Python_SPO_Framework

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 config Namespace Reference

Variables

- str EXCEL_FILE = "Framewrok and SPO Output.xlsx"
- str MAIN_FOLDER = "Main_spo_framework"
- str GROQ_MODEL = "llama-3.3-70b-versatile"
- str GEMINI_MODEL = "gemini-2.5-flash"
- str PROMPTS_FILE = "Prompts/prompts_spo_framework.json"
- str PROMPTS_TABLE = "Prompts/prompts_table.json"
- str WHISPERER_BASE = "https://llmwhisperer-api.us-central.unstract.com/api/v2"
- int TOP_K = 6
- int CHUNK_SIZE = 2000
- int OVERLAP = 200

3.1.1 Detailed Description

```
Configuration file for the SPO Framework project.

Contains constants used across the project, including file paths,
model names, and parameters for text chunking and retrieval.
```

3.1.2 Variable Documentation

3.1.2.1 CHUNK_SIZE

```
int config.CHUNK_SIZE = 2000
```

3.1.2.2 EXCEL_FILE

```
str config.EXCEL_FILE = "Framewrok and SPO Output.xlsx"
```

3.1.2.3 GEMINI_MODEL

```
str config.GEMINI_MODEL = "gemini-2.5-flash"
```

3.1.2.4 GROQ_MODEL

```
str config.GROQ_MODEL = "llama-3.3-70b-versatile"
```

3.1.2.5 MAIN_FOLDER

```
str config.MAIN_FOLDER = "Main_spo_framework"
```

3.1.2.6 OVERLAP

```
int config.OVERLAP = 200
```

3.1.2.7 PROMPTS_FILE

```
str config.PROMPTS_FILE = "Prompts/prompts_spo_framework.json"
```

3.1.2.8 PROMPTS_TABLE

```
str config.PROMPTS_TABLE = "Prompts/prompts_table.json"
```

3.1.2.9 TOP_K

```
int config.TOP_K = 6
```

3.1.2.10 WHISPERER_BASE

```
str config.WHISPERER_BASE = "https://llmwhisperer-api.us-central.unstract.com/api/v2"
```

3.2 extractor Namespace Reference

Functions

- List[str] extract_text_from_pdf (str path)
- List[str] chunk_text (str text, int chunk_size=2000, int overlap=200)
- List[Dict] extract_chunks_from_two_pdfs (str framework_pdf, str spo_pdf, int chunk_size=500, int overlap=200, str folder_name=None)

3.2.1 Detailed Description

```
extractor.py

This module provides utilities to extract and chunk textual data from PDF files,
specifically designed for SPO and Framework PDFs in the SPO-Framework-Extractor pipeline.

Functions:
- extract_text_from_pdf(pdf_path: str) -> List[str]
    Extracts raw text from each page of a PDF and returns a list of page texts.

- chunk_text(text: str, chunk_size: int = 2000, overlap: int = 200) -> List[str]
    Splits a single string into overlapping text chunks of specified size.

- extract_chunks_from_two_pdfs(framework_pdf: str, spo_pdf: str, chunk_size: int = 500,
                              overlap: int = 200, folder_name: str = None) -> List[Dict]
    Extracts and chunks text from two PDFs (framework and SPO) and returns a list
    of dictionaries containing chunk data, source, page number, chunk index, and folder.
```

3.2.2 Function Documentation

3.2.2.1 chunk_text()

```
List[str] extractor.chunk_text (
             str text,
             int  chunk_size = 2000,
             int  overlap = 200)
```

```
Split a string into overlapping text chunks.

Args:
    text (str): Input text to be chunked.
    chunk_size (int): Maximum number of characters per chunk. Default is 2000.
    overlap (int): Number of characters to overlap between consecutive chunks. Default is 200.

Returns:
    List[str]: List of text chunks. Returns an empty list if text is empty.
```

