AR Account Items Updater

Documentation

Contents

1. Workflow Overview

The process is managed by the Accounts Receivables department and involves updating the “Text” and “Assignment” fields for items posted on customer or G/L accounts in SAP transactions FBL3N (G/L account line items) or FBL5N (customer line items):

1. **Loading of Items:**
   The accountant retrieves postings in SAP using the FBL3N (customer items) or FBL5N (general ledger items) transactions.
2. **Data Filtering:**
   After the data is loaded, the accountant filters the table items based on the "Text" or "Assignment" fields to identify the values that need to be changed.
3. **Data Updating:**
   The accountant manually updates each item by selecting it and overwriting the original values with the new ones.

2. Application Overview

The "AR Accounting Items Updater" application is designed to streamline the process of updating the 'Text' and 'Assignment' fields of items as described in the “Workflow Overview” section. Below is an overview of how the application works:

1. **Data Entry by Accountant:**
   - The accountant accesses the "AR Accounting Items Updater" client application.
   - The accountant fills in a template table with the necessary data. This data includes the specific items that need updates and the corresponding new values for the 'Text' and 'Assignment' fields.
2. **Data Upload to Processing Server:**
   - Once the template table is filled out, the accountant uploads the data to the server. This upload initiates the process of updating the items in the respective SAP transactions.
3. **Server Processing:**
   - The server processes the uploaded data, identifying the relevant items in the FBL3N or FBL5N transactions.
   - It then updates the 'Text' and 'Assignment' fields for each specified item according to the data provided in the template.
4. **Excel Report Generation:**
   - After the updates are successfully applied, the server generates an Excel report.
   - This report details the changes made, including which items were updated and the new values applied to the 'Text' and 'Assignment' fields.
   - The report is then sent back to the accountant for review.

2. Installation

The server component of the application is located in the "app" directory, while the client component is in the "client" directory. Each part must be installed separately by running the "install.bat" files located in their respective root directories.

3. Execution

The server side of the application is started by running the "app.bat" file located in the "app" directory. This batch file requires an email address to be passed to the `%email_id%` parameter. The client side of the application is launched by opening the "app.xlsm" file located 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	Manages data updating in the FBL3N SAP transaction.
server/engine/fbl5n.py	Manages data 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	Creates 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 side of the application was developed using Python 3.9. While older versions (3.7–3.8) may also work, they have not been tested. The Python modules are organized in a horizontal three-tiered architecture, with each layer designed to be self-contained (see Fig. 1).

The **top layer** consists of the "app.py" module, which serves as the program's entry point. This layer is responsible for initializing the application, driving the overall processing of business logic by invoking the controller layer, and performing final cleanup tasks. The application is launched by running the "app.bat" executable, which triggers the execution of the module's code using the interpreter, stored in the local Python environment: "../server/engine/env/python/Scripts/python.exe".

The **middle layer** is represented by the "controller.py" module. This layer contains the processing logic that models the established business processes. It also serves as a bridge between the high-level operations handled by the top layer and the low-level operations performed by the bottom layer.

The **bottom layer** consists of the "sap.py," "fbl3n.py," "fbl5n.py," "mails.py," and "report.py" modules. These modules are responsible for executing low-level operations, including retrieving, preparing, and processing user data within dedicated SAP transactions, generating user reports based on processing output, and sending notifications along with an exception report to the user.

The main application configuration and logging parameters are stored in the "app_config.yaml" and "log_config.yaml" files, which are loaded during application initialization.

