Data Science Capstone Project

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 BlazorApp Namespace Reference

Namespaces

- namespace Controllers
- namespace Hubs
- namespace Models

5.2 BlazorApp.Controllers Namespace Reference

Classes

- class MetricsController

5.3 BlazorApp.Hubs Namespace Reference

Classes

- class MetricsHub

5.4 BlazorApp.Models Namespace Reference

Classes

- class PRF1Metric
- class QAItem
- class QAMetric
- class ScalarMetric
- class SummaryData
- class SummaryMetrics

5.5 components Namespace Reference

Namespaces

- namespace book_conversion
- namespace connectors
- namespace corpus
- namespace document_storage
- namespace fact_storage
- namespace metrics
- namespace text_processing

5.6 components.book_conversion Namespace Reference

Classes

- class Book
- class BookFactory
- class BookStream
- class Chunk

  Lightweight container for a span of story text.
- class EPUBToTEI

  Converts EPUB files to XML format (TEI specification).
- class ParagraphStreamTEI

  Streams paragraphs from a TEI file as Chunk objects.
- class Story
- class StoryStreamAdapter

Variables

- nlp = spacy.blank("en")
- sentencizer = nlp.add_pipe("sentencizer")

5.6.1 Variable Documentation

5.6.1.1 nlp

```
nlp = spacy.blank("en")
```

5.6.1.2 sentencizer

```
sentencizer = nlp.add_pipe("sentencizer")
```

5.7 components.connectors Namespace Reference

Classes

- class Connector

  Abstract base class for external connectors.
- class DatabaseConnector

  Abstract base class for database engine connectors.
- class mysqlConnector

  A relational database connector configured for MySQL.
- class postgresConnector

  A relational database connector configured for PostgreSQL.
- class RelationalConnector

  Connector for relational databases (MySQL, PostgreSQL).

5.8 components.corpus Namespace Reference

Functions

- load_booksum ()
- to_df_booksum (ds)
- load_narrativeqa ()
- to_df_nqa (ds)
- normalize_title (t)
- merge_dataframes (df1, df2, suffix1, suffix2, key_columns)
- fuzzy_merge_titles (df1, df2, suffix1, suffix2, key="title", threshold=90, scorer=fuzz.token_sort_ratio)

  Perform a two-way fuzzy merge between two DataFrames on a text column (e.g., book titles).

Variables

- df_booksum = load_booksum()
- df_nqa = load_narrativeqa()
- df = fuzzy_merge_titles(df_booksum, df_nqa, "_booksum", "_nqa", key="title", threshold=70)
- index
- m = Metrics()

5.8.1 Function Documentation

5.8.1.1 fuzzy_merge_titles()

```
fuzzy_merge_titles (
              df1,
              df2,
              suffix1,
              suffix2,
              key = "title",
              threshold = 90,
              scorer = fuzz.token_sort_ratio )
```

Perform a two-way fuzzy merge between two DataFrames on a text column (e.g., book titles).

For each row in the left DataFrame, the function searches the right DataFrame for the most similar string in the specified key column using RapidFuzz. It returns a merged DataFrame containing the best matches above a similarity threshold.

Parameters

df1	The left-hand DataFrame containing a text column to match on.
df2	The right-hand DataFrame containing a text column to match against.
suffix1	The name of the left-hand column.
suffix2	The name of the right-hand column.
key	The name of the column containing the strings to compare (default: "title").
threshold	Minimum similarity score (0–100) required to consider a match valid. Defaults to 90.
scorer	A RapidFuzz scoring function such as `fuzz.token_sort_ratio` or `fuzz.token_set_ratio`.

Returns

A new pandas DataFrame containing the compared strings, score, and all other columns.

Note

This function performs a one-to-one best match per left row. To ensure only confident matches are kept, adjust the `threshold` parameter.

5.8.1.2 load_booksum()

```
load_booksum ( )
```

5.8.1.3 load_narrativeqa()

```
load_narrativeqa ( )
```

5.8.1.4 merge_dataframes()

```
merge_dataframes (
              df1,
              df2,
              suffix1,
              suffix2,
              key_columns )
```

5.8.1.5 normalize_title()

```
normalize_title (
              t )
```

5.8.1.6 to_df_booksum()

```
to_df_booksum (
              ds )
```

5.8.1.7 to_df_nqa()

```
to_df_nqa (
              ds )
```

5.8.2 Variable Documentation

5.8.2.1 df

```
df = fuzzy_merge_titles(df_booksum, df_nqa, "_booksum", "_nqa", key="title", threshold=70)
```

5.8.2.2 df_booksum

```
df_booksum = load_booksum()
```

5.8.2.3 df_nqa

```
df_nqa = load_narrativeqa()
```

5.8.2.4 index

```
index
```

5.8.2.5 m

```
m = Metrics()
```

5.9 components.document_storage Namespace Reference

Classes

- class DocumentConnector

  Connector for MongoDB (document database)

Functions

- MongoHandle mongo_handle (str host, str alias)

  Establish a temporary connection to MongoDB.
- DataFrame _flatten_recursive (DataFrame df)

  Explode all list columns and flatten dict columns until only scalars remain.
- str _sanitize_json (str text)

  Remove comments and other non-JSON content from a MongoDB query string.
- Dict[str, Any] _sanitize_document (Dict[str, Any] doc, Dict[str, Set[Type[Any]]] type_registry)

  Normalize document fields to consistent types for DataFrame construction.
- DataFrame _docs_to_df (List[Dict[str, Any]] docs, bool merge_unspecified=True)

  Convert raw MongoDB documents to a Pandas DataFrame.
- str _find_compatible_nested_key (Type[Any] value_type, Dict[str, Set[Type[Any]]] nested_schema, bool merge_unspecified)

  Find a nested column compatible with the given primitive type.

Variables

- MongoHandle = Generator["Database[Any]", None, None]

5.9.1 Function Documentation

5.9.1.1 _docs_to_df()

```
DataFrame _docs_to_df (
            List[Dict[str, Any]] docs,
            bool  merge_unspecified = True )  [protected]
```

Convert raw MongoDB documents to a Pandas DataFrame.

Handles schema inconsistencies by:

1. First pass: identify all nested column names and their types
2. Second pass: sanitize and wrap primitives using type-compatible nested columns
3. Flatten structures into final DataFrame

Parameters

docs	List of MongoDB documents to convert.
merge_unspecified	If True, merge primitives into type-compatible nested columns using aggressive type casting (int→float, bool→int→float). If False, keep as _unspecified_type columns.

Exceptions

Log.Failure	If parsing query results to JSON fails.

5.9.1.2 _find_compatible_nested_key()

```
str _find_compatible_nested_key (
            Type[Any] value_type,
            Dict[str, Set[Type[Any]]] nested_schema,
            bool merge_unspecified )  [protected]
```

Find a nested column compatible with the given primitive type.

Uses type compatibility hierarchy for aggressive merging: bool → int → float (numeric types) str (isolated, only matches str) Searches for exact match first, then compatible types.

Parameters

value_type	The type of the primitive value to map (e.g., str, int, float).
nested_schema	Dict mapping nested keys to sets of observed types.
merge_unspecified	Whether to attempt type-compatible merging.

Returns

The nested key name to use for wrapping the primitive.

5.9.1.3 _flatten_recursive()

```
DataFrame _flatten_recursive (
            DataFrame df )  [protected]
```

Explode all list columns and flatten dict columns until only scalars remain.

Recursive Process:

1. Find columns containing lists → explode to create new rows
2. Find columns containing dicts → normalize to create new columns
3. Repeat until no lists or dicts remain

Parameters

df	DataFrame with potentially nested structures.

Returns

Fully flattened DataFrame with only scalar values.

5.9.1.4 _sanitize_document()

```
Dict[str, Any] _sanitize_document (
            Dict[str, Any] doc,
            Dict[str, Set[Type[Any]]] type_registry )  [protected]
```

Normalize document fields to consistent types for DataFrame construction.

Converts all field values to lists and tracks type patterns.

- ObjectId → string
- Single value → [value]
- Mixed types tracked in type_registry for conflict resolution

Parameters

doc	MongoDB document to sanitize.
type_registry	Tracks observed types per field path (e.g., {"effects": {str, list}}).

Returns

Document with all fields as lists.

5.9.1.5 _sanitize_json()

```
str _sanitize_json (
            str text )  [protected]
```

Remove comments and other non-JSON content from a MongoDB query string.

Removes the following elements:

- Block comments /∗ ... ∗/
- Single-line comments //
- Half-line comments ... //
- Trailing commas before closing braces
- Newlines and whitespace Preserves bad text inside JSON string values.

Parameters

text	Raw text that may contain comments.

Returns

Cleaned text suitable for JSON parsing.

5.9.1.6 mongo_handle()

```
MongoHandle mongo_handle (
            str host,
            str alias )
```

Establish a temporary connection to MongoDB.

Parameters

host	A valid MongoDB connection string.
alias	A unique name for the usage of this connection.

Allows scoped access to the low-level PyMongo handle from MongoEngine. Usage: with mongo_↩ handle(host=self.connection_string, alias="create_db") as db: (your code here...) This will disconnect all connections on the alias once finished. Helpful when test_connection wants to call execute_query, but continue using its existing db handle after execute_query disconnects.

5.9.2 Variable Documentation

5.9.2.1 MongoHandle

```
MongoHandle = Generator["Database[Any]", None, None]
```

5.10 components.fact_storage Namespace Reference

Classes

- class GraphConnector

  Connector for Neo4j (graph database).

Functions

- Tuple[Optional[List[Tuple[Any,...]]], Optional[List[str]]] filter_valid (List[Tuple[Any,...]] results, List[str] meta, str db_name)

  Filter Cypher query results (nodes or relationships) by database context.

5.10.1 Function Documentation

5.10.1.1 filter_valid()

```
Tuple[Optional[List[Tuple[Any, ...]]], Optional[List[str]]] filter_valid (
            List[Tuple[Any, ...]] results,
            List[str] meta,
            str db_name )
```

Filter Cypher query results (nodes or relationships) by database context.

- Keeps entities where 'db' and 'kg' match the given names.
- Excludes dummy nodes (_init = true).
- Relationships are valid if they or either endpoint match.
- Removes meta columns with no valid entities.

Parameters

results	List of tuples from db.cypher_query().
meta	Column names corresponding to each result element.
db_name	Database name to match.

Returns

(filtered_results, filtered_meta) with only valid entities, or (None, None) if empty.

5.11 components.metrics Namespace Reference

Classes

- class Metrics

  Utility class for computing and posting evaluation metrics.

Functions

- Dict[str, Any] run_questeval (Dict[str, Any] chunk, ∗str qeval_task="summarization", bool use_cuda=False, bool use_question_weighter=True)

    Run QuestEval metric calculation.
- Dict[str, Any] run_bookscore (Dict[str, Any] chunk, ∗str model="gpt-3.5-turbo", int batch_size=10, bool use↩_v2=True)

    Run BooookScore metric for long-form summarization.
- str chunk_bookscore (str book_text, str book_title='book', int chunk_size=2048)

    Chunk a book into BooookScore segments.

5.11.1 Function Documentation

5.11.1.1 chunk_bookscore()

```
str chunk_bookscore (
             str book_text,
             str  book_title = 'book',
             int  chunk_size = 2048 )
```

Chunk a book into BooookScore segments.

Standardizes long-form input into chunks BooookScore can process. Creates a temporary directory and writes chunked pickle for later scoring. This step can be reused independently for multiple summaries.

Parameters

book_text	Full book text to be chunked.
book_title	Name or identifier for the book (default 'book').
chunk_size	Maximum chunk size for book text (default 2048).

Returns

Path to chunked pickle file containing BooookScore-ready segments.

Exceptions

RuntimeError	If BooookScore chunking fails.

5.11.1.2 run_bookscore()

```
Dict[str, Any] run_bookscore (
             Dict[str, Any] chunk,
             *str  model = "gpt-3.5-turbo",
             int  batch_size = 10,
             bool  use_v2 = True )
```

Run BooookScore metric for long-form summarization.

LLM-based coherence evaluation using BooookScore. Runs in CLI via subprocess. Handles full workflow: scoring summary, postprocessing. Can be run on a single chunk or entire book (if already chunked).

Parameters

chunk	MongoDB document containing: • summary: Generated summary (required) • text: Full or partial book text (required) • book_title: Book title for identification (optional, for pickling)
model	Model name (optional, default 'gpt-4')
batch_size	Sentences per batch for v2 (optional, default 10)
use_v2	Use batched evaluation (optional, default True)

Returns

Dict containing a score (range 0-1) and metadata for the provided summary. bookscore: Coherence score for one summary. annotations: True if a gold reference summary was provided. model_used: String describing the LLM model and API used.

Exceptions

KeyError	If required fields are missing from chunk.
RuntimeError	If subprocess execution fails.

5.11.1.3 run_questeval()

```
Dict[str, Any] run_questeval (
            Dict[str, Any] chunk,
            *str  qeval_task = "summarization",
            bool  use_cuda = False,
            bool  use_question_weighter = True )
```

Run QuestEval metric calculation.

Question-answering based evaluation. Generates questions from source/reference, and checks if answers can be found in the summary. For more parameters, see: https://github.com/ThomasScialom/Quest←Eval/blob/main/questeval/questeval_metric.py

Parameters

chunk	MongoDB document containing keys: • summary: Generated summary (required) • text: Source document text (required) • gold_summary: Reference summary (optional, filters for better questions)
qeval_task	Task performed by QuestEval (optional, default is summarization). Must be one of the following: generation / nlg, qa, dialogue, data2text, translation.
use_cuda	Run transformers with GPU enabled.
use_question_weighter	Make some questions more important based on relevancy.

Returns

Dict containing a score (range 0-1) and metadata for the provided summary. questeval_score: Overall semantic precision–recall score for one example (a Summary to evaluate, Source text, and Reference summary). has_reference: True if a gold reference summary was provided.

Exceptions

ImportError	If questeval package not installed.
KeyError	If required fields are missing from chunk.

