AR Payment Advice Converter

Documentation

Contents

1. Workflow Overview

The payment advice conversion process from PDF to Excel is overseen by the Accounts Receivables department. The accountant begins by copying data from a PDF document representing a payment advice into an Excel worksheet. The accountant then uses filtering and Excel functions to prepare the data into a format that serves as a template for allocating items in customer payments.

1. Application Overview

The "AR Payment Advice Converter" application automates the extraction of accounting data from customer payment advice notes received as PDF files. The accountant initiates the process by sending the PDF document to a specified email address with a specific subject line, which the server-side application uses to recognize the request. The application then downloads the attached PDF files, converts them to plain text, and extracts all relevant accounting information from the text. The extracted data is subsequently post-processed, formatted, and placed on a separate sheet within an Excel workbook, which is then emailed back to the user. Currently, the application supports the conversion of payment advice notes issued by OBI Germany and Markant Germany.

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 for the server part of the application.
app/engine	Contains the engine scripts of the server part.
app/engine/processor	Contains modules involved in data processing.
app/engine/processor/__init__.py	Initializes the data processing package.
app/engine/processor/markant.py	Extracts data form Markant documents.
app/engine/processor/obi.py	Extracts data form Obi documents.
app/engine/controller.py	Controls high-level operations of the application.

app/engine/mails.py	Fetches, creates, and sends user emails.
app/engine/excel.py	Creates user excel files.
app/env	Contains a local python environment.
app/logs	Contains application runtime logs.
app/maps	Contains accounting maps
app/maps/markant_de.json	Maps ILN numbers to customer accounts for Markant Germany.
app/maps/obi_de.json	Maps supplier and branch numbers to customer accounts for OBI Germany
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/engine/processor/pdftotext.exe	Extracts text from a pdf file.
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 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: "../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 "mails.py" and "excel.py" modules, as well as the processor package, which contains the "__init__.py", "markant.py", and "obi.py" modules. This layer is responsible for executing all low-level operations on emails and data. These tasks include retrieving PDF documents from the user's email, performing text extraction, converting the data, writing the converted data to an Excel file, and sending the notification along with the Excel file back 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.

