Computer Science and Engineering 2025, 15(4): 102-106
DOI: 10.5923/j.computer.20251504.03

Automating SAP Document Processing from Emailed PDFs: Leveraging AI for Touchless Order Processing

Ajay Verma

MCA – SAP Technical Architect and AI Specialist, Dayton, Ohio, United States of America

Abstract Digital transformation in enterprise resource planning requires robust, automated handling of documents, including scanned PDFs that traditionally demanded extensive manual intervention. This paper presents an approach where SAP's inbound processing capabilities are seamlessly integrated with Python-based AI agents. The process involves extracting scanned PDF attachments from emails using SAP's class CL_EPM_ADS_INBOUND_LIST method PROCESS_INBOUND, converting the internal xstring representation to binary, binary is then transmitted to a Python AI agent, which utilizes advanced optical character recognition (OCR) and natural language processing (NLP) techniques to convert the binary data into human-readable text. Once processed, the resultant text is re-imported into SAP to automatically generate documents such as sales orders. This integration not only minimizes manual data entry but also improves processing speed and accuracy. This process doesn't require ADS (Adobe Document Services) configuration. Additionally, to maintain auditability and enable document retrieval, processed documents are stored on a content server using the SAP function module ARCHIVOBJECT_CREATE_FILE, ensuring that original attachments remain accessible for future reference. While the methodology is applicable to a wide range of document types (including purchase orders and financial documents), this paper illustrates the approach by focusing on the automated creation of sales orders from inbound email PDF attachments.

Keywords SAP, Python, AI Agent, Scanned PDFs, Inbound Email Processing, Sales Order Automation, OCR, NLP

1. Introduction

In today's fast-paced business environment, enterprises are continually seeking ways to reduce manual work and enhance process efficiencies. One challenge is handling scanned documents. Even within robust systems like SAP, many processes involve scanning physical documents, such as invoices, contracts, or purchase orders, then ingesting them manually. Labor-intensive workflows lead to delays, errors, and inflated costs.

This paper explores a practical integration between SAP and a Python-based AI agent that transforms scanned PDF documents into actionable data. By leveraging SAP's inbound email attachment processing capabilities and coupling them with AI-driven text extraction, companies can accelerate document processing workflows to drive efficiency in operations like sales order creation. The proposed system creates a pipeline that extracts the scanned PDF attachment from inbound email, converts it for AI consumption, interprets its contents, and reintegrates the recognizable text into SAP, to reduce manual intervention.

* Corresponding author:
ajaverma@yahoo.com (Ajay Verma)
Received: May 30, 2025; Accepted: Jun. 22, 2025; Published: Jul. 4, 2025
Published online at http://journal.sapub.org/computer

The real innovation here lies in bridging the gap between legacy process steps and modern AI capabilities. The following sections detail our research into the system design, implementation, and operational benefits of this integration.

2. Background

2.1. Process Challenges

Many organizations face persistent issues with manual document posting for example manual order creation in SAP, particularly when dealing with variable-quality PDFs:

- **Manual Data Entry:** This introduces avoidable human errors and takes valuable staff time.
- **Inconsistent Document Quality:** Scanned documents are often noisy or skewed, complicating automated extraction.
- **Delays and Inefficiencies:** Processing bottlenecks can affect revenue recognition and customer service.

2.2. Value Proposition of Automation

By automating this process, businesses stand to gain:

- **Accuracy:** Automated extraction ensures consistency, reducing the risk of costly mistakes.
- **Speed:** SAP orders are processed and fulfilled more quickly.
- **Standardization:** End-to-end processing becomes uniform, supporting operational scalability.
- **Structured Error Handling:** Entries with missing or ambiguous data are flagged for manual attention, ensuring nothing slips through the cracks.

Our solution brings together SAP's enterprise-strength email handling with a modern Python extraction service, supporting a streamlined order-to-cash cycle.

3. System Architecture and Workflow

The architecture integrates SAP's email processing and a Python-based document analysis service using secure REST APIs. The end-to-end process comprises:

1. **Email Reception & Attachment Extraction:** SAP monitors a designated inbox and extracts PDF attachments using standard interfaces.
2. **Transmission to Python Service:** PDF data is securely posted to an external Python service.
3. **Document Analysis:** The Python service decodes, applies OCR/NLP, and returns extracted DATA in readable format.
4. **Validation & Order Creation:** SAP validates the returned data, either creating the order or logging an error for follow-up.
5. **No Use of ADS:** The process is fully SAP native, and Python based avoiding additional middleware dependencies.