5.12 components.text_processing Namespace Reference

Classes

- class LLMConnector

  Connector for prompting and returning LLM output (raw text/JSON) via LangChain.
- class RelationExtractor

Variables

- nlp = spacy.blank("en")
- sentencizer = nlp.add_pipe("sentencizer")

5.12.1 Variable Documentation

5.12.1.1 nlp

```
nlp = spacy.blank("en")
```

5.12.1.2 sentencizer

```
sentencizer = nlp.add_pipe("sentencizer")
```

5.13 src Namespace Reference

Namespaces

- namespace flask

  Generic Flask worker microservice for distributed task processing.
- namespace main
- namespace setup
- namespace util

5.14 src.flask Namespace Reference

Generic Flask worker microservice for distributed task processing.

Functions

- process_task (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] chunk_doc, str boss_url, Callable[[Dict[str, Any]], Dict[str, Any]] task_handler, Any task_kwargs=None)

  Perform the assigned task in a background thread.
- str load_mongo_config (str database)

  Load MongoDB configuration from environment variables.
- str load_boss_config ()

  Load boss service callback URL from environment variables.
- Tuple[Callable[[Dict[str, Any]], Dict[str, Any]], Dict[str, Any]] get_task_info (str task_name)

  Dynamically import and return the appropriate task handler function.
- load_imports (func)

  Pre-warm the task by importing requirements.
- None mark_task_in_progress (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name)

  Mark a task as in-progress in MongoDB before processing begins.
- None save_task_result (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] result)

  Save completed task results to MongoDB.
- None notify_boss (str boss_url, str chunk_id, str task_name, str status)

  Send completion notification to boss service.
- Flask create_app (str task_name, str boss_url)

  Create and configure Flask application for task processing.

Variables

- MongoHandle = Generator["Database[Any]", None, None]
- parser = argparse.ArgumentParser(description="Flask worker microservice")
- required
- True
- help
- args = parser.parse_args()
- str task_queue = Queue()
- target
- task_worker ()

  Background threading system for non-blocking task handling.
- daemon
- str boss_url = load_boss_config()
- PORT = int(os.environ[f"{args.task.upper()}_PORT"])
- Flask app = create_app(args.task, boss_url)
- host
- port
- use_reloader

5.14.1 Detailed Description

Generic Flask worker microservice for distributed task processing.

Supports multiple task types via command-line arguments and dynamic imports.

5.14.2 Function Documentation

5.14.2.1 create_app()

```
Flask create_app (
             str task_name,
             str boss_url )
```

Create and configure Flask application for task processing.

Parameters

task_name	Type of task this worker will process.
boss_url	Callback URL for the boss service.

Returns

Configured Flask application instance.

5.14.2.2 get_task_info()

```
Tuple[Callable[[Dict[str, Any]], Dict[str, Any]], Dict[str, Any]] get_task_info (
             str task_name )
```

Dynamically import and return the appropriate task handler function.

Parameters

task_name	Name of the task type to execute.

Returns

Callable that processes the task data and returns results.

Exceptions

ImportError	If the task module cannot be imported.
AttributeError	If the task function is not found in the module.

5.14.2.3 load_boss_config()

```
str load_boss_config ( )
```

Load boss service callback URL from environment variables.

Returns

Full callback URL for the boss service.

Exceptions

KeyError	If PYTHON_HOST environment variable is missing.

5.14.2.4 load_imports()

```
load_imports (
              func )
```

Pre-warm the task by importing requirements.

Parameters

func	The function to perform a dummy call on.

5.14.2.5 load_mongo_config()

```
str load_mongo_config (
             str database )
```

Load MongoDB configuration from environment variables.

Parameters

database	Name of the MongoDB database to connect to.

Returns

MongoDB connection string.

Exceptions

KeyError	If required environment variables are missing.

5.14.2.6 mark_task_in_progress()

```
None mark_task_in_progress (
             MongoHandle mongo_db,
             str collection_name,
             str chunk_id,
             str task_name )
```

Mark a task as in-progress in MongoDB before processing begins.

Parameters

mongo_db	MongoDB database instance.
collection_name	The name of our primary chunk storage collection in Mongo.
chunk_id	Unique identifier for the chunk within the story.
task_name	Name of the task being executed.

Exceptions

RuntimeError	If task data already exists (preventing overwrites).

5.14.2.7 notify_boss()

```
None notify_boss (
             str boss_url,
             str chunk_id,
             str task_name,
             str status )
```

Send completion notification to boss service.

Parameters

boss_url	Callback URL for the boss service.
chunk_id	Unique identifier for the chunk within the story.
task_name	Name of the completed task.
status	Task completion status ('completed' or 'failed').

5.14.2.8 process_task()

```
process_task (
             MongoHandle mongo_db,
             str collection_name,
             str chunk_id,
             str task_name,
             Dict[str, Any] chunk_doc,
             str boss_url,
             Callable[[Dict[str, Any]], Dict[str, Any]] task_handler,
             Any  task_kwargs = None )
```

Perform the assigned task in a background thread.

This includes updating task status, running the handler, saving results, and notifying the boss service when complete.

Parameters

mongo_db	MongoDB database instance.
collection_name	The name of the target MongoDB collection.

Parameters

chunk_id	Unique identifier for the chunk within the story.
task_name	Name of the task being executed.
chunk_doc	Document data for the current chunk.
boss_url	Callback URL for the boss service.
task_handler	Function that performs the actual task computation.
task_kwargs	Dict of configuration settings for each task.

Exceptions

Exception	Logs and reports failures to the boss service.

5.14.2.9 save_task_result()

```
None save_task_result (
             MongoHandle mongo_db,
             str collection_name,
             str chunk_id,
             str task_name,
             Dict[str, Any] result )
```

Save completed task results to MongoDB.

Parameters

mongo_db	MongoDB database instance.
collection_name	The name of our primary chunk storage collection in Mongo.
chunk_id	Unique identifier for the chunk within the story.
task_name	Name of the task that was executed.
result	Dictionary containing task results to be stored.

5.14.3 Variable Documentation

5.14.3.1 app

```
Flask app = create_app(args.task, boss_url)
```

5.14.3.2 args

```
args = parser.parse_args()
```

5.14.3.3 boss_url

```
str boss_url = load_boss_config()
```

5.14.3.4 daemon

```
daemon
```

5.14.3.5 help

```
help
```

5.14.3.6 host

```
host
```

5.14.3.7 MongoHandle

```
MongoHandle = Generator["Database[Any]", None, None]
```

5.14.3.8 parser

```
parser = argparse.ArgumentParser(description="Flask worker microservice")
```

5.14.3.9 PORT

```
PORT = int(os.environ[f"{args.task.upper()}_PORT"])
```

5.14.3.10 port

```
port
```

5.14.3.11 required

```
required
```

5.14.3.12 target

```
target
```

5.14.3.13 task_queue

```
str task_queue = Queue()
```

5.14.3.14 task_worker

```
task_worker ( )
```

Background threading system for non-blocking task handling.

Allows Flask to immediately respond to the boss service (202: accepted) while processing continues asynchronously in a separate thread.

Continuously process tasks from the global queue in the background.

Each task runs sequentially (or with limited concurrency if multiple workers are started).

Exceptions

Exception	Logs any runtime errors that occur during task execution.

5.14.3.15 True

```
True
```

5.14.3.16 use_reloader

```
use_reloader
```

5.15 src.main Namespace Reference

Functions

- convert_single ()

  Converts one EPUB file to TEI format.
- convert_from_csv ()

  Converts several EPUB files to TEI format.
- chunk_single ()

  Creates a Story and many Chunks from a TEI file.
- test_relation_extraction ()

  Runs REBEL on a basic example; used for debugging.
- process_single ()

  Uses NLP and LLM to process an existing TEI file.
- graph_triple_files (session)

  Loads JSON into Neo4j to test the Blazor graph page.
- output_single (session)

  Generates a summary from triples stored in JSON, and posts data to Blazor.
- full_pipeline (session, collection_name, epub_path, book_chapters, start_str, end_str, book_id, story_id, book_title)
- old_main (session, collection_name)
- pipeline_1 (epub_path, book_chapters, start_str, end_str, book_id, story_id)

  Connects all components to convert an EPUB file to a book summary.
- pipeline_2 (session, collection_name, chunks, book_title)

  Extracts triples from a random chunk.
- pipeline_3 (session, triples)

  Generates a LLM summary using Neo4j triples.
- pipeline_4 (session, collection_name, triples_string, chunk_id)

  Generate chunk summary.
- pipeline_5a (summary, book_title, book_id)

  Send book info to Blazor.
- pipeline_5b (summary, book_title, book_id, chunk, gold_summary="", float bookscore=None, float questeval=None)

  Send metrics to Blazor.
- Dict[str, str] load_worker_config (List[str] task_types)

Load worker service URLs from environment variables.

- None clear_task_data (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name)

  Clear any existing task data before assigning new task to worker.
- bool assign_task_to_worker (str worker_url, str database_name, str collection_name, str chunk_id)

  Assign a task to a worker microservice.
- Flask create_app (DocumentConnector docs_db, str database_name, str collection_name, Dict[str, str] worker_urls)

  Create and configure Flask application for boss service.
- requests.models.Response post_story_status (int boss_port, int story_id, str task, str status)

  Helpers to interact with the Flask boss thread.
- requests.models.Response post_chunk_status (int boss_port, str chunk_id, int story_id, str task, str status)

  Send a chunk-level update to the boss Flask app.
- requests.models.Response post_process_full_story (int boss_port, int story_id, str task_type)

  Process all chunks in MongoDB matching the provided story ID.

Variables

- str tei = "./datasets/examples/trilogy-wishes-1.tei"

  Will revisit later - Book classes need refactoring ###.
- str chapters
- str start = ""
- str end = "But I must say no more."
- list triple_files
- list response_files = ["./datasets/triples/chunk-160_story-1.txt"]
- MongoHandle = Generator["Database[Any]", None, None]
- session = Session(verbose=False)
- DB_NAME = os.environ["DB_NAME"]
- BOSS_PORT = int(os.environ["PYTHON_PORT"])
- COLLECTION = os.environ["COLLECTION_NAME"]
- mongo_db = session.docs_db.get_unmanaged_handle()
- collection = getattr(mongo_db, COLLECTION)
- list task_types = ["questeval", "bookscore"]
- Dict[str, str] worker_urls = load_worker_config(task_types)
- Flask app = create_app(session.docs_db, DB_NAME, COLLECTION, worker_urls)
- app run_app = lambda.run(host="0.0.0.0", port=BOSS_PORT, use_reloader=False)
- target
- daemon
- int story_id = 1
- int book_id = 2
- str book_title = "The Phoenix and the Carpet"
- chunks
- triples
- chunk
- chunk_id = chunk.get_chunk_id()
- triples_string = pipeline_3(session, triples)
- summary = pipeline_4(session, COLLECTION, triples_string, chunk.get_chunk_id())
- requests.models.Response response = post_process_full_story(BOSS_PORT, story_id, task_type)

5.15.1 Function Documentation

5.15.1.1 assign_task_to_worker()

```
bool assign_task_to_worker (
             str worker_url,
             str database_name,
             str collection_name,
             str chunk_id )
```

Assign a task to a worker microservice.

Parameters

worker_url	Full URL of the worker's /start endpoint.
database_name	Name of the MongoDB database to use.
collection_name	The name of our primary chunk storage collection in Mongo.
chunk_id	Unique identifier for the chunk within the story.

Returns

True if task was successfully assigned, False otherwise.

5.15.1.2 chunk_single()

```
chunk_single ( )
```

Creates a Story and many Chunks from a TEI file.

Requires hard-coded specificaitons

- List of all chapter names.
- Optional start / end strings.

5.15.1.3 clear_task_data()

```
None clear_task_data (
             MongoHandle mongo_db,
             str collection_name,
             str chunk_id,
             str task_name )
```

Clear any existing task data before assigning new task to worker.

Parameters

mongo_db	MongoDB database handle.
collection_name	The name of our primary chunk storage collection in Mongo.
chunk_id	Unique identifier for the chunk within the story.
task_name	Name of the task to clear.

5.15.1.4 convert_from_csv()

```
convert_from_csv ( )
```

Converts several EPUB files to TEI format.

Note

Files are specified as rows in a CSV which contains parsing instructions.

5.15.1.5 convert_single()

```
convert_single ( )
```

Converts one EPUB file to TEI format.

5.15.1.6 create_app()

```
Flask create_app (
             DocumentConnector docs_db,
             str database_name,
             str collection_name,
             Dict[str, str] worker_urls )
```

Create and configure Flask application for boss service.

Parameters

docs_db	MongoDB connector class.
database_name	Name of the MongoDB database to use.
collection_name	The name of our primary chunk storage collection in Mongo.
worker_urls	Dictionary mapping task names to worker URLs.

Returns

Configured Flask application instance.

5.15.1.7 full_pipeline()

```
full_pipeline (
              session,
              collection_name,
              epub_path,
              book_chapters,
              start_str,
              end_str,
              book_id,
              story_id,
              book_title )
```

5.15.1.8 graph_triple_files()

```
graph_triple_files (
              session )
```

Loads JSON into Neo4j to test the Blazor graph page.

5.15.1.9 load_worker_config()

```
Dict[str, str] load_worker_config (
             List[str] task_types )
```

Load worker service URLs from environment variables.

Parameters

task_types	List of valid task keys to use when searching the .env

Returns

Dictionary mapping task names to worker URLs.

5.15.1.10 old_main()

```
old_main (
              session,
              collection_name )
```

5.15.1.11 output_single()

```
output_single (
              session )
```

Generates a summary from triples stored in JSON, and posts data to Blazor.

5.15.1.12 pipeline_1()

```
pipeline_1 (
              epub_path,
              book_chapters,
              start_str,
              end_str,
              book_id,
              story_id )
```

Connects all components to convert an EPUB file to a book summary.

Data conversions:

- EPUB file
- XML (TEI)

5.15.1.13 pipeline_2()

```
pipeline_2 (
              session,
              collection_name,
              chunks,
              book_title )
```

Extracts triples from a random chunk.

- JSON triples (NLP & LLM)

5.15.1.14 pipeline_3()

```
pipeline_3 (
              session,
              triples )
```

Generates a LLM summary using Neo4j triples.

- Neo4j graph database
- Blazor graph page

5.15.1.15 pipeline_4()

```
pipeline_4 (
              session,
              collection_name,
              triples_string,
              chunk_id )
```

Generate chunk summary.

5.15.1.16 pipeline_5a()

```
pipeline_5a (
              summary,
              book_title,
              book_id )
```

Send book info to Blazor.