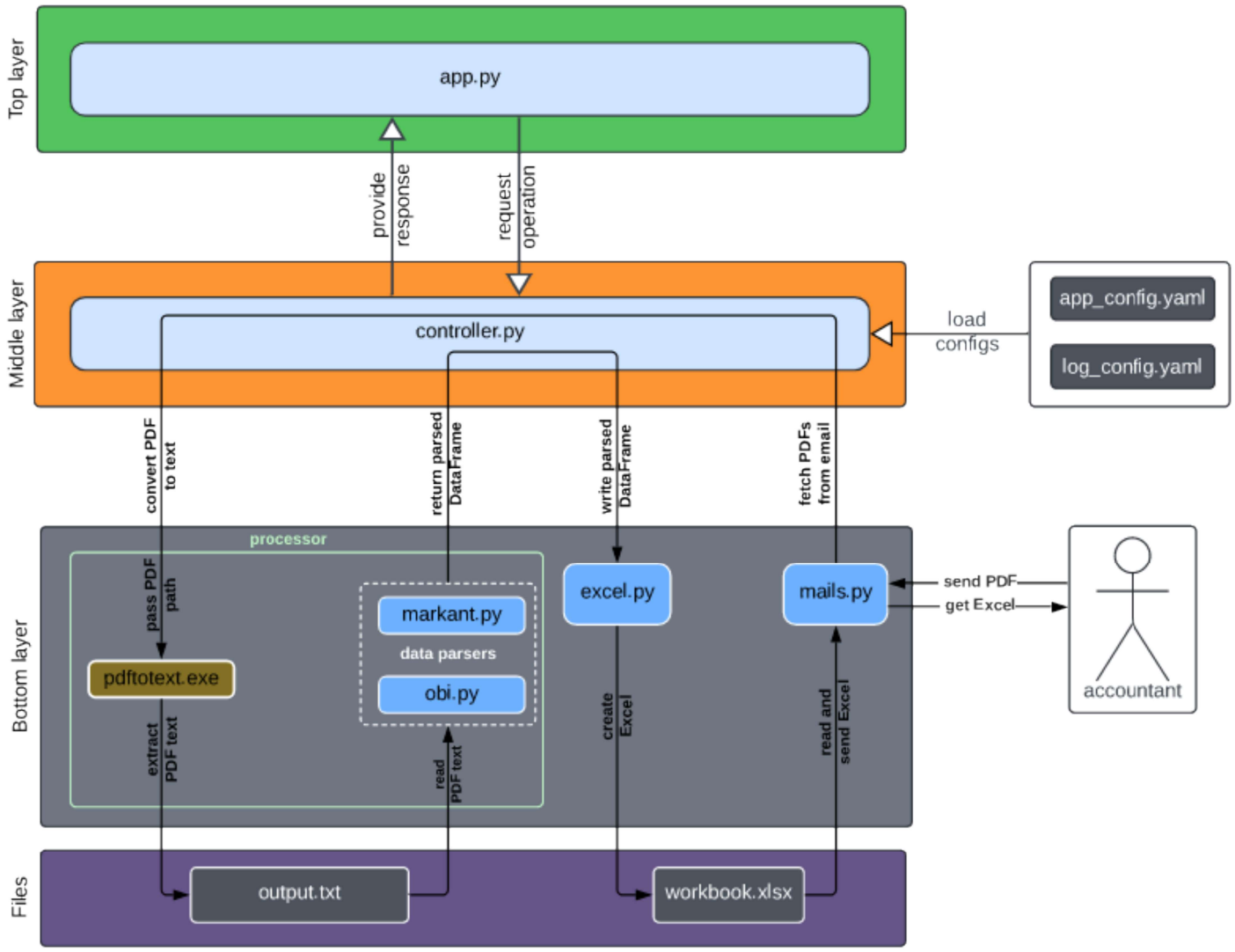


Fig. 1: **The layered design of the application.**

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, an error message is logged, temporary files are deleted, and the program exits with a return code of 2. Also, if there is an error message in the user input, indicating a failure in fetching the email message, a notification is sent to the user with the error message. The program exits with a return code of 2.
- If user input is successfully fetched, the program proceeds to the processing phase.

6.3 Document processing

- The program calls the *convert_documents()* function to convert the attached PDF files to plain text and extract relevant accounting information.
- If the processing phase fails, an error message is logged, temporary files are deleted, and the program exits with a return code of 3. Also, if there is an error message in the processing output, indicating a failure in processing the PDF files, a notification is sent to the user with the error message. Temporary files are deleted, and the program exits with a return code of 0.
- If the processing is successful, the program proceeds to the reporting phase.

6.4 Reporting

- The program sends a notification to the accountant with the extracted accounting data in an Excel 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, logs the system shutdown, shuts down the logging system, and exits with a return code of 0.

