AR Overdue Report

Documentation

Contents

1. Workflow Overview

The process of creating monthly reports from overdue customer postings is managed by the Accounts Receivables department. The accountant begins by logging into the SAP system and navigating to transaction code FBL5N. On the selection screen, the accountant enters the necessary parameters such as company code, customer account, key date, and any other relevant filters to focus specifically on overdue items, using criteria like document date, posting date, or due date. Once the parameters are set, the accountant exports the open items data directly from SAP FBL5N into an Excel worksheet. With the data now in Excel, the accountant proceeds to use filtering and various Excel functions to manipulate and format the data into the desired structure for the monthly report. This involves organizing the data to highlight overdue items effectively and ensuring the report meets the required format and standards for internal review and distribution.

1. Application Overview

The "AR Overdue Report" application automates the generation of the monthly overdue report, streamlining the process described previously. The accountant initiates report creation by sending an email with specific processing parameters to a designated email address, using a particular subject line that the server-side application recognizes as a request. Upon receiving the email, the application extracts the processing parameters from the email body. It then exports the relevant data from SAP FBL5N, parses and converts it into a DataFrame format. Once the data is converted, calculations and filtering are applied to refine the data. Finally, the processed data is written to an Excel worksheet, which is then formatted and pivoted according to the report requirements. The completed Excel worksheet is attached to a notification and sent back to the accountant for review.

2. Installation

To install the application, run the “install.bat” file located in the “app” directory. Follow the on-screen instructions provided by the setup program to complete the installation process.

3. Execution

The application is started by running the “app.bat” file contained in the “app” directory. This batch file requires a sender email address to be passed to the %email_id% parameter.

4. Directory and file structure

The application is organized into the following directories and files:

Name	Description
app	Root directory of the application.
app/engine	Contains the application's engine scripts
app/engine/controller.py	Controls high-level operations of the application.
app/engine/dms.py	Exports accounting data form SAP transaction UDM_DISPUTE.
app/engine/fbl5n.py	Exports accounting data form SAP transaction FBL5N.
app/engine/mails.py	Fetches, creates, and sends user emails.
app/engine/sap.py	Connects to / disconnects form the SAP GUI Scripting engine.
app/engine/processor.py	Performs data parsing and applies filters and calculations to the data.
app/engine/report.py	Creates excel reports.
app/env	Contains a local python environment.
app/logs	Contains application runtime logs.
app/data/Austria_1072/customers.xlsx	Contains list of customer-related info mapped to customer accounts.
app/data/OBI_1001/rebuilds.xlsx	Contains list of open item IDs mapped to their business significance.
app/notification	Contains templates for user notifications.
app/notification/template_error.html	Template for error-reporting notifications.
app/notification/template_completed.html	Template for success-reporting notifications.
app/temp	Contains temporary files.
app/app.py	Program entry point of the application.
app/app.bat	Batch file that runs the automation.
app/install.bat	Installs the server part of the application.
app/app_config.yaml	Contains configurable application settings.
app/log_config.yaml	Contains configurable logging settings.
app/rules.yaml	Contains customer-specific data processing parameters.
app/requirements.txt	Contains a list of site-packages and their versions.
doc	Root directory for project documentation.
doc/Documentation_v1.0.docx	Administrator and user manual.

5. Application design

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: "../app/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** comprises the "sap.py", "dms.py", "fbl5n.py", "processor.py", "report.py", and "mails.py" modules. This layer is responsible for executing all low-level operations on emails, SAP transactions and data.

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.