- Post to Blazor metrics page

5.15.1.17 pipeline_5b()

```
pipeline_5b (
              summary,
              book_title,
              book_id,
              chunk,
              gold_summary = "",
             float  bookscore = None,
             float  questeval = None )
```

Send metrics to Blazor.

- Compute basic metrics (ROUGE, BERTScore)
- Wait for advanced metrics (QuestEval, BooookScore)
- Post to Blazor metrics page

5.15.1.18 post_chunk_status()

```
requests.models.Response post_chunk_status (
             int boss_port,
             str chunk_id,
             int story_id,
             str task,
             str status )
```

Send a chunk-level update to the boss Flask app.

Parameters

boss_port	Port the boss microservice is running on.
chunk_id	Unique identifier for the chunk.
story_id	Unique identifier for the story.
task	Task name (extraction, load_to_mongo, etc.).
status	Status (pending, assigned, in-progress, completed, failed).

Returns

JSON response indicating success or failure.

5.15.1.19 post_process_full_story()

```
requests.models.Response post_process_full_story (
             int boss_port,
             int story_id,
             str task_type )
```

Process all chunks in MongoDB matching the provided story ID.

Parameters

boss_port	Port the boss microservice is running on.
story_id	Unique identifier for the story.
task_type	Worker name (questeval, bookscore).

Returns

JSON response indicating success or failure.

5.15.1.20 post_story_status()

```
requests.models.Response post_story_status (
            int boss_port,
            int story_id,
            str task,
            str status )
```

Helpers to interact with the Flask boss thread.

Used to process our set of example books on pipeline start.

Send a story-level update to the boss Flask app.

Parameters

boss_port	Port the boss microservice is running on.
story_id	Unique identifier for the story.
task	Task name (extraction, load_to_mongo, etc.).
status	Status (pending, assigned, in-progress, completed, failed).

Returns

JSON response indicating success or failure.

5.15.1.21 process_single()

```
process_single ( )
```

Uses NLP and LLM to process an existing TEI file.

5.15.1.22 test_relation_extraction()

```
test_relation_extraction ( )
```

Runs REBEL on a basic example; used for debugging.

5.15.2 Variable Documentation

5.15.2.1 app

```
Flask app = create_app(session.docs_db, DB_NAME, COLLECTION, worker_urls)
```

5.15.2.2 book_id

```
int book_id = 2
```

5.15.2.3 book_title

```
str book_title = "The Phoenix and the Carpet"
```

5.15.2.4 BOSS_PORT

```
BOSS_PORT = int(os.environ["PYTHON_PORT"])
```

5.15.2.5 chapters

```
str chapters
```

Initial value:

```
=  """
CHAPTER 1 BEAUTIFUL AS THE DAY\n
CHAPTER 2 GOLDEN GUINEAS\n
CHAPTER 3 BEING WANTED\n
CHAPTER 4 WINGS\n
CHAPTER 5 NO WINGS\n
CHAPTER 6 A CASTLE AND NO DINNER\n
CHAPTER 7 A SIEGE AND BED\n
CHAPTER 8 BIGGER THAN THE BAKER'S BOY\n
CHAPTER 9 GROWN UP\n
CHAPTER 10 SCALPS\n
CHAPTER 11 THE LAST WISH\n
"""
```

5.15.2.6 chunk

```
chunk
```

5.15.2.7 chunk_id

```
chunk_id = chunk.get_chunk_id()
```

5.15.2.8 chunks

```
chunks
```

Initial value:

```
=  pipeline_1(
        epub_path="./datasets/examples/trilogy-wishes-2.epub",
        book_chapters=,
        start_str="",
        end_str="end of the Phoenix and the Carpet.",
        book_id=book_id,
        story_id=story_id,
    )
```

5.15.2.9 COLLECTION

```
COLLECTION = os.environ["COLLECTION_NAME"]
```

5.15.2.10 collection

```
collection = getattr(mongo_db, COLLECTION)
```

5.15.2.11 daemon

```
daemon
```

5.15.2.12 DB_NAME

```
DB_NAME = os.environ["DB_NAME"]
```

5.15.2.13 end

```
str end = "But I must say no more."
```

5.15.2.14 mongo_db

```
mongo_db = session.docs_db.get_unmanaged_handle()
```

5.15.2.15 MongoHandle

```
MongoHandle = Generator["Database[Any]", None, None]
```

5.15.2.16 response

```
requests.models.Response response = post_process_full_story(BOSS_PORT, story_id, task_type)
```

5.15.2.17 response_files

```
list response_files = ["./datasets/triples/chunk-160_story-1.txt"]
```

5.15.2.18 run_app

```
run_app = lambda.run(host="0.0.0.0", port=BOSS_PORT, use_reloader=False)
```

5.15.2.19 session

```
session = Session(verbose=False)
```

5.15.2.20 start

```
str start = ""
```

5.15.2.21 story_id

```
int story_id = 1
```

5.15.2.22 summary

```
summary = pipeline_4(session, COLLECTION, triples_string, chunk.get_chunk_id())
```

5.15.2.23 target

```
target
```

5.15.2.24 task_types

```
list task_types = ["questeval", "bookscore"]
```

5.15.2.25 tei

```
str tei = "./datasets/examples/trilogy-wishes-1.tei"
```

Will revisit later - Book classes need refactoring ###.

5.15.2.26 triple_files

```
list triple_files
```

Initial value:

```
00001 =  [
00002     "./datasets/triples/chunk-160_story-1.json",
00003     "./datasets/triples/chunk-70_story-1.json",
00004 ]
```

5.15.2.27 triples

```
triples
```

5.15.2.28 triples_string

```
triples_string = pipeline_3(session, triples)
```

5.15.2.29 worker_urls

```
Dict[str, str] worker_urls = load_worker_config(task_types)
```

5.16 src.setup Namespace Reference

Classes

- class Session

  Stores active database connections and configuration settings.

Variables

- session = Session()

5.16.1 Variable Documentation

5.16.1.1 session

```
session = Session()
```

5.17 src.util Namespace Reference

Classes

- class Log

  The Log class standardizes console output.

Functions

- all_none (∗args)

  Checks if all provided args are None.
- DataFrame df_natural_sorted (DataFrame df, List[str] ignored_columns=[ ], List[str] sort_columns=[ ])

  Sort a DataFrame in natural order using only certain columns.
- bool check_values (List[Any] results, List[Any] expected, bool verbose, str log_source, bool raise_error)

  Safely compare two lists of values.

5.17.1 Function Documentation

5.17.1.1 all_none()

```
all_none (
             * args )
```

Checks if all provided args are None.

5.17.1.2 check_values()

```
bool check_values (
             List[Any] results,
             List[Any] expected,
             bool verbose,
             str log_source,
             bool raise_error )
```

Safely compare two lists of values.

Helper for components.connectors.RelationalConnector.test_connection

Parameters

results	A list of observed values from the database.
expected	A list of correct values to compare against.
verbose	Whether to print success messages.
log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Exceptions

Log.Failure	If any result does not match what was expected.

5.17.1.3 df_natural_sorted()

```
DataFrame df_natural_sorted (
             DataFrame df,
             List[str]  ignored_columns = [],
             List[str]  sort_columns = [] )
```

Sort a DataFrame in natural order using only certain columns.

- Column order is alphabetic too, for completely predictable behavior.
- The provided DataFrame will not be modified, since inplace=False by default.
- Existing row numbers will be deleted and regenerated to match the sorted order.

Parameters

df	The DataFrame containing unsorted rows.
ignored_columns	A list of column names to NOT sort by.
sort_columns	A list of column names to sort by FIRST.

5.18 tests Namespace Reference

Namespaces

- namespace conftest
- namespace test_components

5.19 tests.conftest Namespace Reference

Functions

- pytest_addoption (parser)
- session (request)

  Fixture to create session.

5.19.1 Function Documentation

5.19.1.1 pytest_addoption()

```
pytest_addoption (
            parser )
```

5.19.1.2 session()

```
session (
            request )
```

Fixture to create session.

5.20 tests.test_components Namespace Reference

Functions

- RelationalConnector relational_db (Session session)

  Fixture to get relational database connection.
- DocumentConnector docs_db (Session session)

  Fixture to get document database connection.
- GraphConnector graph_db (Session session)

  Fixture to get document database connection.
- None test_db_relational_minimal (RelationalConnector relational_db)

  Tests if the RelationalConnector has a valid connection string.
- None test_db_docs_minimal (DocumentConnector docs_db)

  Tests if the DocumentConnector has a valid connection string.
- None test_db_graph_minimal (GraphConnector graph_db)

  Tests if the GraphConnector has a valid connection string.
- None test_db_relational_comprehensive (RelationalConnector relational_db)

  Tests if the GraphConnector is working as intended.
- None test_db_docs_comprehensive (DocumentConnector docs_db)

  Tests if the GraphConnector is working as intended.
- None test_db_graph_comprehensive (GraphConnector graph_db)

  Tests if the GraphConnector is working as intended.
- Generator[None, None, None] load_examples_relational (RelationalConnector relational_db)

  Fixture to create relational tables using engine-specific syntax.
- None test_sql_example_1 (RelationalConnector relational_db, Generator[None, None, None] load_examples_relational)

  Run queries contained within test files.
- None test_sql_example_2 (RelationalConnector relational_db, Generator[None, None, None] load_examples_relational)

  Run queries contained within test files.
- None test_mongo_example_1 (DocumentConnector docs_db)

  Run queries contained within test files.
- None test_mongo_example_2 (DocumentConnector docs_db)

  Run queries contained within test files.
- None test_mongo_example_3 (DocumentConnector docs_db)

  Run queries contained within test files.
- None test_cypher_example_1 (GraphConnector graph_db)

  Run queries contained within test files.
- None test_cypher_example_2 (GraphConnector graph_db)

  Test social network graph with relationships and mixed query patterns.
- None test_cypher_example_3 (GraphConnector graph_db)

  Test scene and dialogue graphs with proper isolation.
- None _test_query_file (DatabaseConnector db_fixture, str filename, List[str] valid_files)

  Run queries from a local file through the database.

5.20.1 Function Documentation

5.20.1.1 _test_query_file()

```
None _test_query_file (
            DatabaseConnector db_fixture,
            str filename,
            List[str] valid_files )  [protected]
```

Run queries from a local file through the database.

Parameters

db_fixture	Fixture corresponding to the current session's database.
filename	The name of a query file (for example ./tests/example1.sql).
valid_files	A list of file extensions valid for this database type.

5.20.1.2 docs_db()

```
DocumentConnector docs_db (
            Session session )
```

Fixture to get document database connection.

5.20.1.3 graph_db()

```
GraphConnector graph_db (
            Session session )
```

Fixture to get document database connection.

5.20.1.4 load_examples_relational()

```
Generator[None, None, None] load_examples_relational (
            RelationalConnector relational_db )
```

Fixture to create relational tables using engine-specific syntax.

5.20.1.5 relational_db()

```
RelationalConnector relational_db (
            Session session )
```

Fixture to get relational database connection.

5.20.1.6 test_cypher_example_1()

```
None test_cypher_example_1 (
            GraphConnector graph_db )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

5.20.1.7 test_cypher_example_2()

```
None test_cypher_example_2 (
             GraphConnector graph_db )
```

Test social network graph with relationships and mixed query patterns.

Validates comment parsing, semicolon splitting, CREATE/MERGE/MATCH, relationships with properties, and TAG_NODES_ with/without RETURN.

5.20.1.8 test_cypher_example_3()

```
None test_cypher_example_3 (
             GraphConnector graph_db )
```

Test scene and dialogue graphs with proper isolation.

Validates kg property isolation using a scene graph (spatial relationships) and dialogue graph (conversation flow with object references). Tests temp_graph context manager and filter_valid correctness across different graph contexts.

5.20.1.9 test_db_docs_comprehensive()

```
None test_db_docs_comprehensive (
             DocumentConnector docs_db )
```

Tests if the GraphConnector is working as intended.

5.20.1.10 test_db_docs_minimal()

```
None test_db_docs_minimal (
             DocumentConnector docs_db )
```

Tests if the DocumentConnector has a valid connection string.

5.20.1.11 test_db_graph_comprehensive()

```
None test_db_graph_comprehensive (
             GraphConnector graph_db )
```

Tests if the GraphConnector is working as intended.

5.20.1.12 test_db_graph_minimal()

```
None test_db_graph_minimal (
             GraphConnector graph_db )
```

Tests if the GraphConnector has a valid connection string.

5.20.1.13 test_db_relational_comprehensive()

```
None test_db_relational_comprehensive (
            RelationalConnector relational_db )
```

Tests if the GraphConnector is working as intended.

5.20.1.14 test_db_relational_minimal()

```
None test_db_relational_minimal (
            RelationalConnector relational_db )
```

Tests if the RelationalConnector has a valid connection string.

5.20.1.15 test_mongo_example_1()

```
None test_mongo_example_1 (
            DocumentConnector docs_db )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

5.20.1.16 test_mongo_example_2()

```
None test_mongo_example_2 (
            DocumentConnector docs_db )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

5.20.1.17 test_mongo_example_3()

```
None test_mongo_example_3 (
            DocumentConnector docs_db )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

5.20.1.18 test_sql_example_1()

```
None test_sql_example_1 (
            RelationalConnector relational_db,
            Generator[None, None, None] load_examples_relational )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

Note

Uses a table-creation fixture to load / unload schema.

5.20.1.19 test_sql_example_2()

```
None test_sql_example_2 (
            RelationalConnector relational_db,
            Generator[None, None, None] load_examples_relational )
```

Run queries contained within test files.

Internal errors are handled by the class itself, and ruled out earlier. Here we just assert that the received results DataFrame matches what we expected.

Note

Uses a table-creation fixture to load / unload schema.

Chapter 6

Class Documentation

6.1 Book Class Reference

Public Member Functions

- None __init__ (self, str title_key="Title:", str author_key="Author:", str language_key="Language:", str date↩ _key="Release date:")
- Iterator[Tuple[str, Dict[str, Any]]] stream_chapters (self)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 __init__()