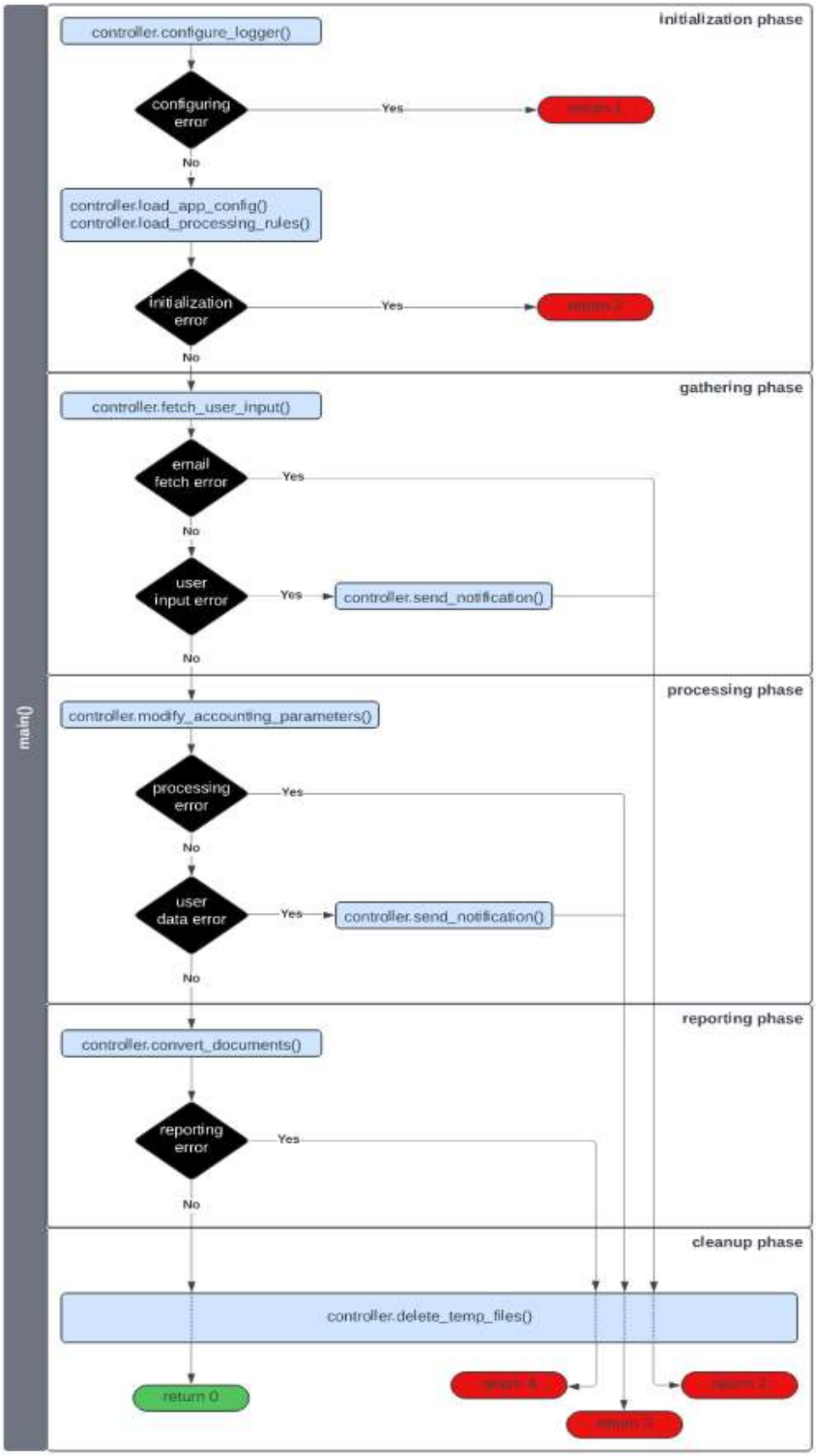


Fig. 2: Program flow from the perspective of the top layer.

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** : *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.

7.2 Module “controller.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-specified data processing parameters.

Parameters:

accounts_map_dir: Path to the directory where account maps are stored.

file_path: Path to the file containing the processing rules.

Returns:

Data processing parameters.

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

Description:

Creates connection to the SAP GUI scripting engine.

Parameters:

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

Returns:

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

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

Description:

Closes connection to the SAP GUI scripting engine.

Parameters:

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

Returns:

The procedure does not return an explicit value.

def **fetch_user_input**(**msg_cfg** : *dict*, **email_id** : *str*) -> *dict:*

Description:

Fetches the processing parameters and data provided by the user.

Parameters:

msg_cfg: Application 'messages' configuration parameters.

email_id: The string ID of the message.