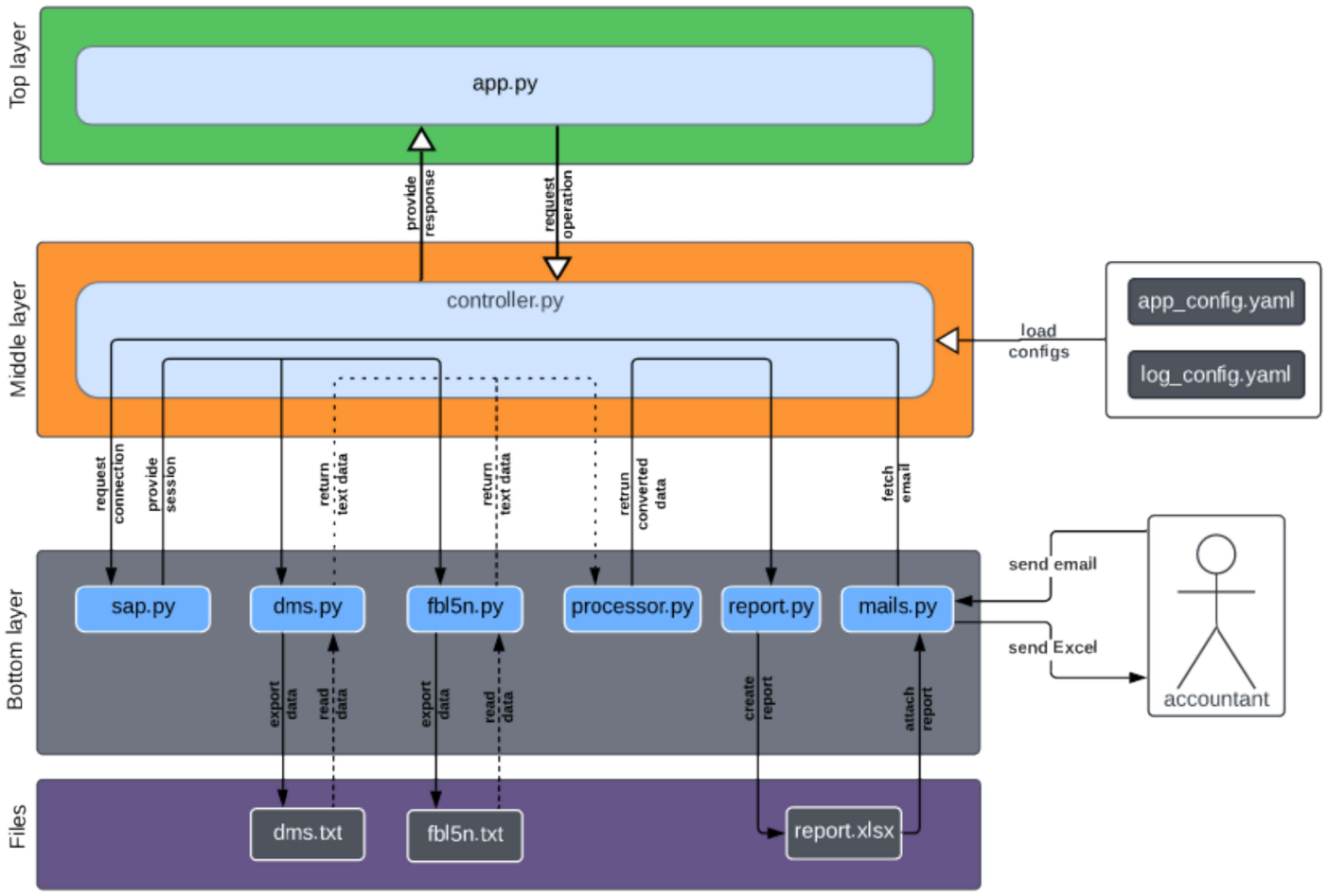

Fig. 1: **The layered design of the application.**

6. Program flow

The main() function is defined as the program's entry point. It takes a dictionary of arguments ("message_id": value) as input. The "message_id" is the ID of the user request message containing the PDF attachment to convert to Excel.

6.1 Initialization

- The program starts by importing necessary modules and calling the *main()* function.
- The program then initializes paths to application directories and paths required during runtime.
- The logging system is configured with the necessary information, such as the application name, version, and current date. If any error occurs during the initialization, the program terminates (return code 1).
- The program attempts to load the application configuration and processing rules from YAML files. If the initialization phase fails, an error message is logged, and the program exits with a return code of 2.
- If the initialization is successful, the program proceeds to fetch user input.