3.1. High-Level Business Process Flow

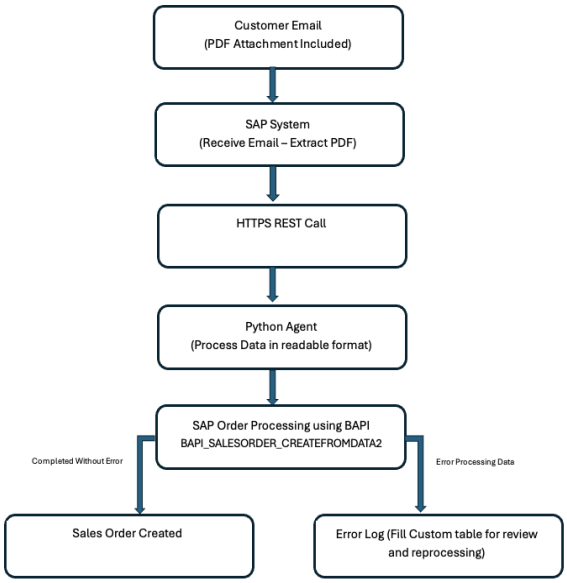

Figure 1. Business Process Flow

4. Implementation Details

Note: Below code is for reference purpose – it is not complete code.

4.1. SAP Implementation [1] [2]

Setup SMTP inbound message, inbound processing using T-Code SCOT

Copy class CL_EPM_ADS_INBOUND_LIST to ZZ_ EPM_ADS_INBOUND_LIST and assign to communication type "Internet Mail" and document class "PDF"

4.1.1. Email Monitoring and Extraction and Middleware Integration

Implement Method CREATE_INSTANCE

```
DATA: cl_ref type ref to zz_emp_ads_inbound_list.
create object cl_ref.
ro_ref = cl_ref.
```

Implement method PROCESS_INBOUND to read email and extract attached PDF data:

```
METHOD if_inbound_exist~process_inbound.

  extract_pdf_attachement(
        EXPORTING
              io_sreq = io_sreq
        IMPORTING
              ev_success   = lv_success
              ev_pdf       = lv_pdf
              ev_filename = lv_filename ).
```

```
  CALL FUNCTION 'SCMS_XSTRING_TO_BINARY'
            EXPORTING
                      buffer                = lv_pdf
            IMPORTING
                      binary_tab            = pdf_binary



  gv_url = 'http://python-server-ip-port/process_pdf_data'.

  CALL_METHOD cl_http_client=>create_by_url(
            EXPORTING
                      url           = gv_url
            IMPORTING
                      client        = lv_http_client ).

  lv_http_client->request->set_header_field (
            name = 'content-type' value 'applicatio\pdf' ).

  lv_http_client->request->set_cdata( lv_pdf ).
  lv_http_client->send( ).


  Wait up to 5 seconds.
  DATA(lv_json) = lv_http_client->response->get_data().

  Perform create_sales_order using lv_json.

ENDMETHOD.
```

4.1.2. Sales Order Creation and Error Handling

After receiving the JSON response, use the data to create sales order:

```
FORM create_sales_order using lv_json.

Fill Header DATA structure
Fill Item DATA internal table
Fill Partner DATA internal table
Fill Schedule line DATA internal table

CALL FUNCTION 'BAPI_SALESORDER_CREATEFROMDAT2'
    EXPORTING
        ORDER_HEADER_IN     = header
        ORDER_HEADER_INX    = headerx
     IMPORTING
        SALESDOCUMENT       = sales_order
     TABLES
        return              = return
        ORDER_ITEMS_IN      = item
        order_items_inx     = itemx
        order_schedules_in  = schedules
        order_schedules_inx = schedulesx
        order_partners      = partner.


ENDFORM.
```