3.2.2.2 extract_chunks_from_two_pdfs()

```
List[Dict] extractor.extract_chunks_from_two_pdfs (
             str framework_pdf,
             str spo_pdf,
             int  chunk_size = 500,
             int  overlap = 200,
             str  folder_name = None)
```

```
Extract text from two PDFs (framework and SPO), chunk each page, and return structured chunks.

Args:
    framework_pdf (str): Path to the framework PDF.
    spo_pdf (str): Path to the SPO PDF.
    chunk_size (int): Maximum characters per chunk. Default is 500.
    overlap (int): Number of overlapping characters between chunks. Default is 200.
    folder_name (str, optional): Name of the folder/company associated with these PDFs.

Returns:
    List[Dict]: A list of dictionaries, each representing a chunk:
        {
            "chunk": str,          # Chunk text
            "source": str,         # "framework" or "spo"
            "page": int,           # Page number (1-indexed)
            "chunk_index": int,    # Index of chunk on this page (1-indexed)
            "folder": str or None  # Folder/company name
        }
```

3.2.2.3 extract_text_from_pdf()

```
List[str] extractor.extract_text_from_pdf (
             str path)
```

```
Extract text from a PDF file.

Args:
    path (str): Path to the PDF file.

Returns:
    List[str]: List of strings, one per page. Index 0 corresponds to page 1.
               If a page has no text, returns an empty string for that page.
```

3.3 main Namespace Reference

Functions

- find_pdf_pair (str folder_path)
- main ()
- main_table ()

3.3.1 Detailed Description

```
main.py

This is the main pipeline file for the SPO-Framework-Extractor project.

It orchestrates the processing of PDFs in the 'Main_framework' folder by:
1. Identifying framework and SPO PDF pairs in each subfolder.
2. Extracting text chunks from the PDFs using `extractor.py`.
3. Parsing the chunks with LLMs using `parser.py`.
4. Writing structured output to Excel using `writer.py`.

Additionally, it handles the tabular pipeline:
1. Extract tables from PDFs using `table_extractor.py`.
2. Parse tables using `table_parser.py`.
3. Write tabular output to Excel using `table_writer.py`.

Functions:
- find_pdf_pair(folder_path: str) -> Tuple[str, str]: Identifies framework and SPO PDFs in a folder.
- main(): Runs the textual data pipeline for all subfolders.
- main_table(): Runs the tabular data pipeline for all subfolders.
```

3.3.2 Function Documentation

3.3.2.1 find_pdf_pair()

```
main.find_pdf_pair (
             str folder_path)
```

```
Identify and return the framework and SPO PDF pair in a folder.

Framework PDF:
    - Must contain 'framework' but NOT 'spo', 'second', or 'second-party-opinion'.
SPO PDF:
    - Must contain 'spo', 'spoc', 'second', or 'second-party-opinion'.

If the folder contains exactly two PDFs and no strict match is found, the two PDFs
are assumed to be the framework and SPO pair.

Args:
    folder_path (str): Path to the folder containing PDF files.

Returns:
    Tuple[str, str]: Paths to the framework PDF and SPO PDF respectively.
                     Returns (None, None) if no valid pair is found.
```

3.3.2.2 main()

```
main.main ()
```

```
Run the textual data pipeline for all subfolders in MAIN_FOLDER.

Workflow:
1. Iterate over subfolders.
2. Find framework and SPO PDF pair.
3. Extract text chunks using extractor.
4. Parse chunks using LLM (Gemini or Groq).
5. Write structured results to Excel immediately.
```

3.3.2.3 main_table()

```
main.main_table ()
```

```
Run the tabular data pipeline for all subfolders in MAIN_FOLDER.

Workflow:
1. Extract tables from PDFs using table_extractor.
2. Parse tables using table_parser.
3. Write parsed tabular data to Excel using table_writer.
```

3.4 parser Namespace Reference

Functions

- Dict build_tfidf_index (List[Dict] chunks)
- List[int] retrieve_top_k (str query, Dict index, int k=5)
- str assemble_context (List[Dict] chunks, List[int] top_indices)
- Dict call_groq (str model, List[Dict] messages, float temperature=0.0, int max_retries=3)
- List[Dict] parse_with_llm (List[Dict] chunks, str prompts_path, str groq_model, int top_k=5)
- Dict call_gemini (str model_gemini, List[Dict] messages, float temperature=0.0, int max_retries=3)
- List[Dict] parse_with_llm_gemini (List[Dict] chunks, str prompts_path, str gemini_model, int top_k=5)