```
None __init__ (
              self,
             str  title_key = "Title:",
             str  author_key = "Author:",
             str  language_key = "Language:",
             str  date_key = "Release date:" )
```

6.1.2 Member Function Documentation

6.1.2.1 stream_chapters()

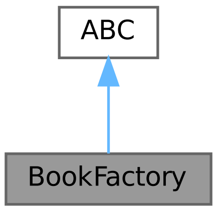
```
Iterator[Tuple[str, Dict[str, Any]]] stream_chapters (
              self )
```

The documentation for this class was generated from the following file:

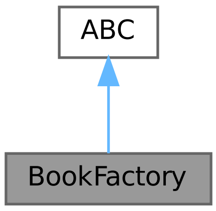
- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.2 BookFactory Class Reference

Inheritance diagram for BookFactory:



Collaboration diagram for BookFactory:



Public Member Functions

- Book create_book (self)

6.2.1 Member Function Documentation

6.2.1.1 create_book()

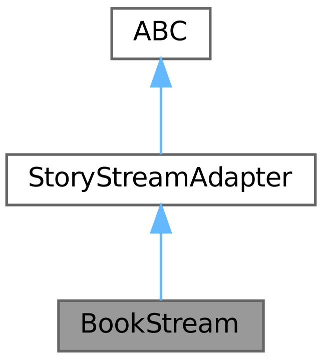
```
Book create_book (
              self )
```

The documentation for this class was generated from the following file:

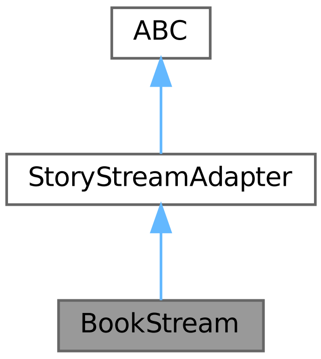
- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.3 BookStream Class Reference

Inheritance diagram for BookStream:



Collaboration diagram for BookStream:



Public Member Functions

- None __init__ (self, Book book)
- Iterator[Chunk] stream_segments (self)

    Yields sanitized parts of a book.

Public Member Functions inherited from StoryStreamAdapter

- Iterator[Chunk] stream_paragraphs (self)

    Concrete helper method to split segments into paragraphs.
- Iterator[str] stream_sentences (self)

    Concrete helper method to split paragraphs into sentences.

Public Attributes

- book

6.3.1 Constructor & Destructor Documentation

6.3.1.1 __init__()

```
None __init__ (
              self,
             Book book )
```

6.3.2 Member Function Documentation

6.3.2.1 stream_segments()

```
Iterator[Chunk] stream_segments (
              self )
```

Yields sanitized parts of a book.

- Story segments usually correspond to chapters.
- They serve as borders between chunking operations, ensuring chunks do not span multiple chapters. Implementation is handled by child classes BookStream, etc.
- Segments should be pre-cleaned and must contain 1 paragraph per line with all other newlines removed.

Reimplemented from StoryStreamAdapter.

6.3.3 Member Data Documentation

6.3.3.1 book

```
book
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.4 Chunk Class Reference

Lightweight container for a span of story text.

Public Member Functions

- None __init__ (self, str text, int book_id, int chapter_number, int line_start, int line_end, int story_id, float story_percent, float chapter_percent, int max_chunk_length=-1)

  Construct a Chunk.
- int char_count (self, bool prune_newlines=False)

  Computes the character count.
- str get_chunk_id (self)

  Use story ID, book ID, chapter, and chapter percentage to generate a chunk ID.
- Dict[str, Any] to_mongo_dict (self)

  Convert Chunk to Mongo document format.
- str __repr__ (self)

Public Attributes

- text
- book_id
- chapter_number
- line_start
- line_end
- story_id
- story_percent
- chapter_percent

6.4.1 Detailed Description

Lightweight container for a span of story text.

- Carries positional metadata so downstream consumers can reconstruct context.
- Filter by story_id to fetch all chunks for a particular story.
- Use story_percent and chapter_percent to quickly sort chunks by intended order.
- Use book_id, chapter_number, line_start, and line_end to locate this chunk within source material.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 __init__()

```
None __init__ (
              self,
             str text,
             int book_id,
             int chapter_number,
             int line_start,
             int line_end,
             int story_id,
             float story_percent,
             float chapter_percent,
             int  max_chunk_length = -1 )
```

Construct a Chunk.

Parameters

text	The text content for this span.
book_id	Corresponds to a single book file in the dataset.
chapter_number	The chapter containing this chunk in the book file, 1-based.
line_start	The starting line within the TEI file, 1-based.
line_end	The inclusive ending line index within the TEI file (>= line_start).
story_id	A stable id for the overall story. May be identical to book_id
story_percent	Approximate progress through the whole story [0.0, 100.0].
chapter_percent	Approximate progress through the current segment [0.0, 100.0].
max_chunk_length	Max allowed characters (<= 0 means "no limit").

Exceptions

ValueError	if text exceeds max_chunk_length when max_chunk_length $>$ 0.

6.4.3 Member Function Documentation

6.4.3.1 __repr__()

```
str __repr__ (
              self )
```

6.4.3.2 char_count()

```
int char_count (
              self,
             bool  prune_newlines = False )
```

Computes the character count.

Parameters

prune_newlines	Whether to remove newlines for the count.

Returns

The number of characters in the chunk text.

6.4.3.3 get_chunk_id()

```
str get_chunk_id (
              self )
```

Use story ID, book ID, chapter, and chapter percentage to generate a chunk ID.

Returns

A string uniquely identifying a chunk.

6.4.3.4 to_mongo_dict()

```
Dict[str, Any] to_mongo_dict (
              self )
```

Convert Chunk to Mongo document format.

Returns

A dictionary which can be easily loaded into MongoDB.

6.4.4 Member Data Documentation

6.4.4.1 book_id

```
book_id
```

6.4.4.2 chapter_number

```
chapter_number
```

6.4.4.3 chapter_percent

```
chapter_percent
```

6.4.4.4 line_end

```
line_end
```

6.4.4.5 line_start

```
line_start
```

6.4.4.6 story_id

```
story_id
```

6.4.4.7 story_percent

```
story_percent
```

6.4.4.8 text

```
text
```

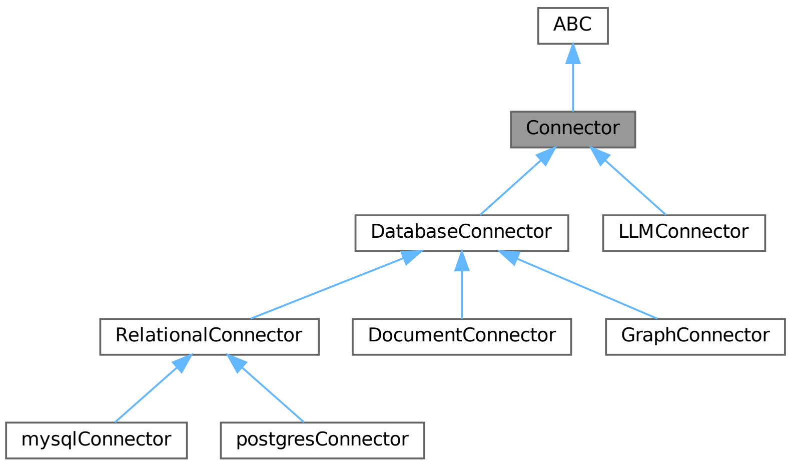
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

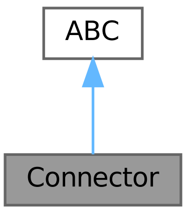
6.5 Connector Class Reference

Abstract base class for external connectors.

Inheritance diagram for Connector:



Collaboration diagram for Connector:



Public Member Functions

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- bool test_connection (self, bool raise_error=True)

  Establish a basic connection to the database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] execute_query (self, str query)

  Send a single command through the connection.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several commands from a file.

6.5.1 Detailed Description

Abstract base class for external connectors.

Note

Credentials are specified in the .env file.

Derived classes should implement:

- **init**
- components.connectors.Connector.configure
- components.connectors.Connector.test_connection
- components.connectors.Connector.execute_query
- components.connectors.Connector.execute_file

6.5.2 Member Function Documentation

6.5.2.1 check_connection()

```
bool check_connection (
              self,
             str log_source,
             bool raise_error )
```

Minimal connection test to determine if our connection string is valid.

Parameters

log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented in RelationalConnector, DocumentConnector, GraphConnector, and LLMConnector.

6.5.2.2 configure()

```
None configure (
              self,
             str DB,
             str database_name )
```

Read connection settings from the .env file.

Parameters

DB	The prefix of fetched credentials.
database_name	The specific service to connect to.

Reimplemented in LLMConnector, and DatabaseConnector.

6.5.2.3 execute_file()

```
List[Optional[DataFrame]] execute_file (
              self,
             str filename )
```

Run several commands from a file.

Parameters

filename	The path to a specified query or prompt file (.sql, .txt).

Returns

Whether the query was performed successfully.

Reimplemented in DatabaseConnector, and LLMConnector.

6.5.2.4 execute_query()

```
Optional[DataFrame] execute_query (
              self,
             str query )
```

Send a single command through the connection.

Parameters

query	A single query to perform on the database.

Returns

The result of the query, or None

Reimplemented in DatabaseConnector, RelationalConnector, DocumentConnector, LLMConnector, and GraphConnector.

6.5.2.5 test_connection()

```
bool test_connection (
              self,
             bool   raise_error = True )
```

Establish a basic connection to the database, and test full functionality.

Can be configured to fail silently, which enables retries or external handling.

Parameters

raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented in LLMConnector, RelationalConnector, DocumentConnector, and GraphConnector.

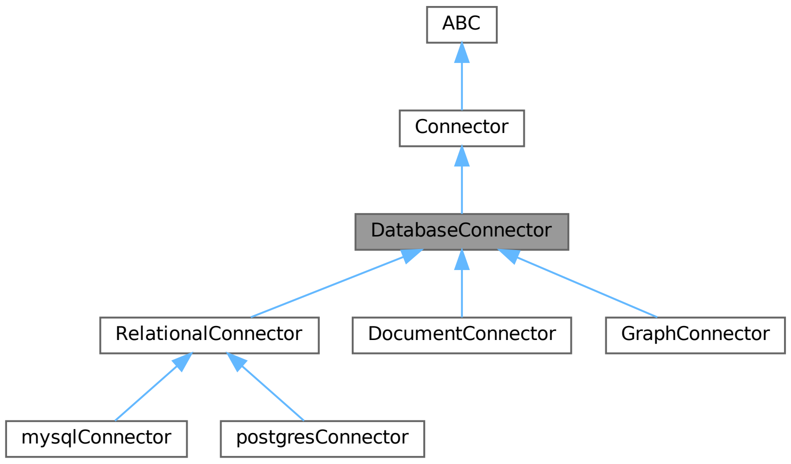
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py

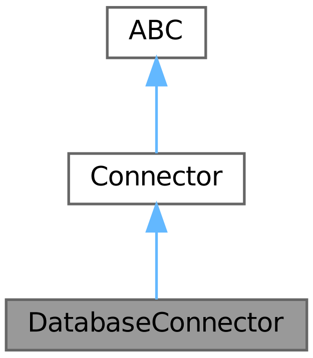
6.6 DatabaseConnector Class Reference

Abstract base class for database engine connectors.

Inheritance diagram for DatabaseConnector:



Collaboration diagram for DatabaseConnector:



Public Member Functions

- None __init__ (self, bool verbose=False)

  Initialize the connector.
- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- None change_database (self, str new_database)

  Update the connection URI to reference a different database in the same engine.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- Optional[DataFrame] execute_query (self, str query)

Send a single command through the connection.

- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

  Automatically generate and run a query for the specified resource.
- None create_database (self, str database_name)

  Use the current database connection to create a sibling database in this engine.
- None drop_database (self, str database_name)

  Delete all data stored in a particular database.
- bool database_exists (self, str database_name)

  Search for an existing database using the provided name.

Public Member Functions inherited from Connector

- bool test_connection (self, bool raise_error=True)

  Establish a basic connection to the database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.

Public Attributes

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.
- List[str] _split_combined (self, str multi_query)

  Checks if a string contains multiple queries.

6.6.1 Detailed Description

Abstract base class for database engine connectors.

Derived classes should implement:

- components.connectors.DatabaseConnector.__init__
- components.connectors.DatabaseConnector.test_connection
- components.connectors.DatabaseConnector.execute_query
- components.connectors.DatabaseConnector._split_combined
- components.connectors.DatabaseConnector.get_dataframe
- components.connectors.DatabaseConnector.create_database
- components.connectors.DatabaseConnector.drop_database
- components.connectors.DatabaseConnector.change_database
- components.connectors.DatabaseConnector.database_exists

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

```
None __init__ (
              self,
             bool   verbose = False )
```

Initialize the connector.

Parameters

verbose	Whether to print debug messages.

Note

Attributes will be set to None until components.connectors.DatabaseConnector.configure() is called.

Reimplemented in RelationalConnector, mysqlConnector, postgresConnector, DocumentConnector, and GraphConnector.

6.6.3 Member Function Documentation

6.6.3.1 _is_single_query()

```
bool _is_single_query (
              self,
             str query )   [protected]
```

Checks if a string contains multiple queries.

Parameters

query	A single or combined query string.

Returns

Whether the query is single (true) or combined (false).

6.6.3.2 _split_combined()

```
List[str] _split_combined (
              self,
             str multi_query )  [protected]
```

Checks if a string contains multiple queries.

Parameters

multi_query	A string containing multiple queries.

Returns

A list of single-query strings.

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.3 change_database()

```
None change_database (
              self,
             str new_database )
```

Update the connection URI to reference a different database in the same engine.

Parameters

new_database	The name of the database to connect to.

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.4 configure()

```
None configure (
              self,
             str DB,
             str database_name )
```

Read connection settings from the .env file.

Parameters

DB	The prefix of fetched database credentials.
database_name	The name of the database to connect to.

Reimplemented from Connector.

6.6.3.5 create_database()

```
None create_database (
               self,
              str database_name )
```

Use the current database connection to create a sibling database in this engine.

Parameters

database_name	The name of the new database to create.

Exceptions

Log.Failure	If the database already exists.

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.6 database_exists()

```
bool database_exists (
               self,
              str database_name )
```

Search for an existing database using the provided name.

Parameters

database_name	The name of a database to search for.

Returns

Whether the database is visible to this connector.

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.7 drop_database()

```
None drop_database (
               self,
              str database_name )
```

Delete all data stored in a particular database.

Parameters

database_name	The name of an existing database.