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

Returns:

Names of the processing parameters and their values:

- "error_message": (*str*) A detailed error message if an exception occurs.
- "email": (*str*) Email address of the sender.
- "attachment_paths": (*list[str]*) List of paths to downloaded attachments.

def **convert_documents**(**rules** : *dict*, **pdf_paths** : *str*, **temp_dir** : *str*, **extractor_path** : *str*, **excel_cfg** : *dict*) -> *dict*:

Description:

Converts payment advice PDFs to Excel files.

First, the customer who issued the document is identified from the document contents.
Then, data is extracted from the document based on the customer-specific rules.

Parameters:

rules: Customer-specific data processing parameters.

pdf_paths: Local paths to the downloaded PDF attachments.

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

extractor_path: Path to the executable that performs text extraction from a PDF.

excel_cfg: Application excel configuration parameters.

Returns:

The processing result, with the following keys and values:

- "error_message": (*str*) An error message if an exception occurs, otherwise "".
- "excel_paths": (*list[str]*) Paths to the generated Excel file(s) if a PDF was successfully converted, otherwise [].

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

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

Description:

Models an MS Exchange server user account.

Parameters:

mailbox: Name of the shared mailbox.
name: Name of the user account.
x_server: Name of the MS Exchange server.

Returns:

The user account object.

Raises:

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

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

def **create_smtp_message**(
sender : *str*, **recipient** : *Union[str, list]*, **subject** : *str*, **body** : *str*,
attachment : *Union[FilePath, list, dict]* = None) -> *SmtpMessage:*

Description:

Creates an SMTP-compatible message.

Parameters:

sender: Email address of the sender.
recipient: Email address or addresses of the recipient.
subject: Message subject.
body: Message body in HTML format.
attachment:
- *None* : The message will be created without any attachment.
- *FilePath* : Path to the file to attach.
- *list* [*FilePath*]: Paths to the files to attach.
- *dict* {*str* : *FilePath*} : file names and paths to attach.
  Attachment type is inferred from the file type.
  The file names will be used as attachment names.
  An invalid file path raises `FileNotFoundError` exception.
- *dict* {*str* : *bytes*} : file names and `byte-like` objects to attach
  Attachment type is inferred from the file name.
  If the data type cannot be inferred, then a raw binary
  object is attached. The file names will be used as attachment names.

Returns: The constructed message.

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

Description:

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

Parameters:

msg: Message to send.
host: Name of the SMTP host server used for message sending.
port: Number of the SMTP server port.
timeout: Number of seconds to wait for the message to be sent (default: 30).
Exceeding this limit will raise an *TimeoutError* exception.
debug: Whether debug messages for connection and for all messages
sent to and received from the server should be captured:
- 0: "off" (default)
- 1: "verbose"
- 2: "timestamped"

Returns:

The procedure does not return an explicit value.

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

Description:

Fetches messages with a specific message ID from an inbox.

Parameters:

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

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

Returns:

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

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

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

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

Description:

Fetches message attachments and their names.

Parameters:

msg: Message from which attachments are fetched.

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

By default, any file attachments are fetched. If an extension (e. g. ".pdf")
is used, then only attachments with that file type are fetched.

Returns:

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

- "name" (*str*): Name of the attachment.
- "data" (*bytes*): Attachment binary data.

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

Description:

Saves message attachments to a local folder.

Parameters:

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

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

By default, all attached files are downloaded. If a file extension (e.g.: '.pdf') is used,
then only attachments of the specified file type are downloaded.

Returns:

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

7.4 Module "excel.py"

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

def **generate_excel_file**(**data** : *DataFrame*, **file** : *FilePath*, **data_sht_name** : *str*, **notes_sht_name** : *str*, **customer** : *str*) -> *None:*

Description:

Creates an excel file from the converted payment advice data.

Parameters:

data: Accounting items extracted from the payment advice.

file: Path to the .xlsx file to create.

data_sht_name: Accounting data sheet name.

notes_sht_name: Notes data sheet name.

customer: Name of the customer issuing the remittance advice.