6.2 Fetching of user input

- User input is fetched by calling the *fetch_user_input()* function, which retrieves the user's email message based on the provided email ID.

- If fetching user input fails or if the user input contains an error message indicating a failure in fetching the email message, an error is logged, the program closes the connection to SAP, and exits with a return code of 2.
- If user input is successfully fetched, the program proceeds to the processing phase.

6.3 Data processing

- The program calls the *controller.fetch_fbl5n_data()* function to export the accounting data from the transaction FBL5N.
- The program calls the *controller.fetch_dms_data()* function to export the disputed data form the transaction UDM_DISPUTE.
- The program calls the evaluate_data() that wrapt data parsing, conversion, combination of the converted FBL5N and DMS data into a single table which is then evaluated according to the specific processing rules contained in the "rules.yaml" file.
- If the processing phase fails, an error message is logged, temporary files are deleted, connection to SAP is closed and the program exits with a return code of 3.
- If the processing is successful, the program proceeds to the reporting phase.

6.4 Reporting

- The program generates an Excel report and sends a notification to the accountant with the report workbook attached.
- If sending the notification fails, an error message is logged, and the program exits with a return code of 4.

6.5 Cleanup

- Finally, the program performs cleanup by deleting temporary files, closes connection to SAP, logs the system shutdown, shuts down the logging system, and exits with a return code of 0.

