Documentation for parser_main.py

Overview

The parser_main.py script is designed to process SEC filings data stored locally. It identifies all company folders, processes specific filing types for each company, and stores the extracted data in a structured manner. The script leverages concurrent processing to handle multiple filings efficiently.

Key Components

1. Data Class: EntityData

- Purpose: Represents the metadata and attributes of a company entity extracted from JSON data.
- Attributes:
    - Primary Fields: cik, entityType, sic, sicDescription, ownerOrg, name, exchanges, ein, description, website, investorWebsite, category, fiscalYearEnd, stateOfIncorporation, stateOfIncorporationDescription, addresses, phone, formerNames, filings.
    - Computed Properties:
        - num_of_form: Returns the number of former names.
        - total_num_of_filings: Returns the total number of filings.
- Methods:
    - __post_init__: Initializes mutable default values for lists and dictionaries.

2. Global Variables

- root_path: The base directory where the data is stored.
- company_folder_names: A list to store the names of company folders.
- filing_types: A list of filing types to process (e.g., 8-K, 10-K).
- COMMON_DATA: A list of common data fields extracted from JSON.

3. One-Timer Functions

- company_folders(root_path):
    - Purpose: Retrieves all folder names (company names) from the root directory.
    - Returns: A list of folder names or None if no folders are found.
- create_preprocessed_folders(folder_names, root_path):
    - Purpose: Creates a preprocessed folder for each company to store processed data.
    - Behavior: Skips creation if no folders are found.

4. File Processing Functions

- process_text_files(root_path, ticker, filing_type, loader):
    - Purpose: Processes text files for a specific company and filing type.
    - Steps:
        1. Checks if the raw folder path exists.
        2. Collects JSON metadata using JsonDataCollector.
        3. Creates a preprocessed folder for the filing type.
        4. Processes each text file, extracting and storing data using companies_main.
    - Error Handling: Logs errors and warnings for missing files or processing failures.

5. Main Function

- main(root_path, company_folder_names, processed_folder_created, filing_types):
    - Purpose: Orchestrates the entire process.
    - Steps:
        1. Initializes the Loader object.
        2. Retrieves company folder names if not already provided.
        3. Creates preprocessed folders if not already created.
        4. Processes each filing type for each company using process_text_files.
    - Error Handling: Ensures the Loader object is properly closed after processing.

Usage

1. Set the root_path:
    - Update the root_path variable to point to the directory containing the company folders.
2. Run the Script:
    - Execute the script by calling the main function:
    - python
    - Copy
    - main(root_path, company_folder_names, processed_folder_created, filing_types)
3. Output:
    - The script will create a preprocessed folder for each company, containing subfolders for each filing type.
    - Processed data will be stored in these folders.

Dependencies

- os: For directory and file operations.
- logging: For logging information, warnings, and errors.
- concurrent.futures: For concurrent processing of filings.
- bs4.BeautifulSoup: For parsing HTML/XML content (if needed).
- ETL.Loader: For loading data into a database or other storage.

- parser.JsonDataCollector: For collecting JSON metadata from filings.
- companies.companies_main: For processing and storing extracted data.

Overview

The parser.py script is designed to parse and extract data from SEC (Securities and Exchange Commission) filings, specifically focusing on HTML documents. The script uses the BeautifulSoup library to parse HTML content and extract relevant information such as headers, document metadata, and text content. It also handles normalization and cleaning of the extracted text.

Classes

Parser

The Parser class is the main class that encapsulates all the functionality for parsing SEC filings. It contains methods for reading documents, extracting header data, processing document content, and normalizing text.

Methods

1. __init__(self)
    - Initializes the Parser class with empty dictionaries for storing document and filing data.
2. find_case_insensitive(self, soup, tag_name)
    - Description: Finds a tag in a case-insensitive manner within a BeautifulSoup object.
    - Parameters:
        - soup: The BeautifulSoup object to search within.
        - tag_name: The name of the tag to find.
    - Returns: The first tag found with the specified name, or None if no such tag exists.
3. read_doc(self, path, empty_file=False)
    - Description: Reads a document from the specified path and returns a BeautifulSoup object.
    - Parameters:
        - path: The file path to read from.
        - empty_file: A flag to indicate if the file is empty (default is False).
    - Returns: A tuple containing the BeautifulSoup object and a boolean indicating if the file was empty.
4. header_data_parser(self, soup)