Exceptions

Log.Failure	If the database does not exist.

Reimplemented in DocumentConnector, GraphConnector, and RelationalConnector.

6.6.3.8 execute_combined()

```
List[Optional[DataFrame]] execute_combined (
                self,
                str multi_query )
```

Run several database commands in sequence.

Parameters

multi_query	A string containing multiple queries.

Returns

A list of query results converted to DataFrames.

6.6.3.9 execute_file()

```
List[Optional[DataFrame]] execute_file (
                self,
                str filename )
```

Run several database commands from a file.

Note

Loads the entire file into memory at once.

Parameters

filename	The path to a specified query file (.sql, .cql, .json).

Returns

Whether the query was performed successfully.

Exceptions

Log.Failure	If any query in the file fails to execute.

Reimplemented from Connector.

6.6.3.10 execute_query()

```
Optional[DataFrame] execute_query (
                self,
               str query )
```

Send a single command through the connection.

Note

If a result is returned, it will be converted to a DataFrame.

Parameters

query	A single query to perform on the database.

Returns

DataFrame containing the result of the query, or None

Exceptions

Log.Failure	If the query fails to execute.

Reimplemented from Connector.

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.11 get_dataframe()

```
Optional[DataFrame] get_dataframe (
                self,
               str name,
               List[str]  columns = [] )
```

Automatically generate and run a query for the specified resource.

Parameters

name	The name of an existing table or collection in the database.
columns	A list of column names to keep.

Returns

DataFrame containing the requested data, or None

Reimplemented in RelationalConnector, DocumentConnector, and GraphConnector.

6.6.3.12 temp_database()

```
Generator[None, None, None] temp_database (
              self,
             str database_name )
```

Temporarily switch to a pseudo-database, creating and dropping it if needed.

- If the target database does not exist, it will be created before yielding and dropped automatically afterward.
- If it already exists, it will be left intact.

Parameters

database_name	The name of the pseudo-database to use temporarily.

6.6.4 Member Data Documentation

6.6.4.1 connection_string

```
connection_string
```

6.6.4.2 db_engine

```
db_engine
```

6.6.4.3 db_type

```
db_type
```

6.6.4.4 host

```
host
```

6.6.4.5 password

```
password
```

6.6.4.6 port

`port`

6.6.4.7 username

`username`

6.6.4.8 verbose

`verbose`

Whether to print debug messages.

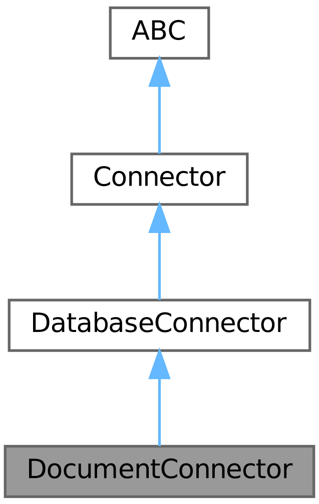
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py

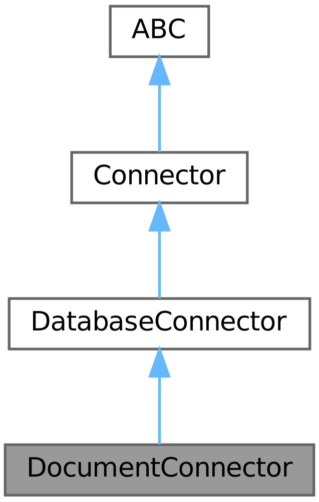
6.7 DocumentConnector Class Reference

Connector for MongoDB (document database)

Inheritance diagram for DocumentConnector:



Collaboration diagram for DocumentConnector:



Public Member Functions

- None __init__ (self, bool verbose=False)

  Creates a new MongoDB connector.
- None change_database (self, str new_database)

  Update the connection URI to reference a different database in the same engine.
- bool test_connection (self, bool raise_error=True)

  Establish a basic connection to the MongoDB database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.
- get_unmanaged_handle (self)

  Expose the low-level PyMongo handle for external use.
- Optional[DataFrame] execute_query (self, str query)

  Send a single MongoDB command using PyMongo.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

  Automatically generate and run a query for the specified collection.
- None create_database (self, str database_name)

  Use the current database connection to create a sibling database in this engine.
- None drop_database (self, str database_name)

  Delete all data stored in a particular database.
- bool database_exists (self, str database_name)

  Search for an existing database using the provided name.
- None delete_dummy (self)

  Delete the initial dummy collection from the database.

Public Member Functions inherited from DatabaseConnector

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.

Public Attributes

- database_name
- verbose
- connection_string

Public Attributes inherited from DatabaseConnector

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions

- list[str] _split_combined (self, str multi_query)

  Divides a string into non-divisible MongoDB commands by splitting on semicolons at depth 0.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

Protected Attributes

- _auth_suffix

6.7.1 Detailed Description

Connector for MongoDB (document database)

- Uses mongoengine.connect(...) on-demand for connections.
- Low-level operations use pymongo via mongoengine.get_db().
- create_database uses an init collection insertion (MongoDB is lazy).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 __init__()

```
None __init__ (
              self,
             bool  verbose = False )
```

Creates a new MongoDB connector.

Parameters

verbose	Whether to print debug messages.

Reimplemented from DatabaseConnector.

6.7.3 Member Function Documentation

6.7.3.1 _split_combined()

```
list[str] _split_combined (
              self,
             str multi_query )  [protected]
```

Divides a string into non-divisible MongoDB commands by splitting on semicolons at depth 0.

Handles nested brackets and semicolons inside JSON strings.

Parameters

multi_query	A string containing multiple queries with possible comments.

Returns

A list of single-query strings (cleaned, ready for JSON parsing).

Reimplemented from DatabaseConnector.

6.7.3.2 change_database()

```
None change_database (
              self,
             str new_database )
```

Update the connection URI to reference a different database in the same engine.

Note

Additional settings are appended as a suffix to the MongoDB connection string.

Parameters

new_database	The name of the database to connect to.

Reimplemented from DatabaseConnector.

6.7.3.3 check_connection()

```
bool check_connection (
              self,
             str log_source,
             bool raise_error )
```

Minimal connection test to determine if our connection string is valid.

Connect to MongoDB using MongoEnigine.connect()

Parameters

log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.7.3.4 create_database()

```
None create_database (
              self,
             str database_name )
```

Use the current database connection to create a sibling database in this engine.

Note

Forces MongoDB to actually create it by inserting a small init document.

Parameters

database_name	The name of the new database to create.

Exceptions

Log.Failure	If we fail to create the requested database for any reason.

Reimplemented from DatabaseConnector.

6.7.3.5 database_exists()

```
bool database_exists (
             self,
            str database_name )
```

Search for an existing database using the provided name.

Parameters

database_name	The name of a database to search for.

Returns

Whether the database is visible to this connector.

Reimplemented from DatabaseConnector.

6.7.3.6 delete_dummy()

```
None delete_dummy (
             self )
```

Delete the initial dummy collection from the database.

Note

Call this method whenever real data is being added to avoid pollution.

6.7.3.7 drop_database()

```
None drop_database (
             self,
            str database_name )
```

Delete all data stored in a particular database.

Parameters

database_name	The name of an existing database.

Exceptions

Log.Failure	If we fail to drop the target database for any reason.

Reimplemented from DatabaseConnector.

6.7.3.8 execute_query()

```
Optional[DataFrame] execute_query (
                 self,
                str query )
```

Send a single MongoDB command using PyMongo.

- The query must be a valid JSON command object (e.g. {"find": "users", "filter": {...}}).
- Mongo shell syntax such as `db.users.find({...})` or `.js` files will NOT work.
- If a result is returned, it will be converted to a DataFrame.

Exceptions

Log.Failure	If the query fails to execute.

Reimplemented from DatabaseConnector.

6.7.3.9 get_dataframe()

```
Optional[DataFrame] get_dataframe (
                 self,
                str name,
                List[str]  columns = [] )
```

Automatically generate and run a query for the specified collection.

Parameters

name	The name of an existing table or collection in the database.
columns	A list of column names to keep.

Returns

DataFrame containing the requested data, or None

Exceptions

Log.Failure	If we fail to create the requested DataFrame for any reason.

Reimplemented from DatabaseConnector.

6.7.3.10 get_unmanaged_handle()

```
get_unmanaged_handle (
              self )
```

Expose the low-level PyMongo handle for external use.

Warning

Connection remains open - use for long-lived services only.

Returns

PyMongo database instance.

6.7.3.11 test_connection()

```
bool test_connection (
              self,
             bool  raise_error = True )
```

Establish a basic connection to the MongoDB database, and test full functionality.

Can be configured to fail silently, which enables retries or external handling.

Parameters

raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.7.4 Member Data Documentation

6.7.4.1 _auth_suffix

```
_auth_suffix  [protected]
```

6.7.4.2 connection_string

```
connection_string
```

6.7.4.3 database_name

```
database_name
```

6.7.4.4 verbose

```
verbose
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py

6.8 EPUBToTEI Class Reference

Converts EPUB files to XML format (TEI specification).

Public Member Functions

- None __init__ (self, str epub_path, bool save_pandoc=False, bool save_tei=True)

  Initialize the converter.
- None convert_to_tei (self)

  Uses Pandoc to draft a TEI string from EPUB.
- None clean_tei (self)

  Wrap root if missing, sanitize ids, and save cleaned TEI.

Public Attributes

- epub_path
- pandoc_xml_path
- raw_tei_content
- clean_tei_content
- tei_path

Static Public Attributes

- dict xml_namespace = {"tei": "http://www.tei-c.org/ns/1.0"}
- str encoding = "utf-8"

Protected Member Functions

- str _sanitize_ids (self, str content)

  Sanitize XML IDs in the TEI content to ensure they are valid and consistent.
- str _prune_bad_tags (self, str content)

  Replace all `lb` tags with newline characters in TEI.

6.8.1 Detailed Description

Converts EPUB files to XML format (TEI specification).

Takes an EPUB book file and converts it to TEI in order to represent its chapter hierarchy.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
None __init__ (
              self,
             str epub_path,
             bool  save_pandoc = False,
             bool  save_tei = True )
```

Initialize the converter.

Parameters

epub_path	String containing the relative path to an EPUB file.
save_pandoc	Flag to save the intermediate Pandoc output to .tei.xml
save_tei	Flag to save the final TEI file as .tei

6.8.3 Member Function Documentation

6.8.3.1 _prune_bad_tags()

```
str _prune_bad_tags (
              self,
             str content )  [protected]
```

Replace all `lb` tags with newline characters in TEI.

6.8.3.2 _sanitize_ids()

```
str _sanitize_ids (
              self,
             str content )  [protected]
```

Sanitize XML IDs in the TEI content to ensure they are valid and consistent.

Pandoc sometimes generates invalid or non-unique `xml:id` attributes (e.g., containing spaces, punctuation, or mixed casing). Since we rely on these IDs as dictionary keys / anchors, we sanitize them using a regex to enforce alphanumeric/underscore/dash format.

Parameters

content	The raw TEI XML string possibly containing invalid xml:id attributes.

Returns

A TEI XML string with valid NCNames, prefixed with 'id_'.

6.8.3.3 clean_tei()

```
None clean_tei (
              self )
```

Wrap root if missing, sanitize ids, and save cleaned TEI.

6.8.3.4 convert_to_tei()

```
None convert_to_tei (
              self )
```

Uses Pandoc to draft a TEI string from EPUB.

6.8.4 Member Data Documentation

6.8.4.1 clean_tei_content

```
clean_tei_content
```

6.8.4.2 encoding

```
str encoding = "utf-8"  [static]
```

6.8.4.3 epub_path

```
epub_path
```

6.8.4.4 pandoc_xml_path

```
pandoc_xml_path
```

6.8.4.5 raw_tei_content

```
raw_tei_content
```

6.8.4.6 tei_path

```
tei_path
```

6.8.4.7 xml_namespace

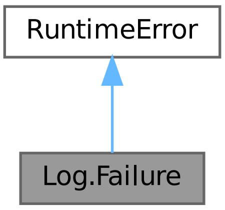
```
dict xml_namespace = {"tei":  "http://www.tei-c.org/ns/1.0"}  [static]
```

The documentation for this class was generated from the following file:

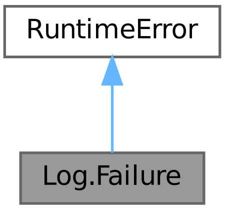
- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.9 Log.Failure Class Reference

Inheritance diagram for Log.Failure:



Collaboration diagram for Log.Failure:



Public Member Functions

- __init__ (self, str prefix="ERROR", str msg="")
- __str__ (self)

Public Attributes

- prefix
- msg

6.9.1 Constructor & Destructor Documentation

6.9.1.1 __init__()

```
__init__ (
              self,
             str  prefix = "ERROR",
             str  msg = "" )
```

6.9.2 Member Function Documentation

6.9.2.1 __str__()

```
__str__ (
              self )
```

6.9.3 Member Data Documentation

6.9.3.1 msg

```
msg
```

6.9.3.2 prefix

```
prefix
```

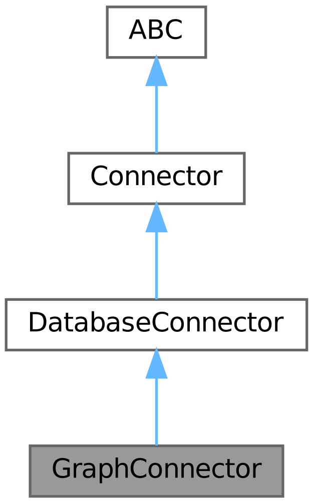
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/util.py

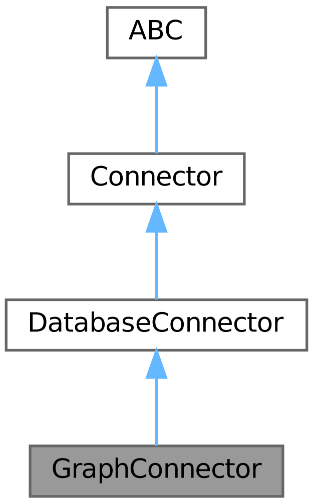
6.10 GraphConnector Class Reference

Connector for Neo4j (graph database).