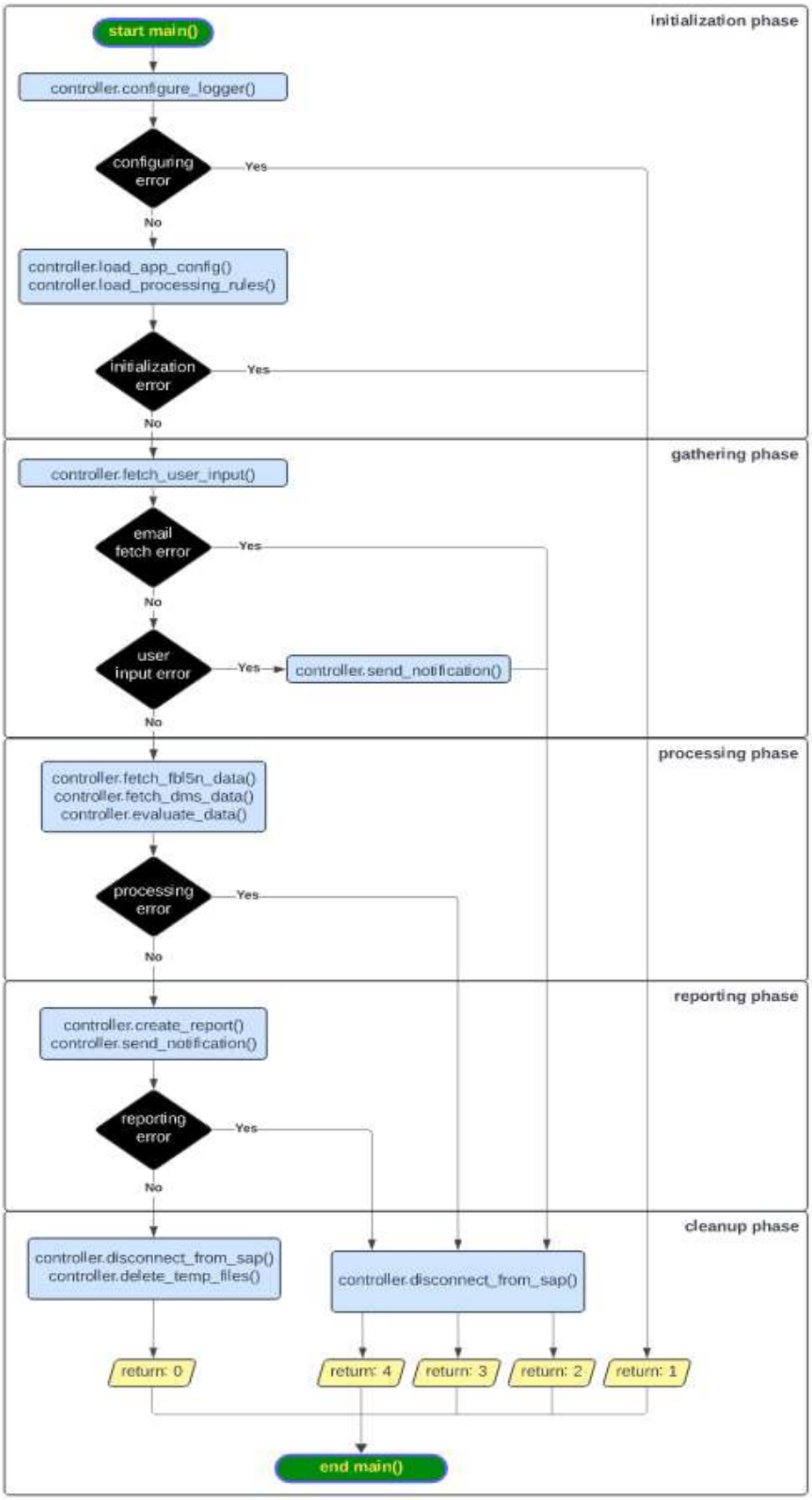

Fig. 2: Program flow from the perspective of the top layer.

7. Modules and files

7.1 Module “app.py”

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

def **main**(****args**) -> *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.

7.2 Module “controller.py”

The controller.py represents the middle layer of the application design and mediates communication between the top layer (app.py) and the highly specialized modules situated on the bottom layer of the design (mails.py, report.py, processor.py, sap.py, dms.py fbl5n.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 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-specific parameters for data evaluation.

Parameters:

file_path: Path to the file containing the processing rules.

Returns:

Data evaluation 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*) Default: ""
  Message that details any errors in user email.
- "error_message": (*str*) Default: "I"
  Type of the message:
  "I": The message is an information.
  "W": The message is a warning.
  "E": The message is an error.
- "email": (*str*) Default: ""
  Email address of the sender.
- "entity": (*str*) Default: ""
  Entity for which the overdue report is created (a worklist or a company code).
- "overdue_day": (*date*) Default: None
  Date for which the open line items are exported from FBL5N.

def **fetch_fbl5n_data**(
temp_dir : *str*, **entity** : *str*, **overdue_day** : *str*, **rules** : *dict*, **data_cfg** : *dict*, **session** : *CDispatch*
) -> *DataFrame*:

Description:

Fetches open line items from the FBL5N transaction. First, the data is exported form the FBL5N as raw text. Then, the FBL5N text data is parsed into a *DataFrame* object.

Parameters:

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

entity: Entity for which the open line items are exported. Can be either a company code or a worklist.
overdue_day: Day for which the open line items are exported.
rules: Entity-specific data processing rules.
data_cfg: Application 'data' configuration parameters.
session: An SAP *GuiSession* object.

Returns:

The FBL5N accounting data.

def **fetch_dms_data**(**temp_dir** : *str*, **data_cfg** : *dict*, **cases** : *Series*, **session** : *CDispatch*) -> *DataFrame*:

Description:

Fetches data for disputed cases from the UDM_DISPUTE transaction.

Parameters:

temp_dir: Path to the directory where temporary files are stored.
data_cfg: Application 'data' configuration parameters.
cases: Case ID numbers extracted from the "Text" field of the FBL5N data.
session: An SAP *GuiSession* object.

Returns:

The DMS cases data.

def **evaluate_data**(**fbl5n_data** : *DataFrame*, **dms_data** : *DataFrame*, **data_dir** : *str*, **entity** : *str*, **rules** : *dict*) -> *DataFrame*:

Description:

Evaluates the FBL5N and DMS data.