Returns:

The procedure does not return an explicit value.

7.5 Module “processor”

The package provides an interface for extraction and parsing of text data from PDF documents that represent payment (remittance) advice. As of the current version, processing of documents issued by Markant Germany and Obi Germany is supported. Parsing of documents issued by Markant Austria should work as well but has not been tested.

def **parse_amount**(**val**: *str*) -> *float:*

Description:

Parses SAP amount string.

Parameters:

val: Value to parse.

Returns:

The parsed value in the *float* data type.

def **parse_amounts**(**vals**: *Series*) -> *Series:*

Description:

Parses SAP amount strings.

Parameters:

vals: String values to parse stored in a *pandas.Series* object.

Returns:

Parsed values converted to *float64* data type, stored in a *pandas.Series* object.

def **identify_customer**(**pdf** : *FilePath*, **dst** : *DirPath*, **extractor** : *FilePath*) -> *str:*

Description:

Identifies the issuer of a PDF remittance advice from the document text.

If the procedure fails to identify the issuer, then an *UnrecognizedCustomerError* exception is raised.
If the text extraction from the PDF fails, then a *PdfConversionError* exception is raised.

Parameters:

pdf: Path to a PDF document that represents the payment advice.
If the PDF is not found at the specified path, then a *FileNotFoundError* exception is raised.

dst: Path to the folder to store the output text file.
If the folder is not found at the specified path, then a *FolderNotFoundError* exception is raised.

extractor: Path to the executable (.exe) file that extracts text from a PDF.
If the extractor is not found at the specified path, then a *FileNotFoundError* exception is raised.
If an invalid extractor file format is used, then a *ValueError* exception is raised.

Returns:

Name and country code of the customer who issued the document:
'OBI_DE': for OBI Germany
'MARKANT_DE': for Markant Germany

def **extract_text**(**pdf** : *FilePath*, **dst** : *DirPath*, **extractor** : *FilePath*, **options** : *str*) -> *str:*

Description:

Extracts text from a PDF document.

Refer to the official PdfToText project documentation on how to use the PDF extractor.
If the text extraction from the PDF fails, then a PdfConversionError exception is raised.

Parameters:

pdf: Path to a PDF document that represents the payment advice.
If the PDF is not found at the specified path, then a *FileNotFoundError* exception is raised.

dst: Path to the folder to store the output text file.
If the folder is not found at the specified path, then a *FolderNotFoundError* exception is raised.

extractor: Path to the executable (.exe) file that extracts text from a PDF.

If the extractor is not found at the specified path, then a *FileNotFoundError* exception is raised.
If an invalid extractor file format is used, then a *ValueError* exception is raised.

options: Conversion options passed to the PDF extractor.

Returns:

Name and country code of the customer who issued the document:
"OBI_DE": if the issuer is OBI Germany
"MARKANT_DE": if the issuer is Markant Germany

def **markant.parse**(**text** : *str*, **accounting_map** : *dict*, **threshold** : *float*, **fields** : *list*, **date_format** : *str* **)** -> *dict*:

Description:

Parses the text extracted from a PDF payment advice.

If parsing of the text fails, then a *ParsingError* exception is raised.

Parameters:

text: Plain text extracted from remittance advice.
accounting_map: ILN numbers mapped to customer accounts.
threshold: The amount limit below which items are written off.
fields: Field names that define the ordering of columns in the processed data.
date_format: An explicit format string that controls the resulting format of the document date.