3.4.1 Detailed Description

```
parser.py

This module handles the parsing of extracted PDF text chunks using either Groq or Gemini LLMs.
It supports:

1. Building a TF-IDF index over chunks for retrieval of relevant context.
2. Retrieving top-k relevant chunks for a given prompt.
3. Assembling a context block for LLM input.
4. Calling Groq or Gemini models to extract structured JSON based on prompts.
5. Parsing and returning the LLM outputs as a list of structured JSON objects.
```

3.4.2 Function Documentation

3.4.2.1 assemble_context()

```
str parser.assemble_context (
             List[Dict] chunks,
             List[int] top_indices)
```

```
Assemble a human-readable context block from selected chunks.

Args:
    chunks (List[Dict]): List of chunk dictionaries.
    top_indices (List[int]): List of indices of chunks to include.

Returns:
    str: Concatenated context string with source, page, and chunk metadata.
```

3.4.2.2 build_tfidf_index()

```
Dict parser.build_tfidf_index (
             List[Dict] chunks)
```

```
Build a TF-IDF vectorizer and matrix for a list of text chunks.

Args:
    chunks (List[Dict]): List of dicts, each containing a 'chunk' key.

Returns:
    Dict: {
        "vectorizer": TfidfVectorizer object,
        "matrix": TF-IDF feature matrix,
        "texts": List[str] of chunk texts
    }
```

3.4.2.3 call_gemini()

```
Dict parser.call_gemini (
             str model_gemini,
             List[Dict] messages,
             float  temperature = 0.0,
             int  max_retries = 3)
```

```
Call Gemini chat model with retries.
```

3.4.2.4 call_groq()

```
Dict parser.call_groq (
             str model,
             List[Dict] messages,
             float  temperature = 0.0,
             int  max_retries = 3)
```

```
Call Groq chat model with retries.

Args:
    model (str): Groq model name.
    messages (List[Dict]): List of messages with "role" and "content".
    temperature (float): Sampling temperature.
    max_retries (int): Number of retry attempts if call fails.

Returns:
    Dict: Raw response from Groq API.
```

3.4.2.5 parse_with_llm()

```
List[Dict] parser.parse_with_llm (
             List[Dict] chunks,
             str prompts_path,
             str groq_model,
             int  top_k = 5)
```

```
Parse chunks using Groq LLM based on provided prompts.
```

3.4.2.6 parse_with_llm_gemini()

```
List[Dict] parser.parse_with_llm_gemini (
             List[Dict] chunks,
             str prompts_path,
             str gemini_model,
             int  top_k = 5)
```

```
Parse chunks using Gemini LLM based on provided prompts.
```

3.4.2.7 retrieve_top_k()

```
List[int] parser.retrieve_top_k (
             str query,
             Dict index,
             int  k = 5)
```

```
Retrieve indices of top-k most similar chunks to the query.

Args:
    query (str): Query string.
    index (Dict): TF-IDF index from `build_tfidf_index`.
    k (int): Number of top results to return.

Returns:
    List[int]: List of top-k indices into index['texts'] with positive similarity.
```

3.5 table_extractor Namespace Reference

Functions

- find_framework_and_spo_pdfs (folder_path)
- get_pages_with_tables_pdfplumber (pdf_path)
- assemble_pages_with_pypdf (src_pdf_path, page_indices, writer=None)
- create_label_page_bytes (text)
- write_temp_merged_pdf (framework_pdf, spo_pdf, tmp_suffix=".pdf")
- call_whisperer_and_get_text (merged_pdf_path)
- process_subfolders_in_memory (root_folder)

Variables

- ROOT_FOLDER = MAIN_FOLDER
- WHISPERER_BASE = WHISPER_BASE
- WHISPERER_API_KEY = os.getenv("LLMWHISPERER_API_KEY")