Parameters:

fbl5n_data: The FBL5N accounting data.
dms_data: The DMS dispute data.
data_dir: Path to the application directory where accounting data used for the evaluation is stored.
entity: Entity for which the open line items are exported.
rules: Entity-specific data processing rules.

Returns:

The result of the data evaluation.

def **create_report**(**data** : *DataFrame*, **report_cfg** : *dict*, **entity** : *str*, **rules** : *dict*, **overdue_day** : *date*, **temp_dir**: *str*) -> *str*:

Description:

Creates user report from the processing result.

Parameters:

data: The evaluation result from which report will be generated.
data_cfg: Application 'data' configuration parameters.
entity: Entity for which the open line items are exported.
rules: Entity-specific data processing rules.
overdue_day: Day for which the open line items are exported.
data_dir: The application directory where additional accounting info files are stored,
which are necessary for creating pivot tables in the user report.
temp_dir: Path to the directory where temporary files are stored.

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.

7.3 Module "mails.py"

The mails.py module provides a simplified interface for managing emails for a specific account hosted on an Exchange Web Services (EWS) server. It streamlines common email-related tasks, such as sending, receiving, and organizing emails, through an intuitive interface. Most of the procedures within this module rely on the exchangelib package, which must be installed and properly configured before using the module.

def **get_account**(**mailbox** : *str*, **name** : *str*, **x_server** : *str*) -> *Account:*

Description:

Retrieves and models an MS Exchange server user account based on the provided parameters.

Parameters:

mailbox: The name of the shared mailbox associated with the user account.
name: The name of the user account to retrieve.
x_server: The name of the MS Exchange server hosting the mailbox.

Returns:

An object representing the user account retrieved from the MS Exchange server.

Raises:

CredentialsNotFoundError:
If the file containing the account credentials cannot be found at the specified path.
CredentialsParameterMissingError:
If a required credential parameter is missing in 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: The email address of the sender.

recipient: The email address of the recipient, or a list of email addresses.

subject: The subject of the email message.

body: The body of the email message in HTML format.

attachment: Specifies the attachment(s) to be included with the email:

- *None* (default): No attachment will be included.
- *FilePath*: A valid file path to a single file.
- *list*: A list of file paths to be attached.
- *dict*: A dictionary where keys are file names and values are either file paths or *bytes-like* objects.
  - If the value is a file path, the corresponding file will be attached using the key as the attachment name.
  - If the value is a *bytes-like* object, its contents will be attached with the key used as the attachment name.

An invalid file path will raise a *FileNotFoundError*.

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.

Parameters:

msg: The message object to be sent.

host: The SMTP host server used to send the message.

port: The port number of the SMTP server.

timeout: The number of seconds to wait for the message to be sent. Defaults to 30 seconds.
If the timeout is exceeded, a *TimeoutError* exception will be raised.

debug: The debug level for capturing connection messages and interactions with the server:

- 0: Debugging is off (default).
- 1: Verbose debugging.
- 2: Timestamped debugging.

Returns:

The procedure does not return an explicit value.

Raises:

UndeliveredError: Raised if the message is not delivered to all recipients.

TimeoutError: Raised if the operation exceeds the specified timeout.

def **get_messages**(**acc** : *Account*, **email_id** : *str*) -> *list:*

Description:

Fetches messages with a specific message ID from an inbox.

Parameters:

acc: The account used to access the inbox where the messages are stored.

email_id: The ID of the message to fetch (corresponding to the "Message.message_id" property).

Returns:

A list of `exchangelib.Message` objects representing the retrieved messages.
If no messages with the specified ID are found, an empty list is returned.
This may occur if the message ID is incorrect or if the message has been deleted

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

Description:

Fetches attachments from a message, filtering them by file extension.

Parameters:

msg: The message object from which attachments are to be fetched.

ext: The file extension to filter attachments by. Defaults to ".*", which fetches all file types.
If a specific extension (e. g., ".pdf") is provided, only attachments with that file type will be fetched.