Inheritance diagram for GraphConnector:



Collaboration diagram for GraphConnector:



Public Member Functions

- None __init__ (self, bool verbose=False)

  Creates a new Neo4j connector.
- None change_database (self, str new_database)

  Update the connection URI to reference a different database in the same engine.
- Generator[None, None, None] temp_graph (self, str graph_name)

  Temporarily inspect the specified graph, then swap back when finished.
- bool test_connection (self, bool raise_error=True)

  Establish a basic connection to the Neo4j database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] execute_query (self, str query, bool _filter_results=True)

  Send a single Cypher query to Neo4j.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

  Automatically generate and run a query for the specified Knowledge Graph collection.
- List[str] get_unique (self, str key)

  Retrieve all unique values for a specified node property.
- None create_database (self, str database_name)

  Create a fresh pseudo-database if it does not already exist.
- None drop_database (self, str database_name)

  Delete all nodes stored under a particular database name.
- None drop_graph (self, str graph_name)

  Delete all nodes stored under a particular graph name.
- bool database_exists (self, str database_name)

  Search for an existing database using the provided name.
- None delete_dummy (self)

  Delete the initial dummy node from the database.
- None add_triple (self, str subject, str relation, str object_)

  Add a semantic triple to the graph using raw Cypher.
- DataFrame get_edge_counts (self, int top_n=10)

  Return node names and their edge counts, ordered by edge count descending.
- DataFrame get_all_triples (self, Optional[str] graph_name=None)

  Return all triples in the specified graph as a pandas DataFrame.
- None print_nodes (self, int max_rows=20, int max_col_width=50)

  Print all nodes and edges in the current pseudo-database with row/column formatting.
- None print_triples (self, int max_rows=20, int max_col_width=50)

  Print all nodes and edges in the current pseudo-database with row/column formatting.
- str IS_DUMMY_ (self, str alias='n')

  Generates Cypher code to select dummy nodes inside a WHERE clause.
- str NOT_DUMMY_ (self, str alias='n')

  Generates Cypher code to select non-dummy nodes inside a WHERE clause.
- str SAME_DB_KG_ (self)

  Generates a Cypher pattern dictionary to match nodes by current database and graph name.

Public Member Functions inherited from DatabaseConnector

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.

Static Public Member Functions

- List[Tuple[str, str, str]] normalize_triples (Any data)

  Normalize flexible LLM output into a list of clean (subject, relation, object) triples.

Public Attributes

- database_name
- verbose
- graph_name
- connection_string

Public Attributes inherited from DatabaseConnector

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions

- List[str] _split_combined (self, str multi_query)

  Divides a string into non-divisible CQL queries, ignoring comments.
- None _execute_tag_db (self)

  Sweeps the database for untagged nodes and relationships, and adds a 'db' attribute.
- Tuple[Optional[List[Tuple[Any,...]]], Optional[List[str]]] _get_updated (self, List[Tuple[Any,...]] results)

  Re-fetch nodes and edges after changing the remote copy in Neo4j.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

6.10.1 Detailed Description

Connector for Neo4j (graph database).

- Uses neomodel to abstract some operations, but raw CQL is required for many tasks.
- Neo4j does not support multiple logical databases in community edition, so we emulate them.
- This is achieved by using a 'db' property (database name) and 'kg' property (graph name) on nodes.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 __init__()

```
None __init__ (
                  self,
                bool   verbose = False )
```

Creates a new Neo4j connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from DatabaseConnector.

6.10.3 Member Function Documentation

6.10.3.1 _execute_tag_db()

```
None _execute_tag_db (
              self )  [protected]
```

Sweeps the database for untagged nodes and relationships, and adds a 'db' attribute.

6.10.3.2 _get_updated()

```
Tuple[Optional[List[Tuple[Any, ...]]], Optional[List[str]]] _get_updated (
              self,
             List[Tuple[Any, ...]] results )  [protected]
```

Re-fetch nodes and edges after changing the remote copy in Neo4j.

Parameters

results	Original list of untagged tuples from db.cypher_query().

Returns

Latest version of results fetched from the database.

6.10.3.3 _split_combined()

```
List[str] _split_combined (
              self,
             str multi_query )  [protected]
```

Divides a string into non-divisible CQL queries, ignoring comments.

Parameters

multi_query	A string containing multiple queries.

Returns

A list of single-query strings.

Reimplemented from DatabaseConnector.

6.10.3.4 add_triple()

```
None add_triple (
              self,
             str subject,
             str relation,
             str object_ )
```

Add a semantic triple to the graph using raw Cypher.

Parameters

subject	A string representing the entity performing an action.
relation	A string describing the action.
object← _	A string representing the entity being acted upon.

Note

LLM output should be pre-normalized using components.fact_storage.GraphConnector.normalize_triples.

Exceptions

Log.Failure	If the triple cannot be added to our graph database.

6.10.3.5 change_database()

```
None change_database (
              self,
             str new_database )
```

Update the connection URI to reference a different database in the same engine.

Note

Neo4j does not accept database names routed through the connection string.

Parameters

new_database	The name of the database to connect to.

Reimplemented from DatabaseConnector.

6.10.3.6 check_connection()

```
bool check_connection (
              self,
            str log_source,
            bool raise_error )
```

Minimal connection test to determine if our connection string is valid.

Connect to Neo4j executing a query: db.cypher_query()

Parameters

log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.10.3.7 create_database()

```
None create_database (
             self,
            str database_name )
```

Create a fresh pseudo-database if it does not already exist.

Note

This change will apply to any new nodes created after components.connectors.DatabaseConnector.change_database is called.

Parameters

database_name	A database ID specifying the pseudo-database.

Exceptions

Log.Failure	If we fail to create the requested database for any reason.

Reimplemented from DatabaseConnector.

6.10.3.8 database_exists()

```
bool database_exists (
              self,
             str database_name )
```

Search for an existing database using the provided name.

Parameters

database_name	The name of a database to search for.

Returns

Whether the database is visible to this connector.

Reimplemented from DatabaseConnector.

6.10.3.9 delete_dummy()

```
None delete_dummy (
              self )
```

Delete the initial dummy node from the database.

Note

Never use this. Enables the existence of an "empty" database.

6.10.3.10 drop_database()

```
None drop_database (
              self,
             str database_name )
```

Delete all nodes stored under a particular database name.

Parameters

database_name	A database ID specifying the pseudo-database.

Exceptions

Log.Failure	If we fail to drop the target database for any reason.

Reimplemented from DatabaseConnector.

6.10.3.11 drop_graph()

```
None drop_graph (
              self,
             str graph_name )
```

Delete all nodes stored under a particular graph name.

Parameters

graph_name	The name of a graph in the current database.

Exceptions

Log.Failure	If we fail to drop the target graph for any reason.

6.10.3.12 execute_query()

```
Optional[DataFrame] execute_query (
              self,
             str query,
             bool  _filter_results = True )
```

Send a single Cypher query to Neo4j.

Note

If a result is returned, it will be converted to a DataFrame.

Parameters

query	A single query to perform on the database.
_filter_results	If True, limit results to the current database. Needed for internal helper functions.

Returns

DataFrame containing the result of the query, or None

Exceptions

Log.Failure	If the query fails to execute.

Reimplemented from DatabaseConnector.

6.10.3.13 get_all_triples()

```
DataFrame get_all_triples (
              self,
             Optional[str]  graph_name = None )
```

Return all triples in the specified graph as a pandas DataFrame.

Parameters

graph_name	The graph to query. If None, uses self.graph_name.

Exceptions

Log.Failure	If the query fails to retrieve the requested DataFrame.

6.10.3.14 get_dataframe()

```
Optional[DataFrame] get_dataframe (
                self,
               str name,
               List[str]  columns = [] )
```

Automatically generate and run a query for the specified Knowledge Graph collection.

- Fetches all public node attributes, the internal ID, and all labels (e.g. :Person :Character)
- Does not explode lists or nested values
- Different approach than DocumentConnector because our node attributes are usually flat key:value already.

Parameters

name	The name of an existing table or collection in the database.
columns	A list of column names to keep.

Returns

DataFrame containing the requested data, or None

Exceptions

Log.Failure	If we fail to create the requested DataFrame for any reason.

Reimplemented from DatabaseConnector.

6.10.3.15 get_edge_counts()

```
DataFrame get_edge_counts (
                self,
               int  top_n = 10 )
```

Return node names and their edge counts, ordered by edge count descending.

Parameters

top↩_n	Number of top nodes to return (by edge count). Default is 10.

Returns

DataFrame with columns: node_name, edge_count

Exceptions

Log.Failure	If the query fails to retrieve the requested DataFrame.

6.10.3.16 get_unique()

```
List[str] get_unique (
              self,
             str key )
```

Retrieve all unique values for a specified node property.

Queries all nodes in the database and extracts distinct values for the given key.

Parameters

key	The node property name to extract unique values from (e.g. 'db' or 'kg').

Returns

A list of unique values for the specified key, or an empty list if none exist.

Exceptions

Log.Failure	If the query fails to execute.

6.10.3.17 IS_DUMMY_()

```
str IS_DUMMY_ (
              self,
             str  alias = 'n' )
```

Generates Cypher code to select dummy nodes inside a WHERE clause.

Usage: MATCH (n) WHERE {self.IS_DUMMY_('n')};

Returns

A string containing Cypher code.

6.10.3.18 normalize_triples()

```
List[Tuple[str, str, str]] normalize_triples (
             Any data )  [static]
```

Normalize flexible LLM output into a list of clean (subject, relation, object) triples.

- Accepts dicts, lists of dicts, tuples, or dicts-of-lists.
- Joins list values, trims, and sanitizes for Cypher safety.
- Enforces uppercase underscore-safe relation labels.

Parameters

data	Raw LLM output to normalize.

Returns

List of sanitized (s, r, o) triples ready for insertion.

Exceptions

ValueError	If input format cannot be parsed.

6.10.3.19 NOT_DUMMY_()

```
str NOT_DUMMY_ (
              self,
             str  alias = 'n' )
```

Generates Cypher code to select non-dummy nodes inside a WHERE clause.

Usage: MATCH (n) WHERE {self.NOT_DUMMY_('n')};

Returns

A string containing Cypher code.

6.10.3.20 print_nodes()

```
None print_nodes (
              self,
             int  max_rows = 20,
             int  max_col_width = 50 )
```

Print all nodes and edges in the current pseudo-database with row/column formatting.

6.10.3.21 print_triples()

```
None print_triples (
              self,
             int  max_rows = 20,
             int  max_col_width = 50 )
```

Print all nodes and edges in the current pseudo-database with row/column formatting.

6.10.3.22 SAME_DB_KG_()

```
str SAME_DB_KG_ (
              self )
```

Generates a Cypher pattern dictionary to match nodes by current database and graph name.

Usage: MATCH (n {self.SAME_DB_KG_()})

Returns

A string containing Cypher code.

6.10.3.23 temp_graph()

```
Generator[None, None, None] temp_graph (
              self,
             str graph_name )
```

Temporarily inspect the specified graph, then swap back when finished.

Parameters

graph_name	The name of a graph in the current database.

6.10.3.24 test_connection()

```
bool test_connection (
              self,
             bool  raise_error = True )
```

Establish a basic connection to the Neo4j database, and test full functionality.

Can be configured to fail silently, which enables retries or external handling.

Parameters

raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.10.4 Member Data Documentation

6.10.4.1 connection_string

```
connection_string
```

6.10.4.2 database_name

```
database_name
```

6.10.4.3 graph_name

```
graph_name
```

6.10.4.4 verbose

```
verbose
```

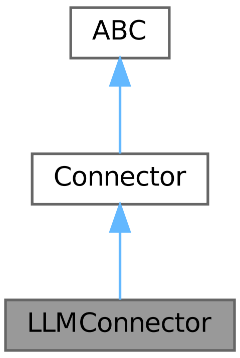
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/fact_storage.py

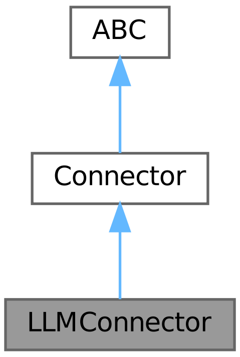
6.11 LLMConnector Class Reference

Connector for prompting and returning LLM output (raw text/JSON) via LangChain.

Inheritance diagram for LLMConnector:



Collaboration diagram for LLMConnector:



Public Member Functions

- __init__ (self, float temperature=0, str system_prompt="You are a helpful assistant.")

  Initialize the connector.
- configure (self)

  Initialize the LangChain LLM using environment credentials.
- test_connection (self)

  Send a trivial prompt to verify LLM connectivity.

- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.
- str execute_full_query (self, str system_prompt, str human_prompt)

  Send a single prompt to the LLM with separate system and human instructions.
- str execute_query (self, str query)

  Send a single prompt through the connection and return raw LLM output.
- str execute_file (self, str filename)

  Run a single prompt from a file.

Public Attributes

- model_name
- temperature
- system_prompt
- llm

6.11.1 Detailed Description

Connector for prompting and returning LLM output (raw text/JSON) via LangChain.

Note

The method components.text_processing.LLMConnector.execute_query simplifies the prompt process.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 __init__()

```
__init__ (
             self,
            float  temperature = 0,
            str  system_prompt = "You are a helpful assistant." )
```

Initialize the connector.

Note

Model name is specified in the .env file.

6.11.3 Member Function Documentation

6.11.3.1 check_connection()

```
bool check_connection (
             self,
            str log_source,
            bool raise_error )
```

Minimal connection test to determine if our connection string is valid.

Parameters

log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.11.3.2 configure()

```
configure (
              self )
```

Initialize the LangChain LLM using environment credentials.

Reads:

- OPENAI_API_KEY from .env for authentication
- LLM_MODEL and LLM_TEMPERATURE to override defaults

Reimplemented from Connector.

6.11.3.3 execute_file()

```
str execute_file (
              self,
             str filename )
```

Run a single prompt from a file.

Reads the entire file as a single string and sends it to execute_query.

Parameters

filename	Path to the prompt file (.txt)

Returns

Raw LLM response as a string.

Reimplemented from Connector.

6.11.3.4 execute_full_query()

```
str execute_full_query (
              self,
             str system_prompt,
             str human_prompt )
```

Send a single prompt to the LLM with separate system and human instructions.

6.11.3.5 execute_query()

```
str execute_query (
              self,
             str query )
```

Send a single prompt through the connection and return raw LLM output.