3.5.1 Detailed Description

```
table_extractor.py

Extracts table data from SPO and framework PDFs per company:
- Detects framework and SPO PDFs in each company folder
- Identifies pages containing tables using pdfplumber
- Merges relevant pages into a temporary PDF
- Sends the merged PDF to LLM Whisperer for extraction
- Returns the extracted text per company
```

3.5.2 Function Documentation

3.5.2.1 assemble_pages_with_pypdf()

```
table_extractor.assemble_pages_with_pypdf (
              src_pdf_path,
              page_indices,
              writer = None)
```

```
Add selected pages from a source PDF to a PdfWriter object.

Args:
    src_pdf_path (str): Path to the source PDF.
    page_indices (List[int]): List of 0-based page indices to add.
    writer (PdfWriter, optional): Existing PdfWriter object. Creates new if None.

Returns:
    PdfWriter: Updated PdfWriter object containing selected pages.
```

3.5.2.2 call_whisperer_and_get_text()

```
table_extractor.call_whisperer_and_get_text (
              merged_pdf_path)
```

```
Send a PDF to LLM Whisperer and return the extracted text.

Args:
    merged_pdf_path (str): Path to the merged PDF.

Returns:
    str: Extracted text from the PDF.

Raises:
    RuntimeError: If Whisperer does not return a valid whisper_hash.
```

3.5.2.3 create_label_page_bytes()

```
table_extractor.create_label_page_bytes (
              text)
```

```
Create a single-page PDF as bytes containing a centered label text.

Args:
    text (str): Label text to place on the PDF page.

Returns:
    bytes: PDF file content in bytes.
```

3.5.2.4 find_framework_and_spo_pdfs()

```
table_extractor.find_framework_and_spo_pdfs (
              folder_path)
```

```
Return the framework and SPO PDF paths from a given folder.

Args:
    folder_path (str): Path to the company’s folder.

Returns:
    Tuple[str | None, str | None]: (framework_pdf_path, spo_pdf_path). None if not found.
```

3.5.2.5 get_pages_with_tables_pdfplumber()

```
table_extractor.get_pages_with_tables_pdfplumber (
              pdf_path)
```

```
Return the 0-based page indices of a PDF that contain tables.

Args:
    pdf_path (str): Path to the PDF file.

Returns:
    List[int]: List of page indices containing tables.
```

3.5.2.6 process_subfolders_in_memory()

```
table_extractor.process_subfolders_in_memory (
              root_folder)
```

```
Process all company subfolders in memory:
- Find framework and SPO PDFs
- Extract table pages
- Merge pages into temporary PDF
- Send to LLM Whisperer
- Return extracted text per company

Args:
    root_folder (str): Path to the root folder containing company subfolders.

Returns:
    Dict[str, str]: Mapping of company name to extracted text.
```

3.5.2.7 write_temp_merged_pdf()

```
table_extractor.write_temp_merged_pdf (
              framework_pdf,
              spo_pdf,
              tmp_suffix = ".pdf")
```

```
Create a temporary merged PDF containing only pages with tables from framework and SPO PDFs.

Args:
    framework_pdf (str | None): Path to the framework PDF.
    spo_pdf (str | None): Path to the SPO PDF.
    tmp_suffix (str): Suffix for the temporary file (default ".pdf").

Returns:
    str | None: Path to the temporary merged PDF, or None if no pages were added.
```

3.5.3 Variable Documentation

3.5.3.1 ROOT_FOLDER

```
table_extractor.ROOT_FOLDER = MAIN_FOLDER
```

3.5.3.2 WHISPERER_API_KEY

```
table_extractor.WHISPERER_API_KEY = os.getenv("LLMWHISPERER_API_KEY")
```

3.5.3.3 WHISPERER_BASE

```
table_extractor.WHISPERER_BASE = WHISPER_BASE
```

3.6 table_parser Namespace Reference

Functions

- dict parser_for_table (str extracted_text, str prompt_json_path)