Returns:

A list of dictionaries, each containing the following attachment details:
- "name" (str): The name of the attachment file.
- "data" (bytes): The binary data of the attachment.

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

Description:

Saves message attachments to a specified local folder.

Parameters:

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

Dst: The path to the folder where the attachments will be saved. - If the destination folder doesn't exist, a *FolderNotFoundError* exception will be raised.

ext: The file extension used to filter the attachments to be downloaded.
If a specific file extension (e. g., '.pdf') is provided, only attachments of that type will be downloaded.

Returns:

A list of file paths to the stored attachments.

7.4 Module "report.py"

The module generates Excel overdue reports form the evaluated data.

def **create_report_obi_de**(**file** : *FilePath*, **evaluated** : *DataFrame*, **fields** : *list*, **sht_names** : *dict*) -> *None*:

Description:

Creates Excel overdue report for OBI Germany from the evaluated data.

Parameters:

file: The path to the .xlsx report file to be created.

evaluated: The evaluated data that will be written to the report sheet.

fields: A list of fields that defines columns to include in the overdue report, specifying their order.

sht_names: A dictionary mapping data types to their respective sheet names in the Excel file.

Returns:

The procedure does not return an explicit value.

def **create_report_austria**(**file** : *FilePath*, **evaluated** : *DataFrame*, **fields** : *list*, **sht_names** : *dict*) -> *None*:

Description:

Creates Excel overdue report for Austria from the evaluated data.

Parameters:

file: The path to the .xlsx report file to be created.

evaluated: The evaluated data that will be written to the report sheet.

fields: A list of fields that defines columns to include in the overdue report, specifying their order.

sht_names: A dictionary mapping data types to their respective sheet names in the Excel file.

Returns:

The procedure does not return an explicit value.

7.5 Module “processor”

The module contains procedures for performing operations on accounting data such as parsing, cleaning, conversion, evaluation and calculations.

def **convert_fbl5n_data**(**data** : *str*, **case_id_rx** : *str*) -> *DataFrame*:

Description:

Converts plain FBL5N text data into a panel dataset.

Parameters:

data: The plain text data exported form the FBL5N transaction.

case_id_rx: A regex pattern used for matching and extracting country-specific case ID numbers from the 'Text' field in the data.

Returns:

The converted data as a pandas DataFrame with the following columns and data types:

- "Head_Office": `UInt64`
- "Branch: `UInt64`
- "Currency": `string`
- "Document_Number": `UInt64`
- "Document_Type": `string`
- "Document_Date": `object`
- "Due_Date": `object`
- "Arrears": `Int32`
- "Clearing_Document": `UInt64`
- "DC_Amount": `float64`
- "Account_Assignment": `string`
- "Tax": `string`
- "Text": `string`
- "Clearing_Date": `datetime64[ns]`
- "Case_ID": `UInt64`

def **convert_dms_data**(**data**: *str*) -> *DataFrame*:

Description:

Converts plain DMS text data to a panel dataset.

Parameters:

data: The plain text data exported form the UDM_DISPUTE transaction.

Returns:

The converted data (column name and data type):

- "Case_ID": `UInt64`
- "Debitor": `UInt64`
- "Created_On": `object`
- "Processor": `string`
- "Status_Sales": `string`
- "Status_AC": `string`
- "Notification": `UInt64`
- "Category_Description": `string`

- "Category": `UInt8`
- "Root_Cause": `category`
- "Autoclaims_Note": `string`
- "Fax_Number": `string`
- "Status": `UInt8`
- "DMS_Assignment": `object`

def **evaluate_obi_de**(**fbl5n_data** : *DataFrame*, **dms_data** : *DataFrame*, **queries** : *dict*, **acc_data_paths** : *str*) -> *DataFrame*:

Description:

Evaluates overdue parameters for OBI Germany from the exported FBL5N and DMS data.