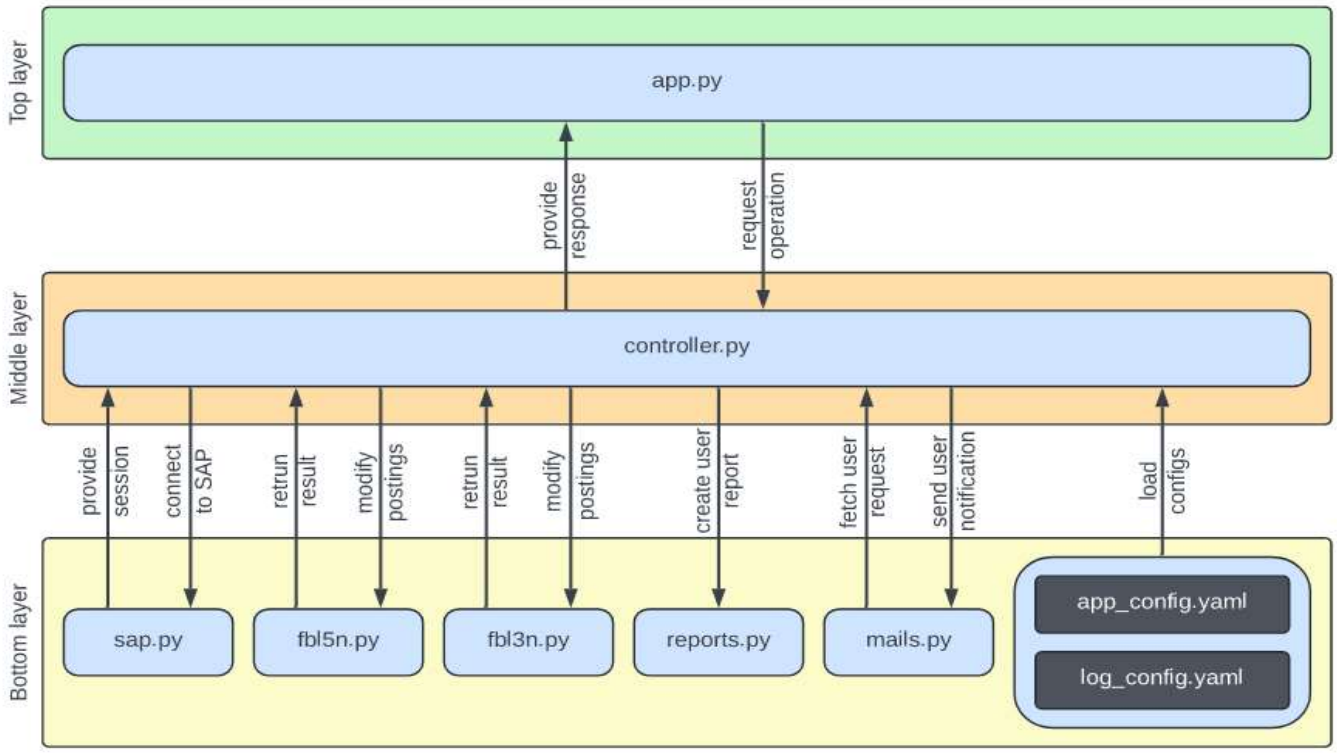


Fig. 1: The layered design of the server part of the application.

5.1 Module “app.py”

The module contains 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 Module “controller.py”

The module serves as the middle layer in the application's design, mediating communication between the top layer (app.py) and the highly specialized modules in the bottom layer (fbl3n.py, fbl5n.py, mails.py, report.py, and sap.py). 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 the application's logging system.

Parameters:

log_dir: The path to the directory where the log file will be stored.

cfg_path: The path to a YAML/YML file containing the application’s configuration parameters.

header: A sequence of lines to be printed 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 YAML/YML file.

Parameters:

cfg_path: The path to the YAML/YML file containing the application’s configuration parameters.

Returns:

A dictionary containing the application configuration parameters.

def **connect_to_sap**(**system** : *str*) -> *CDispatch:*

Description:

Establishes a connection to the SAP GUI scripting engine.

Parameters:

system: The SAP system to connect to using the scripting engine.

Returns:

An SAP *GuiSession* object (wrapped in the *win32:CDispatch* class) representing the active user session.

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

Description:

Closes the connection to the SAP GUI scripting engine.

Parameters:

sess: An SAP *GuiSession* object (wrapped in the *win32:CDispatch* class) representing the 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:

Retrieves processing parameters and data provided by the user.
If the user message is no longer available (e.g. it gets accidentally deleted), then a `RuntimeError` exception is raised. If the processing parameters are not found in the user message or the user provides incorrect values, then `ValueError` exception is raised.

Parameters:

msg_cfg: Configuration parameters for application messages.

email_id: The string ID associated with the message.

Returns:

A dictionary 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.
- "company_code": *str*
  Company code provided by the sender.
- "data": *pandas.DataFrame*
  A DataFrame containing the converted attachment data.
- "account_type": *str*
  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": *dict*
  Details of the attachment:
  - "name": *str*
    The name of the attachment.
  - "content": *any*
    The content of the attachment before conversion.