Variables

- LLM_API_KEY = os.getenv("GROQ_API_KEY")
- MODEL_NAME = GROQ_MODEL
- PROMPT_JSON_PATH = PROMPTS_TABLE

3.6.1 Detailed Description

```
table_parser.py

Parses extracted table text per company using a Large Language Model (LLM):
- Reads extracted text from memory or cached text files
- Loads instructions and JSON schema from Prompts/prompts_table.json
- Sends extracted text + instructions to the LLM
- Returns structured output as a Python dictionary, ensuring valid JSON even if LLM output is messy
```

3.6.2 Function Documentation

3.6.2.1 parser_for_table()

```
dict table_parser.parser_for_table (
             str extracted_text,
             str prompt_json_path)
```

```
Use LLM to extract structured info from extracted text according to the prompt JSON.
Returns a Python dictionary, ensuring valid JSON even if the LLM output is messy.

:param extracted_text: str, text extracted from table PDFs for a company
:param prompt_json_path: path to the JSON file containing task_description and output_json_structure
:param client: LLM client object
:param model_name: model name for the LLM
:return: dict, parsed JSON (fallback to {"_raw": <LLM output>} if parsing fails)
```

3.6.3 Variable Documentation

3.6.3.1 LLM_API_KEY

```
table_parser.LLM_API_KEY = os.getenv("GROQ_API_KEY")
```

3.6.3.2 MODEL_NAME

```
table_parser.MODEL_NAME = GROQ_MODEL
```

3.6.3.3 PROMPT_JSON_PATH

```
table_parser.PROMPT_JSON_PATH = PROMPTS_TABLE
```

3.7 table_writer Namespace Reference

Functions

- writer_to_excel_table (dict answer, str EXCEL_FILE)

3.7.1 Detailed Description

```
table_writer.py

Writes structured table extraction results from JSON into an Excel file.
- Creates or ensures existence of two sheets:
  1. 'Eligibility+EU Tax' -- detailed criteria per Use of Proceeds
  2. 'SDG' -- summary of SDGs per Use of Proceeds
- Handles appending to existing sheets safely, avoiding overwrite issues.
```

3.7.2 Function Documentation

3.7.2.1 writer_to_excel_table()

```
table_writer.writer_to_excel_table (
             dict answer,
             str EXCEL_FILE)
```

```
Writes structured data from a parsed JSON (`answer`) into an Excel file.

Args:
    answer (dict): JSON output containing 'Use_of_Proceeds', SDGs, and Eligibility Criteria.
    EXCEL_FILE (str): Path to the Excel file where data should be written.

Behavior:
    - Creates the Excel file if it does not exist.
    - Ensures sheets 'Eligibility+EU Tax' and 'SDG' exist.
    - Appends new data to the sheets without overwriting existing rows.
    - Automatically generates a new Framework ID (F001, F002, ...).
```

3.8 writer Namespace Reference

Functions

- str _get_next_framework_id (ws)
- Workbook _init_workbook ()
- None write_to_excel (Dict json_data, str run_for)

3.8.1 Detailed Description

```
writer.py

Handles writing structured framework and SPO data into an Excel workbook.
- Initializes workbook with required sheets if missing.
- Generates unique Framework IDs.
- Supports writing both 'framework' and 'spo' JSON data.
- Updates relevant sheets including:
  * Framework Overview
  * Governance
  * SPO Summary
```

3.8.2 Function Documentation

3.8.2.1 _get_next_framework_id()

```
str writer._get_next_framework_id (
              ws)  [protected]
```

```
Generate the next Framework ID in sequence (F001, F002, ...).

Args:
    ws (openpyxl.worksheet.worksheet.Worksheet): Worksheet containing existing frameworks.

Returns:
    str: Next Framework ID.
```

3.8.2.2 _init_workbook()

```
Workbook writer._init_workbook ()  [protected]
```

```
Initialize the Excel workbook if it does not exist, creating required sheets.

Returns:
    openpyxl.workbook.workbook.Workbook: The loaded or newly created workbook.
```

3.8.2.3 write_to_excel()

```
None writer.write_to_excel (
             Dict json_data,
             str run_for)
```

```
Write extracted JSON data into the Excel workbook.

Args:
    json_data (dict): Structured data for a framework or SPO.
    run_for (str): Either 'framework' or 'spo' to indicate type of data.

Behavior:
    - For 'framework': generates a new Framework ID, writes Framework Overview and Governance.
    - For 'spo': updates last framework row, writes SPO Summary.
    - Saves workbook after writing.
```