Returns:

Accounting parameters extracted from the document and their values:
"items": (*pandas.DataFrame*) Accounting items.
"supplier_id": (*str,* default: "") Not applicable for Markant.
"remittance_number": (*str*) Document number.
"remittance_date": (*str*) Document date.
"remittance_name": (*str*) Name of the remittance advice in local language as stated in the document.
"remittance_type": (*str*) Type of the remittance advice:
"invoicing": for Journal 10 - Rechnungen/Gutschrifte
"other": for Journal 20 - Belastungen/Ruckbelastungen
"services": for Journal 22 - Sonstige Leistungen
"corrections": for Journal 30 - Korrekturen
"": for unrecognized payment advice type.

def **obi.parse**(**text** : *str*, **accounting_map** : *dict*, **threshold** : *float*, **fields** : *list*, **date_format** : *str* **)** -> *dict*:

Description:

Parses the text extracted from a PDF payment advice.

If parsing of the text fails, then a *ParsingError* exception is raised.

Parameters:

text: Plain text extracted from remittance advice.
accounting_map: Supplier ID numbers and branch numbers to customer accounts.
threshold: The amount limit below which items are written off.
fields: Field names that define the ordering of columns in the processed data.
date_format: An explicit format string that controls the resulting format of the document date.

Returns:

Accounting parameters extracted from the document and their values:
"items": (*pandas.DataFrame*) Accounting items.
"supplier_id": (*str,* default: "") Ledvance listing ID in the customer's accounting.
"remittance_number": (*str*) Document number.
"remittance_date": (*str*) Document date.

"remittance_name": (*str,* default: “”) Not applicable for OBI.
"remittance_type": (*str,* default: “”) Not applicable for OBI.

7.6 File “app_config.yaml”

This file contains the main application configuration:

excel:
Parameters related to Excel file generation.

data_sheet_name: *str,* default: ‘Data’
Name of the sheet where extracted data is written.

notes_sheet_name: *str,* default: ‘Notes’
Name of the sheet where the additional information data is written.

messages:
Parameters related to message processing.

requests:
Parameters related to processing incoming user requests.

account: *str*, default: ‘lbs.robot@ledvance.com’
Name of the account used to log into the mailbox with the user request emails.
mailbox: *str*, default: ‘lbs.robot@ledvance.com’
Name of the mailbox with the user request emails.
server: *str*, default: ‘outlook.office365.com’
Name of the mailbox with the user request emails.

notifications:
Parameters related to sending notifications to users.

send: *bool*, default: true
Whether notifications with processing result are sent to users.
sender: *str,* default: ‘notifications@ledvance.com’
Email address of the notification sender.
subject: *str*, default: ‘Notification of payment advice conversion’
Subject of the user notification.
host: *str*, default: ‘intrelay.ledvance.com’
Name of the server hosting the SMTP service.
port: *int*, default: 25
Number of the port used to connect to the host.

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 customer-specific parameters (rules) that control the processing specifics of the PDF to Excel data conversion.

OBI_DE:

Parameters that control the processing of OBI documents.

threshold: *float*, default: 50.0
The amount limit below which items are written off.
conversion_mode: *str*, default: ‘-layout -enc UTF-8 -nopgbrk’
Processing options passed to the PDF text extractor.
accounting_map: str, default: ‘obi_de’
Name of the file that contains the accounting map from which customer accounts are assigned to branches.
excel_name: *str*, default: $docnum$_Avis_$suppnum$_$docdate$
Name of the Excel file to which the extracted data is written.
The $docnum$ placeholder is replaced by the payment advice number. The $docdate$ placeholder is replaced by the payment advice date. The $suppnum$ placeholder is replaced by the supplier ID stated in the payment advice.
date_format: *str*, default: '%d%b%Y'
String that controls the resulting format of the document date to be used in the ‘excel_name’ parameter.
layout: *list[str]*, default: [
Branch_Number, Document_Number, Document_Type, Case_ID, On_Account_Text, Gross_Amount_(ABS), Gross_Amount, Deduction, Net_Amount, Discount, Provision_Discount, Tax_Code, Debitor, GL_Account]
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.