    - Description: Parses the SEC header from the BeautifulSoup object and extracts the accession number.
    - Parameters:
        - soup: The BeautifulSoup object containing the SEC filing.
    - Returns: A tuple containing a dictionary with the header data and the accession number.
5. document_data(self, soup, styles)
    - Description: Extracts document data such as document ID, sequence, filename, description, and text content from the BeautifulSoup object. It also handles thematic breaks and page numbers.
    - Parameters:
        - soup: The BeautifulSoup object containing the SEC filing.
        - styles: A list of style conditions to identify thematic breaks.
    - Returns: A dictionary containing the extracted document data.
6. construct_master_dict(self, master_document_dict, header, accession_number)
    - Description: Constructs a master dictionary that organizes the extracted document data and header information.
    - Parameters:
        - master_document_dict: A dictionary containing the extracted document data.
        - header: A dictionary containing the header data.
        - accession_number: The accession number of the filing.
    - Returns: A dictionary containing the master filing data.
7. restore_windows_1252_characters(self, restore_string)
    - Description: Replaces C1 control characters in a Unicode string with the corresponding Windows-1252 characters.
    - Parameters:
        - restore_string: The string to be processed.
    - Returns: The processed string with restored characters.
8. extract_text_from_html(self, html_str)
    - Description: Extracts and normalizes text from an HTML string, specifically handling <TEXT> content.
    - Parameters:
        - html_str: The HTML string to extract text from.
    - Returns: A tuple containing the BeautifulSoup object and the extracted raw text.
9. normalize_text_content(self, raw_text)
    - Description: Cleans and normalizes raw text extracted from HTML.
    - Parameters:
        - raw_text: The raw text to be normalized.
    - Returns: The cleaned and normalized text.
10. process_document_data(self, filing_docs)
    - Description: Processes and normalizes text for each document entry in the filing_docs dictionary.
    - Parameters:

        - filing_docs: A dictionary containing the document data to be processed.
    - Returns: The processed and normalized document data.
11. normalize_filing_docs(self, filing_docs)
    - Description: Normalizes all documents in the filing_docs dictionary.
    - Parameters:
        - filing_docs: A dictionary containing the document data to be normalized.
    - Returns: The normalized document data.
12. parse_html(self, filing_docs)
    - Description: Parses the HTML content of the documents in the filing_docs dictionary and extracts the text.
    - Parameters:
        - filing_docs: A dictionary containing the document data to be parsed.
    - Returns: A string containing the accumulated extracted text.
13. extract_text(self, element, handle_tables=True)
    - Description: Recursively extracts clean, de-duplicated text from a BeautifulSoup element. Tables are handled separately if handle_tables is True.
    - Parameters:
        - element: The BeautifulSoup element to extract text from.
        - handle_tables: A flag to indicate whether to handle tables separately (default is True).
    - Returns: A list of unique text lines extracted from the element.
14. extract_table(self, table_element)
    - Description: Extracts text from a BeautifulSoup <table> element and formats it as a tab-separated string.
    - Parameters:
        - table_element: The BeautifulSoup <table> element to extract text from.
    - Returns: A string containing the tab-separated table data.

Notes

- The script is designed to handle various edge cases, such as empty files, missing tags, and different HTML parsers.
- The normalize_text_content method ensures that the extracted text is clean and free of unnecessary whitespace and control characters.
- The parse_html method handles the extraction of text from HTML content, including the removal of <ix:header> tags and the extraction of table data.

Dependencies

- re: Regular expression module for string manipulation.
- unicodedata: Module for Unicode normalization.
- bs4 (BeautifulSoup): Library for parsing HTML and XML documents.

Overview

The companies.py script is designed to process SEC filings for individual companies. It uses the functionality provided by the parser.py script to parse, extract, and normalize data from raw SEC filings. The processed data is then stored in a preprocessed folder under the respective filing type for each company. Additionally, the script collects and stores JSON data related to the company and its filings.

Dependencies

- parser.py: Contains the Parser class used for parsing and extracting data from SEC filings.
- os: Provides functions for interacting with the operating system, such as file and directory manipulation.
- json: Used for handling JSON data.

Functions

companies_main(raw_path, preprocessed_path, file_name, ticker, loader, company_id)

- Description: This function is the main entry point for processing SEC filings for a specific company. It reads the raw filing, parses the data, normalizes it, and stores the processed data in a preprocessed folder.
- Parameters:
    1. raw_path: The path to the raw SEC filing file.
    2. preprocessed_path: The path where the preprocessed data will be stored.
    3. file_name: The name of the filing (e.g., "10-K", "8-K").
    4. ticker: The ticker symbol of the company.
    5. loader: An object used to insert and store company and filing data.
    6. company_id: The unique identifier for the company.
- Workflow:
    1. Check if the preprocessed file already exists: If the file already exists, the function skips processing.
    2. Read the raw file: The function reads the raw SEC filing using the Parser class.
    3. Extract header data: The header data and accession number are extracted from the filing.
    4. Store header data: The header data is stored in a global dictionary (all_headers) for later use.
    5. Extract document data: The function extracts document data such as document ID, sequence, filename, description, and text content.
    6. Construct the master dictionary: The extracted data is organized into a master dictionary.
    7. Normalize the filing documents: The text content of the documents is normalized.