Parameters

query	A single string prompt to send to the LLM.

Returns

Raw LLM response as a string.

Reimplemented from Connector.

6.11.3.6 test_connection()

```
test_connection (
              self )
```

Send a trivial prompt to verify LLM connectivity.

Returns

Whether the prompt executed successfully.

Reimplemented from Connector.

6.11.4 Member Data Documentation

6.11.4.1 llm

```
llm
```

6.11.4.2 model_name

```
model_name
```

6.11.4.3 system_prompt

```
system_prompt
```

6.11.4.4 temperature

```
temperature
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py

6.12 Log Class Reference

The Log class standardizes console output.

Classes

- class Failure

Static Public Member Functions

- None success (str prefix="PASS", str msg="", bool verbose=True)

  A success message begins with a green prefix.
- None warn (str prefix="PASS", str msg="", bool verbose=True)

  A warning message begins with a yellow prefix.
- None fail (str prefix="ERROR", str msg="", bool raise_error=True, Optional[Exception] other_error=None)

  A failure message begins with a red prefix.
- None success_legacy (str msg="")

  A legacy success message begins with a Green Plus.
- None fail_legacy (str msg="")

  A legacy failure message begins with a Red X.

Static Public Attributes

- bool USE_COLORS = True

  Enable ANSI colors in output.
- str GREEN = "\033[32m"

  ANSI code for green text.
- str RED = "\033[31m"

  ANSI code for red text.
- str YELLOW = "\033[33m"

  ANSI code for yellow text.
- str BRIGHT = "\033[93m"

  ANSI code for bright yellow / cream.
- str WHITE = "\033[0m"

  ANSI code to reset color.
- str SUCCESS_COLOR = GREEN

  ANSI color applied to the prefix of success messages.
- str WARNING_COLOR = YELLOW

  ANSI color applied to the prefix of ignored fail messages.
- str FAILURE_COLOR = RED

  ANSI color applied to the prefix of critical fail messages.
- str MSG_COLOR = BRIGHT

  ANSI color applied to the body of every Log message.
- bool FULL_DF = False

  When printing the results of a query.
- str conn_abc = "BASE CONNECTOR: "
- str db_conn_abc = "CONNECTOR: "
- str rel_db = "REL DB: "
- str gr_db = "GRAPH DB: "
- str doc_db = "DOCS DB: "
- str bad_addr = "BAD ADDRESS: "
- f msg_bad_addr = lambda connection_string"Failed to connect on {connection_string}"
- str bad_path = "FILE NOT FOUND: "
- f msg_bad_path = lambda file_path"Failed to open file '{file_path}'"
- f msg_good_path = lambda file_path"Reading contents of file '{file_path}'"
- f msg_good_exec_f = lambda file_path"Finished executing queries from '{file_path}'"
- f msg_bad_exec_f = lambda file_path"Error occurred while executing queries from '{file_path}'"
- f msg_db_connect = lambda database_name"Successfully connected to database: {database_name}"
- str good_val = "VALID RESULT: "
- str bad_val = "INCORRECT RESULT: "
- f msg_compare = lambda observed, expected"Expected {expected}, got {observed}"
- tuple msg_result
- tuple msg_good_table
- tuple msg_good_coll
- tuple msg_good_graph
- f msg_bad_table = lambda name"Table '{name}' not found"
- f msg_bad_coll = lambda name"Collection '{name}' not found"
- f msg_bad_graph = lambda name"Graph '{name}' not found"
- str test_conn = "CONNECTION TEST: "
- str test_basic = "BASIC: "
- str test_info = "DB INFO: "
- str test_df = "GET DF: "
- str test_tmp_db = "CREATE DB: "
- str msg_unknown_error = "An unhandled error occurred."

- str get_df = "GET_DF: "
- str create_db = "CREATE_DB: "
- str drop_db = "DROP_DB: "
- str run_q = "QUERY: "
- str run_f = "FILE EXEC: "
- str drop_gr = "DROP_GRAPH: "
- f msg_success_managed_db = lambda managed, database_name"Successfully {managed} database '{database_name}'"
- tuple msg_fail_manage_db
- f msg_success_managed_gr = lambda managed, database_name"Successfully {managed} graph '{database_name}'"
- f msg_fail_manage_gr = lambda manage, database_name, connection_string"Failed to {manage} graph '{database_name}' on connection {connection_string}"
- f msg_fail_parse = lambda alias, bad_value, expected_type"Could not convert {alias} with value {bad_value} to type {expected_type}"
- tuple msg_multiple_query
- f msg_good_exec_q = lambda query"Executed successfully:\n'{query}'"
- f msg_good_exec_qr = lambda query, results"Executed successfully:\n'{query}'\n{Log.msg_result(results)}"
- f msg_bad_exec_q = lambda query"Failed to execute query:\n'{query}'"
- str kg = "KG: "
- str pytest_db = "PYTEST (DB): "
- str db_exists = "DB_EXIST: "
- f msg_db_exists = lambda database_name"Database '{database_name}' already exists."
- f msg_db_not_found = lambda database_name, connection_string"Could not find database '{database_↩ name}' using connection '{connection_string}'"
- f msg_db_current = lambda database_name"Cannot drop database '{database_name}' while connected to it!"
- str swap_db = "SWAP_DB: "
- str swap_kg = "SWAP_GRAPH: "
- f msg_swap_db = lambda old_db, new_db"Switched from database '{old_db}' to database '{new_db}'"
- f msg_swap_kg = lambda old_kg, new_kg"Switched from graph '{old_kg}' to graph '{new_kg}'"
- str get_unique = "UNIQUE: "

6.12.1 Detailed Description

The Log class standardizes console output.

6.12.2 Member Function Documentation

6.12.2.1 fail()

```
None fail (
             str  prefix = "ERROR",
             str  msg = "",
             bool  raise_error = True,
             Optional[Exception]  other_error = None )  [static]
```

A failure message begins with a red prefix.

Parameters

prefix	The context of the message.
msg	The message to print.
raise_error	Whether to raise an error.
other_error	Another Exception resulting from this failure.

Exceptions

Log.Failure	If raise_error is True

6.12.2.2 fail_legacy()

```
None fail_legacy (
             str  msg = "" )  [static]
```

A legacy failure message begins with a Red X.

Parameters

msg	The message to print.

6.12.2.3 success()

```
None success (
             str  prefix = "PASS",
             str  msg = "",
             bool  verbose = True )  [static]
```

A success message begins with a green prefix.

Parameters

prefix	The context of the message.
msg	The message to print.
verbose	Whether to actually print. Saves space and reduces nested if statements.

6.12.2.4 success_legacy()

```
None success_legacy (
             str  msg = "" )  [static]
```

A legacy success message begins with a Green Plus.

Parameters

msg	The message to print.

6.12.2.5 warn()

```
None warn (
             str  prefix = "PASS",
             str  msg = "",
             bool  verbose = True )  [static]
```

A warning message begins with a yellow prefix.

Parameters

prefix	The context of the message.
msg	The message to print.
verbose	Whether to actually print. Saves space and reduces nested if statements.

6.12.3 Member Data Documentation

6.12.3.1 bad_addr

```
str bad_addr = "BAD ADDRESS: "  [static]
```

6.12.3.2 bad_path

```
str bad_path = "FILE NOT FOUND: "  [static]
```

6.12.3.3 bad_val

```
str bad_val = "INCORRECT RESULT: "  [static]
```

6.12.3.4 BRIGHT

```
str BRIGHT = "\033[93m"  [static]
```

ANSI code for bright yellow / cream.

6.12.3.5 conn_abc

```
str conn_abc = "BASE CONNECTOR: "  [static]
```

6.12.3.6 create_db

```
str create_db = "CREATE_DB: "  [static]
```

6.12.3.7 db_conn_abc

```
str db_conn_abc = "CONNECTOR: "  [static]
```

6.12.3.8 db_exists

```
str db_exists = "DB_EXIST: " [static]
```

6.12.3.9 doc_db

```
str doc_db = "DOCS DB: " [static]
```

6.12.3.10 drop_db

```
str drop_db = "DROP_DB: " [static]
```

6.12.3.11 drop_gr

```
str drop_gr = "DROP_GRAPH: " [static]
```

6.12.3.12 FAILURE_COLOR

```
str FAILURE_COLOR = RED [static]
```

ANSI color applied to the prefix of critical fail messages.

6.12.3.13 FULL_DF

```
bool FULL_DF = False [static]
```

When printing the results of a query.

6.12.3.14 get_df

```
str get_df = "GET_DF: " [static]
```

6.12.3.15 get_unique

```
str get_unique = "UNIQUE: " [static]
```

6.12.3.16 good_val

```
str good_val = "VALID RESULT: " [static]
```

6.12.3.17 gr_db

```
str gr_db = "GRAPH DB: "  [static]
```

6.12.3.18 GREEN

```
str GREEN = "\033[32m"  [static]
```

ANSI code for green text.

6.12.3.19 kg

```
str kg = "KG: "  [static]
```

6.12.3.20 msg_bad_addr

```
f msg_bad_addr = lambda connection_string"Failed to connect on {connection_string}"  [static]
```

6.12.3.21 msg_bad_coll

```
f msg_bad_coll = lambda name"Collection '{name}' not found"  [static]
```

6.12.3.22 msg_bad_exec_f

```
f msg_bad_exec_f = lambda file_path"Error occurred while executing queries from '{file_path}'"
[static]
```

6.12.3.23 msg_bad_exec_q

```
f msg_bad_exec_q = lambda query"Failed to execute query:\n'{query}'"  [static]
```

6.12.3.24 msg_bad_graph

```
f msg_bad_graph = lambda name"Graph '{name}' not found"  [static]
```

6.12.3.25 msg_bad_path

```
f msg_bad_path = lambda file_path"Failed to open file '{file_path}'"  [static]
```

6.12.3.26 msg_bad_table

```
f msg_bad_table = lambda name"Table '{name}' not found"  [static]
```

6.12.3.27 MSG_COLOR

```
str MSG_COLOR = BRIGHT  [static]
```

ANSI color applied to the body of every Log message.

6.12.3.28 msg_compare

```
f msg_compare = lambda observed, expected"Expected {expected}, got {observed}"  [static]
```

6.12.3.29 msg_db_connect

```
f msg_db_connect = lambda database_name"Successfully connected to database:  {database_name}"
[static]
```

6.12.3.30 msg_db_current

```
f msg_db_current = lambda database_name"Cannot drop database '{database_name}' while connected
to it!"  [static]
```

6.12.3.31 msg_db_exists

```
f msg_db_exists = lambda database_name"Database '{database_name}' already exists."  [static]
```

6.12.3.32 msg_db_not_found

```
f msg_db_not_found = lambda database_name, connection_string"Could not find database '{database↵
_name}' using connection '{connection_string}'"  [static]
```

6.12.3.33 msg_fail_manage_db

```
tuple msg_fail_manage_db  [static]
```

Initial value:

```
=  (
        lambda manage, database_name, connection_string: f"Failed to {manage} database '{database_name}' on
      connection {connection_string}"
    )
```

6.12.3.34 msg_fail_manage_gr

```
f msg_fail_manage_gr = lambda manage, database_name, connection_string"Failed to {manage}
graph '{database_name}' on connection {connection_string}"  [static]
```

6.12.3.35 msg_fail_parse

```
f msg_fail_parse = lambda alias, bad_value, expected_type"Could not convert {alias} with value
{bad_value} to type {expected_type}"  [static]
```

6.12.3.36 msg_good_coll

```
tuple msg_good_coll  [static]
```

Initial value:

```
=  (
        lambda name, df: f
    )
```

6.12.3.37 msg_good_exec_f

```
f msg_good_exec_f = lambda file_path"Finished executing queries from '{file_path}'"  [static]
```

6.12.3.38 msg_good_exec_q

```
f msg_good_exec_q = lambda query"Executed successfully:\n'{query}'"  [static]
```

6.12.3.39 msg_good_exec_qr

```
f msg_good_exec_qr = lambda query, results"Executed successfully:\n'{query}'\n{Log.msg_result(results)}"
[static]
```

6.12.3.40 msg_good_graph

```
tuple msg_good_graph  [static]
```

Initial value:

```
=  (
        lambda name, df: f
    )
```

6.12.3.41 msg_good_path

```
f msg_good_path = lambda file_path"Reading contents of file '{file_path}'"  [static]
```

6.12.3.42 msg_good_table

```
tuple msg_good_table  [static]
```

Initial value:

```
=  (
        lambda name, df: f
    )
```

6.12.3.43 msg_multiple_query

```
tuple msg_multiple_query  [static]
```

Initial value:

```
=  (
        lambda n_queries, query: f"A combined query ({n_queries} results) was executed as a single query.
      Extra results were discarded. Query:\n{query}"
    )
```

6.12.3.44 msg_result

```
tuple msg_result  [static]
```

Initial value:

```
=  (
        lambda results: f
    )
```

6.12.3.45 msg_success_managed_db

```
f msg_success_managed_db = lambda managed, database_name"Successfully {managed} database '{database←
_name}'"  [static]
```

6.12.3.46 msg_success_managed_gr

```
f msg_success_managed_gr = lambda managed, database_name"Successfully {managed} graph '{database←
_name}'"  [static]
```

6.12.3.47 msg_swap_db

```
f msg_swap_db = lambda old_db, new_db"Switched from database '{old_db}' to database '{new_←
db}'"  [static]
```

6.12.3.48 msg_swap_kg

```
f msg_swap_kg = lambda old_kg, new_kg"Switched from graph '{old_kg}' to graph '{new_kg}'"
[static]
```

6.12.3.49 msg_unknown_error

```
str msg_unknown_error = "An unhandled error occurred."  [static]
```

6.12.3.50 pytest_db

```
str pytest_db = "PYTEST (DB): "  [static]
```

6.12.3.51 RED

```
str RED = "\033[31m"  [static]
```

ANSI code for red text.

6.12.3.52 rel_db

```
str rel_db = "REL DB: "  [static]
```

6.12.3.53 run_f

```
str run_f = "FILE EXEC: "  [static]
```

6.12.3.54 run_q

```
str run_q = "QUERY: "  [static]
```

6.12.3.55 SUCCESS_COLOR

```
str SUCCESS_COLOR = GREEN  [static]
```

ANSI color applied to the prefix of success messages.

6.12.3.56 swap_db