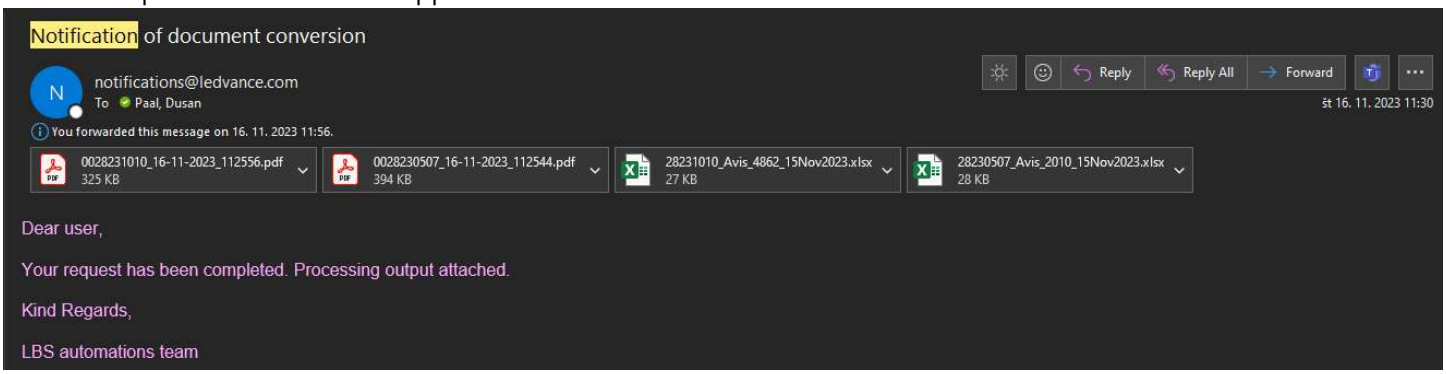
MARKANT_DE:

Parameters that control the processing of OBI documents.

threshold: *float*, default: 50.0
The amount limit below which items are written off.
conversion_mode: *str*, default: ‘-table -fixed 4 -enc UTF-8 -nopgbrk’
Processing options passed to the PDF text extractor.
accounting_map: *str*, default: ‘markant_de’
excel_name: *str*, default: $docnum$_Avis_$doctype$_$docdate$
The $docnum$ placeholder is replaced by the payment advice number. The $docdate$ placeholder is replaced by the payment advice date. The $doctype $ placeholder is replaced by the German name of the payment advice type.
date_format: *str*, default: '%d%b%Y'
String that controls the resulting format of the document date to be used in the ‘excel_name’ parameter.
layout: *list[str]*, default: [
Document_Number, Document_Date, Document_Type, Archive_Number, Original_Document, Search_Key, Debitor, ILN, Case_ID, On_Account_Text, Gross_Amount, Gross_Amount_(ABS), Markant_SB_Condition, Customer_SB_Condition, Discount, DL_Condition, Net_Amount, Tax_Rate, Tax_Code]
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.

8. User manual

In the current version, the conversion of payment advices issued only by OBI Germany and Markant Germany is implemented. To convert a PDF payment advice to Excel, the user sends the document as an attachment via email with the subject "AR_Payment_Advice_Converter" to the address: lbs.robot@ledvance.com. The application performs data extraction and sends the Excel file and the original PDF file back to the sender's email address. Conversion of multiple documents per one email is also supported.



For **OBI Germany**, the resulting Excel table consists of the following columns:

- **Branch Number**: Number of the OBI branch to which the item refers.
- **Document Number**: Number of the accounting document originally issued by OBI
- **Document Type**: Type of the item:
  - *Debit*: A debit note amount to be posted to the customer account.
  - *Credit*: A credit amount to be posted to the customer account
  - *Invoice*: A paid invoice.
  - *Credit/Invoice*: Could refer either a credit note or an invoice. Used when the automation cannot unambiguously identify the document type.
  - *WriteOff Penalty*: Penalty-related debit or credit note to be written off (50.00 EUR threshold).
  - *WriteOff Others*: Debit or a credit note to be written off, for items where document category is other than penalty (50.00 EUR threshold).
- **Case ID**: Number of the related DMS case. All fields are blank by default.
- **On Account Text**: Text to be used when posting the payment to the respective customer/GL account. All fields are blank by default.
- **Gross Amount (ABS)**: Absolute value of the gross amount of the line item.
- **Gross Amount**: Gross amount of the item.
- **Gross Amount Deduction**: The total amount (Discount + Provision Discount amount) deducted from the gross amount of the item.
- **Net Amount**: Net amount of the item.
- **Discount**: Discount amount taken by the customer (3% of the item gross amount).
- **Provision Discount**: Provision discount amount taken by the customer (2% of the item gross amount).
- **Tax Code**: Tax code used for posting the item to the respective customer/GL account.
  For items issued by an OBI branch other than 850, the tax codes are identified automatically. Where discount and provision discount values are equal 0.00 the tax code "A0" is used. Where the discount and provision discount values are not equal 0,00, the tax code "C3" is used. For items issued by the OBI branch 850 (bonus-related items), the "check" value is used, since the automation cannot identify the tax code from the data. Rather than that, the tax code must be determined manually by the accountant.
- **Debitor**: Number of the customer account related to the item.

The account number is identified from the accounting map stored in "obi_de.json" file by searching the map keys for a match in the supplier number and the branch number. If a match is found, then the respective value is placed into the field, otherwise it remains blank.

- **GL Account**: Number of the general ledger account for writing off the item amount.

For **Markant Germany**, the resulting Excel table consists of the following columns (may vary by Journal type):

- **Document Number**: (Journal 10/20) Number of the accounting document originally issued by Markant.
- **Document Date**: (Journal 10/20) Date of the item.
- **Document Type**: (Journal 10/20) Type of the item:
  - *Invoice*: (Journal 10) A paid invoice.
  - *Debit*: (Journal 20) A debit note amount to be posted to the customer account.
  - *Credit*: (Journal 20) A credit amount to be posted to the customer account.
  - *WriteOff*: (Journal 20) Debit or a credit note to be written off (50.00 EUR threshold).
- **Archive Number**: (Journal 10/20) Number of the archive record in the Markant database.
- **Original Document**: (Journal 10/20) Number of the original document to which the item relates.
- **Search Key**: (Journal 10/20) The key to use for searching the record in DMS by Title.
- **Debitor**: (Journal 10/20) Number of the customer account related to the item.

  The account number is identified from the accounting map stored in "markant_de.json" file by searching the map keys for a match in the item ILN number. If a match is found, then the respective value is placed into the field, otherwise it remains blank.
- **ILN:** (Journal 10/20) International location number of the customer or branch that booked the item.
- **Case ID**: (Journal 10/20) Number of the related DMS case. All fields are blank by default.
- **On Account Text**: (Journal 20) Text to be used when posting the payment to the respective customer/GL account. All fields are blank by default.
- **Gross Amount**: (Journal 10/20) Value of the item gross amount.
- **Gross Amount (ABS)**: (Journal 20) Absolute value of the item gross amount.
- **Markant SB Condition**: (Journal 10/20) Markant "Sofortbonus" condition amount.
- **Customer SB Condition**: (Journal 10/20) Customer "Sofortbonus" condition amount.
- **Discount**: (Journal 10/20) Discount amount taken by the customer from the item gross amount.
- **DL Condition:** (Journal 10/20) Services condition amount.
- **Net Amount**: (Journal 10/20) Net amount of the item.
- **Tax Rate**: (Journal 10/20) Tax rate in % used to calculate the item taxed amount.
- **Tax Code**: (Journal 10/20) Tax code used for posting the item to the respective customer/GL account:

  C3: where tax rate is 0%
  AA: where tax rate is 16%
  AB: where tax rate is 19%
  C6: where tax rate is 20%

9. Revision

Version	Date	Author	Description
1.0	20.11.2023	Dusan Paal	Initial version