Parameters:

fbl5n_data: The converted FBL5N data.

dms_data: The converted DMS data.

queries: Pandas data queries and their respective descriptions.
The queries are executed to the data, and, if any records are found, the respective query description is written to the "Note" column of the queried data.

acc_data_paths: Paths to Excel (xlsx) files that contain an additional info for each disputed case.
The information is then joined to the case ID numbers in the FBL5N data.

Returns:

The evaluation result.

def **evaluate_austria**(
fbl5n_data : *DataFrame*, **dms_data** : *DataFrame*, **customer_data** : *FilePath*, **queries** : *dict*
) -> *DataFrame*:

Description:

Evaluates overdue parameters for Austria from the exported FBL5N and DMS data.

Parameters:

fbl5n_data: The converted FBL5N data.

dms_data: The converted DMS data.

customer_data: Path to the customers.xlsx file that contains an additional info for each customer account, such as the responsible Sales Person, Channel, customer name and country. The information is then joined to the customer accounts in the FBL5N data.

queries: Pandas data queries and their respective descriptions.
The queries are executed to the data, and, if any records are found, the respective query description is written to the "Note" column of the queried data.

Returns:

The evaluation result.

7.6 File "app_config.yaml"

This file contains the main application configuration:

sap:

Parameters related to connecting to SAP.

system: *str*

The SAP system to connect to.

data:

Parameters related to data export from SAP.

fbl5n_layout: *str*

Name of the layout used for FBL5N data.

dms_layout: *str*
Name of the layout used for DMS data.

report:
Parameters related to generating Excel reports.

report_name: *str*

The report name consists of a fixed string and placeholders that can be replaced by actual values:

- $entity$: to be replaced by a company code or a worklist name
- $company_code$: to be replaced by a company code value
- $date$: to be replaced by the overdue date provided by the user

Example: "AR_Overdue_Report_$entity$_$company_code$_$date$.xlsx"

messages:
Parameters for management of emails.

requests:
Parameters related to processing incoming user requests.

account: *str*
The name of the account used to log into the mailbox containing user request emails.
mailbox: *str*
The name of the mailbox containing user request emails.
server: *str*
The name of the server hosting the mailbox containing user request emails.

notifications:
Parameters related to sending notifications to users.

send: *bool*
Indicates whether notifications with processing results are sent to users.
sender: *str*
The email address of the notification sender.
subject: *str*
The subject of the user notification.
host: *str*
The name of the server hosting the SMTP service.
port: *int*
The port number used to connect to the host.

7.7 File "log_config.yaml"

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.

7.8 File “rules.yaml”

This file contains entity-specific parameters (rules) that control the processing specifics of the parsed data. The entity can be a customer name (e. g. “Markant”) or a company code (e. g. “1001”).

Entity: *str*
Parameters that control the processing of documents for a specific entity.

country: *str*
The name of the country where the entity is located.
case_id_rx: *str*
The regex pattern that matches case ID numbering for the entity.
company_code: str
A 4-digit number indicating the company code of the entity.
type: *str*
The type of the entity (worklist or company code)
queries: dict[*str* : *str*]
A dictionary of pandas query strings mapped to the text values to place in the “Text” field where the data matches the query.
Example: "Status == 1 and Root_Cause == 'L01'": "Klärung bei CS Nearshore."
report_fields: *list*[*str*], default: [Document_Number, Document_Type, DC_Amount, Currency, Tax, Document_Date, Due_Date, Overdue_Days, Head_Office, Branch, Debitor, Clearing_Document, Text, Case_ID, ID_Match, Amount_Match, Tax_Match, Status, Status_Sales, Note, Status_AC, Notification, Created_On, Category, Category_Desc, Root_Cause, Autoclaims_Note, Fax_Number, Processor, DMS_Assignment]
The list of field names that defines the order of the fields in the resulting Excel table. The order can be changed, or field names can be removed as needed. New field names can be added only if they are implemented in the data processor.
report_sheets:
Parameters related to report sheets.
data_sheet_name: *str*

8. Revision

Version	Date	Author	Description
1.0	20.11.2023	Dusan Paal	Initial version