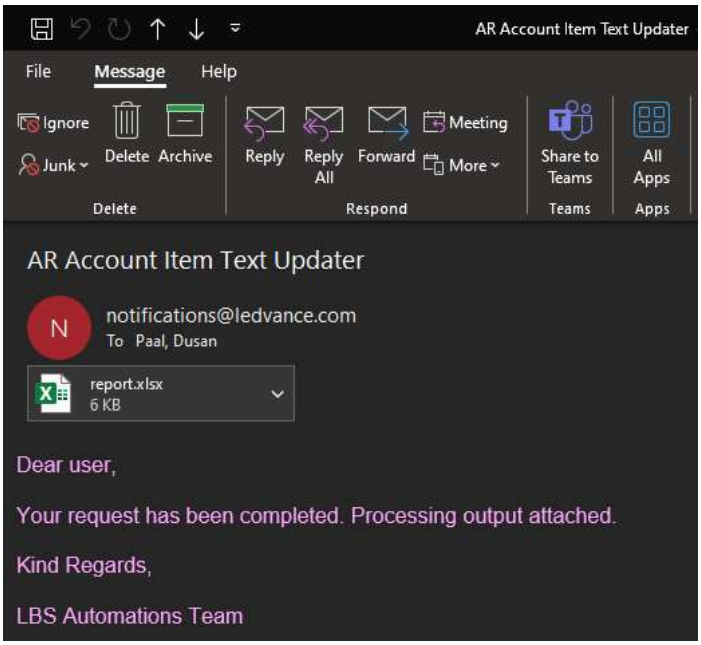
Now you will be prompted to provide the company code under which the entered items were posted. Enter the desired 4-digit value and press the "OK" button.



In the final prompt, confirm sending data to server by pressing the “Yes” button if you wish to proceed, or press the “No” button if you wish to cancel the operation.



When the server finishes processing your request, you will receive an email notification of completion with an attached Excel report with the original input data and the processing result recorded separately for each item in the "Message" column.



Account	Old Text	New Text	New Assignment	Message
66010030	DE5352200290 D ??? (1052521)	DE5352200290 D 10326833 (1052521)	10326833	Text updated. Assignment updated.
66010030	DE3112300713 D ??? (4928783)	DE3112300713 D 10325172 (4928783)	10325172	Text updated. Assignment updated.
66010030	DE1942300604 D ??? (4928740)	DE1942300604 D 10324553 (4928740)	10324553	Text updated. Assignment updated.

Revision

Version	Date	Author	Description
1.0	20.11.2023	Dusan Paal	Initial version