8. Store the preprocessed data: The normalized data is written to a file in the preprocessed folder.
9. Collect and store JSON data: The function collects additional JSON data related to the company and its filings and stores it using the loader object.

json_data_helper(json_data_collector)

- Description: This function is a helper function intended to collect JSON data. However, the implementation is incomplete.
- Parameters:
    - json_data_collector: An object used to collect JSON data.
- Returns: This function currently does not return any value as it is incomplete.

Global Variables

- styles: A list of style conditions used to identify thematic breaks in the document. These styles are passed to the document_data method in the Parser class.
- all_headers: A global dictionary that stores the header data for each company and filing type.

Notes

- The script is designed to handle multiple filings for different companies and store the processed data in an organized manner.
- The loader object is used to insert and store company and filing data. The implementation of this object is not provided in the script.
- The json_data_helper function is currently incomplete and does not perform any meaningful operations.

Function Return Values and Their Usage

parser_main.py

1. company_folders(root_path)
    - Returns: A list of folder names (company names) in the root_path.
    - Used in:
        - main(): The returned list is stored in company_folder_names and used to iterate over companies.
2. create_preprocessed_folders(folder_names, root_path)
    - Returns: None (creates folders on the filesystem).
    - Used in:
        - main(): Called to ensure preprocessed folders exist for each company.
3. process_text_files(root_path, ticker, filing_type, loader)
    - Returns: None (processes files and writes output to the filesystem).
    - Used in:
        - main(): Called for each company and filing type to process the raw filings.
4. main(root_path, company_folder_names, processed_folder_created, filing_types)
    - Returns: None (orchestrates the entire process).
    - Used in:
        - This is the entry point of the script, so it is not called by other functions.

---

companies.py

1. companies_main(raw_path, preprocessed_path, file_name, ticker, loader, company_id)
    - Returns: None (writes processed data to the filesystem).
    - Used in:
        - process_text_files(): Called to parse and process each filing for a company.

---

parser.py

1. Parser.read_doc(path, empty_file=False)
    - Returns: A tuple containing:
        - A BeautifulSoup object (soup).
        - A boolean (empty_file) indicating if the file was empty.
    - Used in:
        - companies_main(): The soup object is used to parse the filing, and empty_file is checked to skip empty files.
2. Parser.header_data_parser(soup)
    - Returns: A tuple containing:
        - A dictionary (header) with the SEC header data.
        - The accession number (accession_number).
    - Used in:

        - companies_main(): The header dictionary is stored in all_headers, and the accession_number is used to organize the data.
3. Parser.document_data(soup, styles)
    - Returns: A dictionary (master_document_dict) containing extracted document data (e.g., document ID, sequence, filename, description, text).
    - Used in:
        - companies_main(): The returned dictionary is passed to construct_master_dict().
4. Parser.construct_master_dict(master_document_dict, header, accession_number)
    - Returns: A dictionary (filing_dict) containing the organized filing data.
    - Used in:
        - companies_main(): The returned dictionary is used to access filing_documents.
5. Parser.normalize_filing_docs(filing_docs)
    - Returns: A dictionary (norm_data) containing the normalized document data.
    - Used in:
        - companies_main(): The returned dictionary is passed to parse_html().
6. Parser.parse_html(filing_docs)
    - Returns: A string (file_acumulated_data) containing the parsed and accumulated text from the filing.
    - Used in:
        - companies_main(): The returned string is written to the preprocessed folder.
7. Parser.extract_text(element, handle_tables=True)
    - Returns: A list of unique text lines extracted from the HTML element.
    - Used in:
        - parse_html(): Called to extract text from the HTML content.
8. Parser.extract_table(table_element)
    - Returns: A string containing tab-separated table data.
    - Used in:
        - extract_text(): Called to handle table elements within the HTML.

JsonDataCollector Class

1. JsonDataCollector.collect_data()
    - Returns: A dictionary (json_data) containing the collected JSON metadata.
    - Used in:
        - process_text_files(): The returned dictionary is used to create an EntityData object and store metadata.
2. JsonDataCollector.remove_filings()
    - Returns: None (modifies the json_data dictionary in place).
    - Used in:
        - collect_data(): Called to remove the filings key from the JSON data.

Data Flow Summary

1. parser_main.py
    - Calls company_folders() to get company names.
    - Calls create_preprocessed_folders() to set up folders.
    - Calls process_text_files() for each company and filing type.
2. process_text_files()
    - Calls companies_main() to parse the filing.
    - Calls JsonDataCollector.collect_data() to collect metadata.
3. companies_main()
    - Calls Parser.read_doc() to read the filing.
    - Calls Parser.header_data_parser() to extract header data.
    - Calls Parser.document_data() to extract document data.
    - Calls Parser.construct_master_dict() to organize the data.
    - Calls Parser.normalize_filing_docs() to normalize the text.
    - Calls Parser.parse_html() to extract and accumulate text.
4. JsonDataCollector.collect_data()
    - Collects JSON metadata and removes the filings key.