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

Description:

Modifies accounting item parameters in FBL3N/FBL5N based on user-supplied data.
Currently, only the 'Text' and 'Assignment' fields can be modified.

Parameters:

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

data: The message attachment data in a pandas DataFrame.

acc_type: The type of accounts being processed (e.g., "customer" or "general_ledger").

cocd: The company code associated with the accounts being processed.
data_cfg: Configuration parameters for handling the data.

Returns:

A dictionary containing 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:

Generates a user report based on the processing outcome.

Parameters:

temp_dir: The path to the directory where temporary files are stored.
data_cfg: Configuration parameters for handling the data.
data: The data containing the processing outcome from which the report will be generated.

Returns:

The path to the generated 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 the processing result to the user.

Parameters:

msg_cfg: Configuration parameters for application messages.
user_mail: The email address of the user who requested processing.
template_dir: The path to the directory containing notification templates.
attachment: Either a dictionary containing attachment name and data, or a file path to the attachment.
error_msg: An error message to include in the notification. Default is an empty string (no error message).

Returns:

The procedure does not return an explicit value.

def **delete_temp_files**(**temp_dir** : *str*) -> *None*:

Description:

Removes all temporary files from the specified directory.

Parameters:

temp_dir: The path to the directory where temporary files are stored.

Returns:

The procedure does not return an explicit value.

5.3 Module "sap.py"

The module provides 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.

If the connection fails, a *SapConnectionError* exception is raised.
If the specified SAP executable path is not found, a *FileNotFoundError* exception is raised.

Parameters:

system: The SAP system with the GUI Scripting Engine to connect to:
- "P25": Productive SSO.
- "Q25": Quality Assurance SSO.

exe: The path to the local SAP GUI executable.
If not specified, 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* context object representing the active session to be disconnected.

Returns:

The procedure does not return an explicit value.

5.4 Module "mails.py"

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 Module "flb5n.py"

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 Module “flb3n.py”

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 Module "report.py"

The module creates user reports from the processing output.

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

Description:

Generates an excel report from the processing outcome.

Parameters:

file: The path where the XLSX report file will be created.

data: The DataFrame containing the processing outcome that will be written to the report.

sht_name: Name of the report sheet.

Returns:

The name of the sheet in the Excel report where the data will be written.

5.8 File “app_config.yaml”

This file contains the main configuration settings for the application:

sap:

SAP configuration parameters.

system: *str*

The code of the SAP GUI system to connect to (e. g.: ‘P25’).

data:

Parameters related to data processing.

fbl3n_layout: *str* or *null*

The 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*

The name of the user report file. Any non-empty string is valid (e. g.: ‘report’).

sheet_name: *str*

The name of the data sheet in 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*

The name of the account used to log into the mailbox for the user request emails.

mailbox:

The name of the mailbox containing the user request emails.

server: *str*

The server hosting the mailbox.

notifications:

Parameters related to sending notifications to users.

send: *bool*

If *true*, notifications are sent to users.

sender: *str*

The email address of the notification sender.

subject: *str*

The subject line of the notification.

host: *str*

The server hosting the SMTP service.

port: *int*

The port number used to connect to the SMTP server.

5.9 File "log_config.yaml"

This file contains configuration settings for the application's logging system. The standard logging parameters and their usage are detailed in the official Python documentation. Additionally, this configuration includes custom parameters:

retain_logs_days: `int`
Specifies the number of days that old log files will be retained before they are deleted.

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.