4.2. Python Service [2] [3]

A lightweight Python web service (Flask or FastAPI) is used for PDF processing:

```
from flask import Flask, request, jsonify
import binascii
import pytesseract
from PIL import Image
import io

app = Flask(__name__)

@app.route('/process_pdf', methods=['POST'])
def process_pdf():
    data = request.get_json()
    hex_data = data.get('hexData')
    if not hex_data:
        return jsonify({"error": "No hex data provided"}), 400

    try:
        binary_data = binascii.unhexlify(hex_data)
        image = Image.open(io.BytesIO(binary_data))
        processed_image = image.convert("L")
        extracted_text = pytesseract.image_to_string(processed_image)
        structured_data = parse_extracted_text(extracted_text)
        return jsonify(structured_data)
    except Exception as e:
        return jsonify({"error": str(e)}), 400

def parse_extracted_text(text):
    data = {}
    data['CustomerName']          = extract_field(text, 'Customer')
    data['PurchaseOrderNumber'] = extract_field(text, 'PO')
    data['ProductCode']           = extract_field(text, 'Product')
    data['Quantity']              = extract_field(text, 'Quantity')
    data['DeliveryDate']          = extract_field(text, 'Delivery Date')
    return data

def extract_field(text, keyword):
    # Placeholder logic: should be replaced by more robust parsing
    if keyword in text:
        return "Extracted_" + keyword
    return ""

if __name__ == '__main__':
    app.run(ssl_context='adhoc', port=5000)
```

Note: Actual field extraction would be more sophisticated in a production environment.

4.3. Design without ADS

This solution is intentionally designed without requiring SAP ADS, keeping the architecture lightweight and reducing operational overhead. SAP's built-in email and HTTP capabilities, paired with a dedicated external extraction service, ensure robust and maintainable integration.

5. Security, Testing, and Performance

5.1. Security

- **Encryption:** All data exchanges use HTTPS.
- **Access Controls:** Endpoints are protected via authentication tokens and SAP's standard authorization checks.

- **Logging:** Both SAP and Python services keep detailed logs for traceability.
- **Data Integrity:** Careful error handling and validation at each step protect against data corruption.

5.2. Testing

- **Unit Testing:** Individual functions are tested separately in both ABAP and Python.
- **Integration Testing:** The end-to-end flow is validated using sample documents.
- **Load and Fault Tolerance:** The solution is tested under high volume and with intentionally malformed data to ensure resilience.

5.3. Performance

- **Scalability:** Both SAP and the Python service can be scaled horizontally.
- **Caching:** The architecture supports concurrent requests and temporary caching if needed.

6. Challenges and Opportunities for Improvement

6.1. Existing Limitations

- **Document Quality:** Poor scan quality still presents challenges for OCR.
- **Flexible Field Extraction:** Ongoing enhancement is required for robust parsing of non-standard document layouts.
- **Processing Time:** Complex OCR/NLP steps may introduce some latency under heavy loads.

6.2. Future Directions

- **Advanced ML Models:** Introducing deep learning could boost accuracy for both OCR and field extraction.
- **Real-Time Processing:** Moving to event-driven processing for lower latency.
- **Wider Document Types:** Supporting handwritten or mobile-generated POs.
- **Proactive Correction:** Leveraging order history to suggest corrections for incomplete fields.

7. Conclusions

This paper outlines a practical and robust method for automating SAP sales order creation from emailed PDF documents, leveraging only standard SAP and Python integration techniques. The architecture is designed for real-world enterprise environments—delivering accuracy, speed, and operational efficiency without increasing system complexity. The proposed solution reduces manual workload, increases reliability, and lays a scalable foundation for future enhancements in order automation.

REFERENCES

[1] SAP Integration Best Practices. https://learning.sap.com/learning-journeys/getting-started-with-sap-integration-solution-advisory-methodology/defining-integration-best-practices_b3bc1a05-9dee-400d-9937-51acc95fee76, https://community.sap.com/t5/enterprise-resource-planning-blog-posts-by-members/save-process-incoming-e-mail-and-attachments-in-sap/ba-p/13556119.

[2] Practical Python for Business Data Processing. https://learnpython.com/blog/python-for-business/, https://stackoverflow.com/questions/45480280/convert-scanned-pdf-to-text-python.

[3] Automation with OCR & NLP – Journal of Intelligent Enterprise Systems. https://ijirt.org/publishedpaper/IJIRT175285_PAPER.pdf, https://stackoverflow.com/questions/45480280/convert-scanned-pdf-to-text-python.