Chapter 4

File Documentation

4.1 config.py File Reference

Namespaces

- namespace config

Variables

- str config.EXCEL_FILE = "Framewrok and SPO Output.xlsx"
- str config.MAIN_FOLDER = "Main_spo_framework"
- str config.GROQ_MODEL = "llama-3.3-70b-versatile"
- str config.GEMINI_MODEL = "gemini-2.5-flash"
- str config.PROMPTS_FILE = "Prompts/prompts_spo_framework.json"
- str config.PROMPTS_TABLE = "Prompts/prompts_table.json"
- str config.WHISPERER_BASE = "https://llmwhisperer-api.us-central.unstract.com/api/v2"
- int config.TOP_K = 6
- int config.CHUNK_SIZE = 2000
- int config.OVERLAP = 200

4.2 extractor.py File Reference

Namespaces

- namespace extractor

Functions

- List[str] extractor.extract_text_from_pdf (str path)
- List[str] extractor.chunk_text (str text, int chunk_size=2000, int overlap=200)
- List[Dict] extractor.extract_chunks_from_two_pdfs (str framework_pdf, str spo_pdf, int chunk_size=500, int overlap=200, str folder_name=None)

4.3 main.py File Reference

Namespaces

- namespace main

Functions

- main.find_pdf_pair (str folder_path)
- main.main ()
- main.main_table ()

4.4 parser.py File Reference

Namespaces

- namespace parser

Functions

- Dict parser.build_tfidf_index (List[Dict] chunks)
- List[int] parser.retrieve_top_k (str query, Dict index, int k=5)
- str parser.assemble_context (List[Dict] chunks, List[int] top_indices)
- Dict parser.call_groq (str model, List[Dict] messages, float temperature=0.0, int max_retries=3)
- List[Dict] parser.parse_with_llm (List[Dict] chunks, str prompts_path, str groq_model, int top_k=5)
- Dict parser.call_gemini (str model_gemini, List[Dict] messages, float temperature=0.0, int max_retries=3)
- List[Dict] parser.parse_with_llm_gemini (List[Dict] chunks, str prompts_path, str gemini_model, int top_k=5)

4.5 table_extractor.py File Reference

Namespaces

- namespace table_extractor

Functions

- table_extractor.find_framework_and_spo_pdfs (folder_path)
- table_extractor.get_pages_with_tables_pdfplumber (pdf_path)
- table_extractor.assemble_pages_with_pypdf (src_pdf_path, page_indices, writer=None)
- table_extractor.create_label_page_bytes (text)
- table_extractor.write_temp_merged_pdf (framework_pdf, spo_pdf, tmp_suffix=".pdf")
- table_extractor.call_whisperer_and_get_text (merged_pdf_path)
- table_extractor.process_subfolders_in_memory (root_folder)

Variables

- table_extractor.ROOT_FOLDER = MAIN_FOLDER
- table_extractor.WHISPERER_BASE = WHISPER_BASE
- table_extractor.WHISPERER_API_KEY = os.getenv("LLMWHISPERER_API_KEY")

4.6 table_parser.py File Reference

Namespaces

- namespace table_parser

Functions

- dict table_parser.parser_for_table (str extracted_text, str prompt_json_path)

Variables

- table_parser.LLM_API_KEY = os.getenv("GROQ_API_KEY")
- table_parser.MODEL_NAME = GROQ_MODEL
- table_parser.PROMPT_JSON_PATH = PROMPTS_TABLE

4.7 table_writer.py File Reference

Namespaces

- namespace table_writer

Functions

- table_writer.writer_to_excel_table (dict answer, str EXCEL_FILE)

4.8 writer.py File Reference

Namespaces

- namespace writer

Functions

- str writer._get_next_framework_id (ws)
- Workbook writer._init_workbook ()
- None writer.write_to_excel (Dict json_data, str run_for)