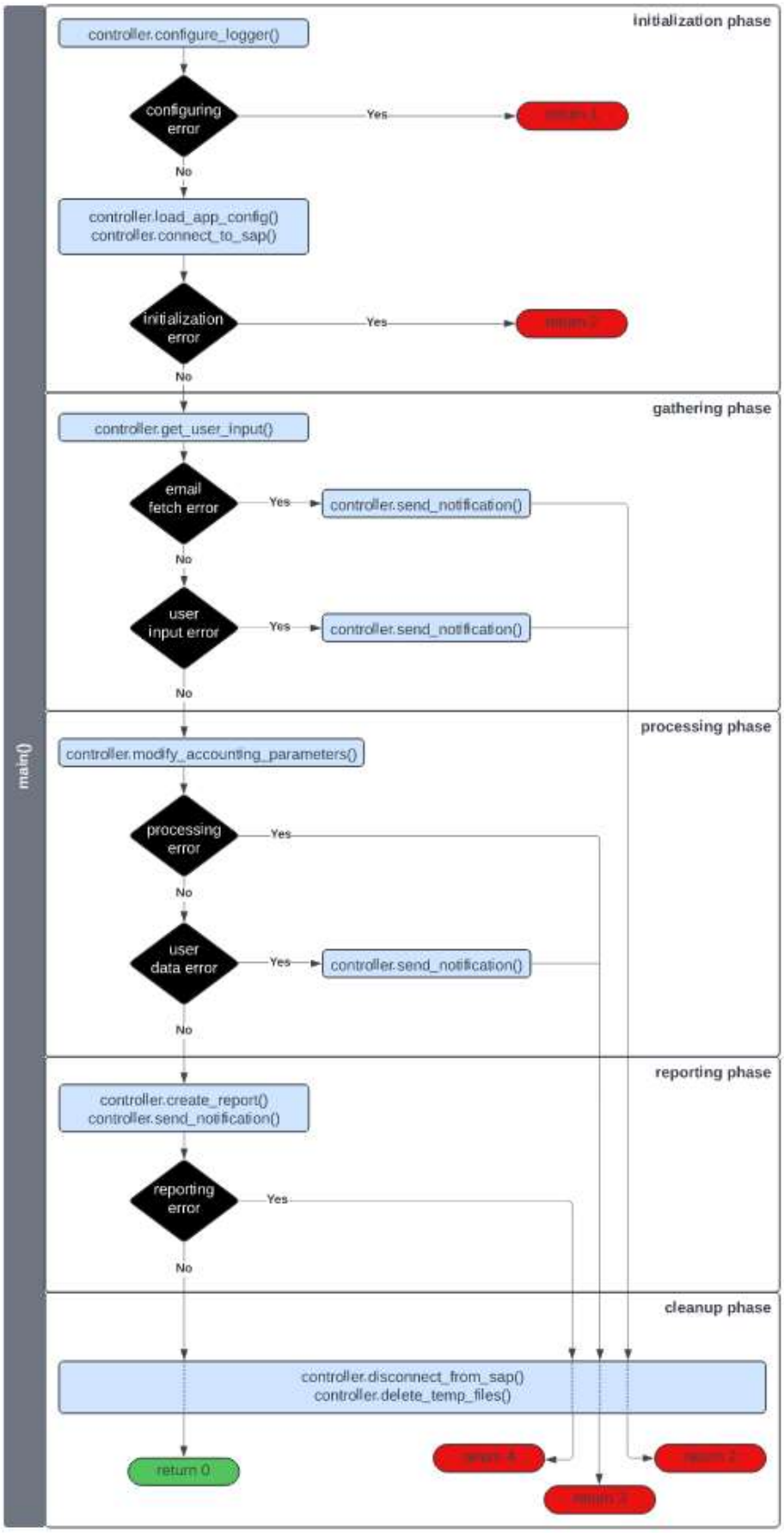


Fig. 2: Program flow from the perspective of the top layer. The execution starts by calling the "main()" procedure. The program then continues with initialization, data gathering, item processing, and finishes by reporting of the outcome to the user. If no internal error occurs, the program finishes 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 File "app.xlsm"

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

- **Account**: Either G/L or customer account numbers.
  Warning: Combining both types of accounts within the same template 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 values to replace the existing 'Assignment' values of open postings.

The file also contains VBA modules that provide procedures for data upload to the processing server

7.1.1. Module "MD_Controller"

Sub **Send**()

Description:

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

7.1.1. Module "MD_Messenger"

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

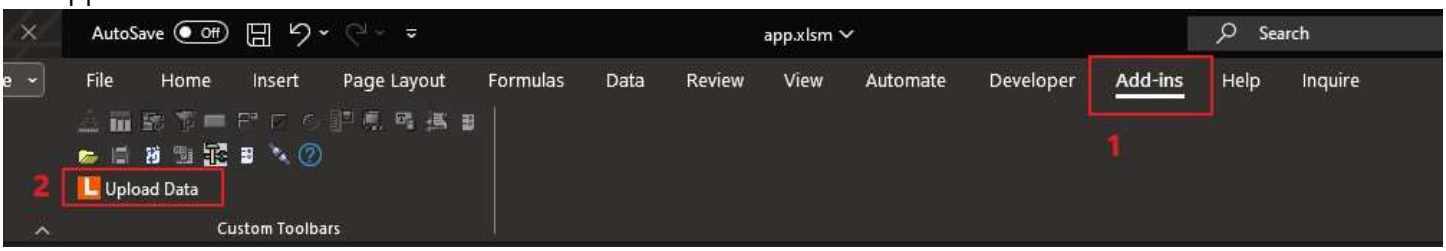
Unzip the *client.zip* archive and install the client application by running the *install.bat* file contained in the root directory. Follow the steps in the installation application. Once the installation is done, a desktop shortcut *Account Updater* is created.

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:

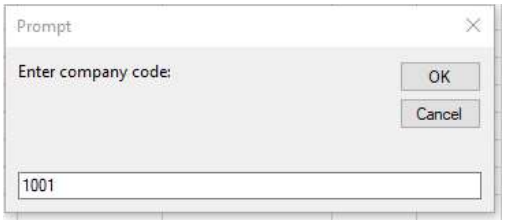
- **Account**: Enter either G/L or customer accounts that contain the open line items to update.
  Warning: Combining both types of accounts in one file is not allowed.
- **Old text**: Enter the current 'Text' values of open line items to change.
- **New text**: Enter new values to replace the existing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

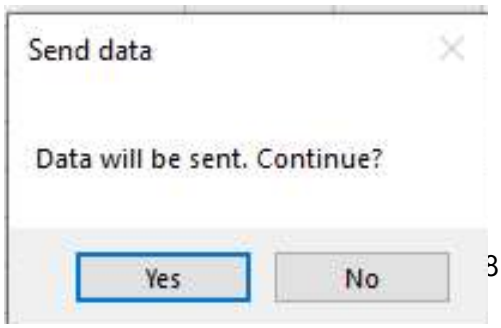
Account	Old Text	New Text	New Assignment
66791580	RE0000634417 D ??? (4928908)	RE0000634417 D 10326857 (4928908)	10326857
66791580	RE0000489944 D ??? (4928847)	RE0000489944 D 10326848 (4928847)	10326848
66791580	RE0000516723 D ??? (4928675)	RE0000516723 D 10326853 (4928675)	10326853
66791580	RE0000196888 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.

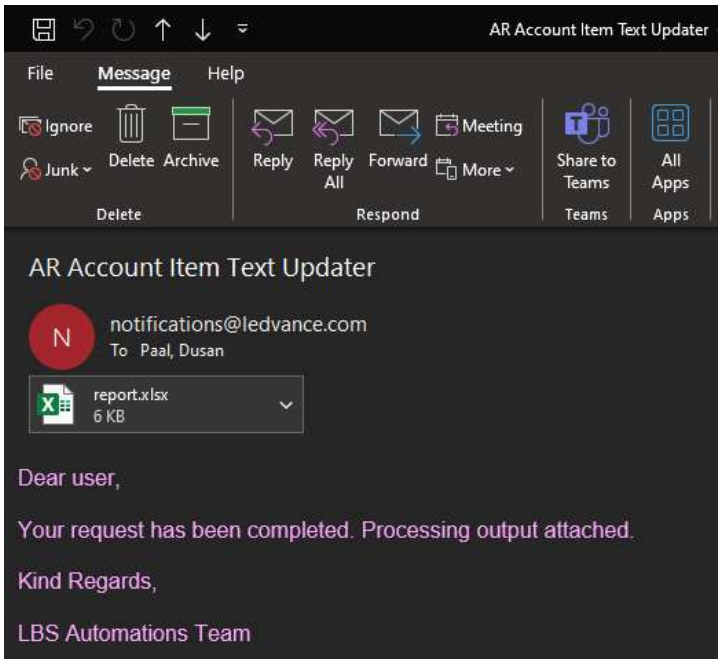


Now you will be prompted to provide the company code under which the entered items were posted. Enter the desired 4-digit value and press the "OK" button. In the final prompt, confirm sending data to server by pressing the "Yes" button if you wish to proceed, or press the "No" button if you wish to cancel 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