```
str swap_db = "SWAP_DB: "  [static]
```

6.12.3.57 swap_kg

```
str swap_kg = "SWAP_GRAPH: "  [static]
```

6.12.3.58 test_basic

```
str test_basic = "BASIC: "  [static]
```

6.12.3.59 test_conn

```
str test_conn = "CONNECTION TEST: "  [static]
```

6.12.3.60 test_df

```
str test_df = "GET DF: "  [static]
```

6.12.3.61 test_info

```
str test_info = "DB INFO: "  [static]
```

6.12.3.62 test_tmp_db

```
str test_tmp_db = "CREATE DB: "  [static]
```

6.12.3.63 USE_COLORS

```
bool USE_COLORS = True  [static]
```

Enable ANSI colors in output.

6.12.3.64 WARNING_COLOR

```
str WARNING_COLOR = YELLOW  [static]
```

ANSI color applied to the prefix of ignored fail messages.

6.12.3.65 WHITE

```
str WHITE = "\033[0m"  [static]
```

ANSI code to reset color.

6.12.3.66 YELLOW

```
str YELLOW = "\033[33m"  [static]
```

ANSI code for yellow text.

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/util.py

6.13 Metrics Class Reference

Utility class for computing and posting evaluation metrics.

Public Member Functions

- None __init__ (self)
- bool post_payload (self, Dict[str, Any] payload)

    POST directly to Blazor (soon to be deprecated)
- None post_basic_metrics (self, str book_id, str book_title, str summary, str gold_summary="", str text="", ∗∗Any kwargs)

    POST basic evaluation scores to Blazor (ROUGE, BERTScore).
- None post_basic_output (self, str book_id, str book_title, str summary)

    POST dummy date to Blazor.

Static Public Member Functions

- Dict[str, Any] compute_basic_metrics (str summary, str gold_summary, str chunk)

    Compute ROUGE and BERTScore.
- Dict[str, Any] create_summary_payload (str book_id, str book_title, str summary, str gold_summary, Dict[str, Any] metrics=None)

    Create the full Blazor payload for a single book.
- Dict[str, Any] generate_default_metrics (float rouge1_f1=0.0, float rouge2_f1=0.0, float rougeL_↩ f1=0.0, float rougeLsum_f1=0.0, float bert_precision=0.0, float bert_recall=0.0, float bert_f1=0.↩ 0, float booook_score=0.0, float questeval_score=0.0, str qa_question1="UNKNOWN", str qa_↩ gold1="UNKNOWN", str qa_generated1="UNKNOWN", bool qa_correct1=False, float qa_accuracy1=0.0, str qa_question2="UNKNOWN", str qa_gold2="UNKNOWN", str qa_generated2="UNKNOWN", bool qa_↩ correct2=False, float qa_accuracy2=0.0)

    Generate the metrics sub-payload with customizable default values.
- Dict[str, Any] generate_example_metrics ()

    Create a placeholder payload with dummy values.

Public Attributes

- HOST
- PORT
- url

Static Public Attributes

- int timeout_seconds = 900

6.13.1 Detailed Description

Utility class for computing and posting evaluation metrics.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 __init__()

```
None __init__ (
              self )
```

6.13.3 Member Function Documentation

6.13.3.1 compute_basic_metrics()

```
Dict[str, Any] compute_basic_metrics (
            str summary,
            str gold_summary,
            str chunk )  [static]
```

Compute ROUGE and BERTScore.

Parameters

summary	A text string containing a book summary
gold_summary	A summary to compare against
chunk	The original text of the chunk.

Returns

Dict containing 'rouge' and 'bertscore' keys. Scores are nested with inconsistent schema.

6.13.3.2 create_summary_payload()

```
Dict[str, Any] create_summary_payload (
            str book_id,
            str book_title,
            str summary,
            str gold_summary,
            Dict[str, Any]  metrics = None )  [static]
```

Create the full Blazor payload for a single book.

Parameters

book_id	Unique identifier for one book.
book_title	String containing the title of a book.
summary	String containing a book summary.
gold_summary	Optional summary to compare against.
metrics	Dictionary containing various nested evaluation metrics.

Returns

A dictionary with C#-style key names.

6.13.3.3 generate_default_metrics()

```
Dict[str, Any] generate_default_metrics (
            float  rouge1_f1 = 0.0,
            float  rouge2_f1 = 0.0,
            float  rougeL_f1 = 0.0,
            float  rougeLsum_f1 = 0.0,
            float  bert_precision = 0.0,
            float  bert_recall = 0.0,
            float  bert_f1 = 0.0,
            float  booook_score = 0.0,
            float  questeval_score = 0.0,
            str  qa_question1 = "UNKNOWN",
            str  qa_gold1 = "UNKNOWN",
            str  qa_generated1 = "UNKNOWN",
            bool  qa_correct1 = False,
            float  qa_accuracy1 = 0.0,
            str  qa_question2 = "UNKNOWN",
            str  qa_gold2 = "UNKNOWN",
            str  qa_generated2 = "UNKNOWN",
            bool  qa_correct2 = False,
            float  qa_accuracy2 = 0.0 )  [static]
```

Generate the metrics sub-payload with customizable default values.

Parameters

rouge1_f1	The ROUGE-1 evaluation metric.
rouge2_f1	The ROUGE-2 evaluation metric.
rougeL_f1	The ROUGE-L evaluation metric.
rougeLsum_f1	The ROUGE-Lsum evaluation metric.
bert_precision	The BERTScore precision score.
bert_recall	The BERTScore recall score.
bert_f1	The BERTScore F1 score.
booook_score	The BooookScore evaluation metric.
questeval_score	The QuestEval evaluation metric.
qa_question1	A question about the book.
qa_gold1	The correct answer to the question.
qa_generated1	A generated answer to judge.
qa_correct1	Whether our answer is correct.
qa_accuracy1	The accuracy score for this QA sample.
qa_question2	A question about the book.
qa_gold2	The correct answer to the question.
qa_generated2	A generated answer to judge.
qa_correct2	Whether our answer is correct.
qa_accuracy2	The accuracy score for this QA sample.

Returns

Dictionary containing various nested evaluation metrics.

6.13.3.4 generate_example_metrics()

```
Dict[str, Any] generate_example_metrics ( )  [static]
```

Create a placeholder payload with dummy values.

Returns

Full payload with nested metrics.

6.13.3.5 post_basic_metrics()

```
None post_basic_metrics (
              self,
             str book_id,
             str book_title,
             str summary,
             str  gold_summary = "",
             str  text = "",
             **Any kwargs )
```

POST basic evaluation scores to Blazor (ROUGE, BERTScore).

Parameters

book_id	Unique identifier for one book.
book_title	String containing the title of a book.
summary	String containing a book summary.
gold_summary	Optional summary to compare against.
text	A string containing text from the book.
kwargs	Any additional named arguments will be added to the payload.

6.13.3.6 post_basic_output()

```
None post_basic_output (
              self,
             str book_id,
             str book_title,
             str summary )
```

POST dummy date to Blazor.

Parameters

book_id	Unique identifier for one book.
book_title	String containing the title of a book.
summary	String containing a book summary.

6.13.3.7 post_payload()

```
bool post_payload (
              self,
             Dict[str, Any] payload )
```

POST directly to Blazor (soon to be deprecated)

Verify and POST a given payload using the requests API.

Parameters

payload	JSON dictionary containing data for a single book.

Returns

Whether the POST operation was successful.

6.13.4 Member Data Documentation

6.13.4.1 HOST

```
HOST
```

6.13.4.2 PORT

```
PORT
```

6.13.4.3 timeout_seconds

```
int timeout_seconds = 900  [static]
```

6.13.4.4 url

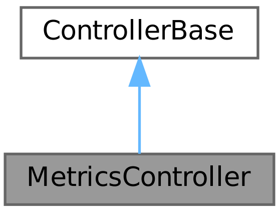
```
url
```

The documentation for this class was generated from the following file:

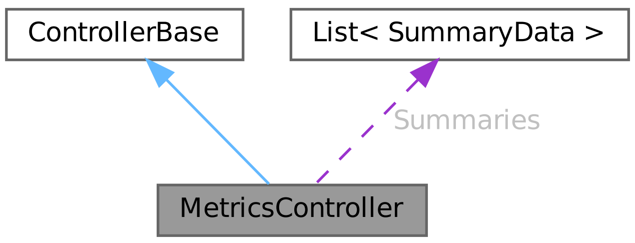
- /home/runner/work/dsci-capstone/dsci-capstone/components/metrics.py

6.14 MetricsController Class Reference

Inheritance diagram for MetricsController:



Collaboration diagram for MetricsController:



Public Member Functions

- MetricsController (ILogger< MetricsController > logger, IHubContext< MetricsHub > hubContext)
- async Task< IActionResult > Post ([FromBody] SummaryData summary)
- IActionResult GetIndex (int id)
- IActionResult GetAll ()

Private Attributes

- readonly ILogger< MetricsController > _logger
- readonly IHubContext< MetricsHub > _hubContext

Static Private Attributes

- static readonly List< SummaryData > Summaries = new()

6.14.1 Constructor & Destructor Documentation

6.14.1.1 MetricsController()

```
MetricsController (
            ILogger< MetricsController > logger,
            IHubContext< MetricsHub > hubContext )
```

6.14.2 Member Function Documentation

6.14.2.1 GetAll()

```
IActionResult GetAll ( )
```

6.14.2.2 GetIndex()

```
IActionResult GetIndex (
            int id )
```

6.14.2.3 Post()

```
async Task< IActionResult > Post (
            [FromBody] SummaryData summary )
```

6.14.3 Member Data Documentation

6.14.3.1 _hubContext

```
readonly IHubContext<MetricsHub> _hubContext  [private]
```

6.14.3.2 _logger

```
readonly ILogger<MetricsController> _logger  [private]
```

6.14.3.3 Summaries

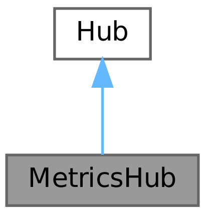
```
readonly List<SummaryData> Summaries = new()  [static], [private]
```

The documentation for this class was generated from the following file:

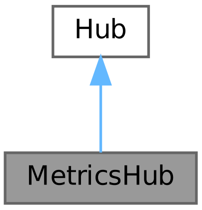
- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/MetricsController.cs

6.15 MetricsHub Class Reference

Inheritance diagram for MetricsHub:



Collaboration diagram for MetricsHub:



Public Member Functions

- MetricsHub (ILogger< MetricsHub >? logger=null)
- override async Task OnConnectedAsync ()
- override async Task OnDisconnectedAsync (Exception? exception)

Private Attributes

- readonly? ILogger< MetricsHub > _logger

6.15.1 Constructor & Destructor Documentation

6.15.1.1 MetricsHub()

```
MetricsHub (
            ILogger< MetricsHub >?  logger = null )
```

6.15.2 Member Function Documentation

6.15.2.1 OnConnectedAsync()

```
override async Task OnConnectedAsync ( )
```

6.15.2.2 OnDisconnectedAsync()

```
override async Task OnDisconnectedAsync (
            Exception?  exception )
```

6.15.3 Member Data Documentation

6.15.3.1 _logger

```
readonly?  ILogger<MetricsHub> _logger  [private]
```

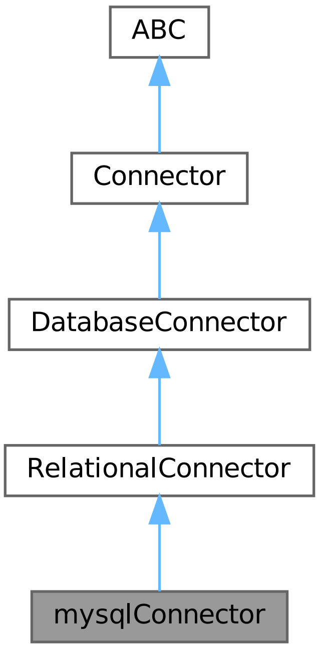
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Hubs/MetricsHub.cs

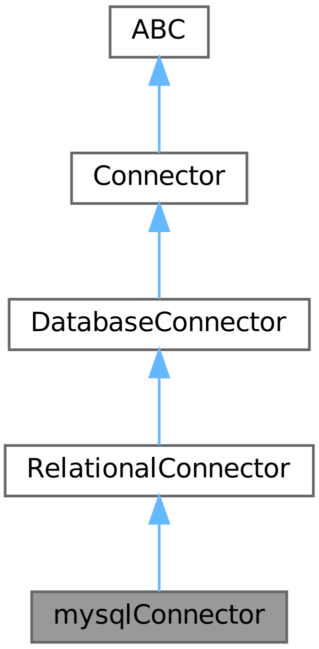
6.16 mysqlConnector Class Reference

A relational database connector configured for MySQL.

Inheritance diagram for mysqlConnector:



Collaboration diagram for mysqlConnector:



Public Member Functions

- None __init__ (self, bool verbose=False)

    Configures the relational connector.

Public Member Functions inherited from RelationalConnector

- "RelationalConnector" from_env (cls, bool verbose=False)

    Decides what type of relational connector to create using the .env file.
- None change_database (self, str new_database)

    Update the connection URI to reference a different database in the same engine.
- bool test_connection (self, bool raise_error=True)

    Establish a basic connection to the database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

    Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] execute_query (self, str query)

    Send a single command to the database connection.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

    Automatically generate and run a query for the specified table using SQLAlchemy.
- None create_database (self, str database_name)

    Use the current database connection to create a sibling database in this engine.
- None drop_database (self, str database_name="")

    Delete all data stored in a particular database.
- bool database_exists (self, str database_name)

    Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.

Static Public Attributes

- dict specific_queries

Additional Inherited Members

Public Attributes inherited from RelationalConnector

- database_name
- verbose
- connection_string
- db_type

Public Attributes inherited from DatabaseConnector

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions inherited from RelationalConnector

- List[str] _split_combined (self, str multi_query)

  Divides a string into non-divisible SQL queries using `sqlparse`.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

6.16.1 Detailed Description

A relational database connector configured for MySQL.

Note

Should be hidden from the user using a factory method.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 __init__()

```
None __init__ (
                self,
              bool   verbose = False )
```

Configures the relational connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from RelationalConnector.

6.16.3 Member Data Documentation

6.16.3.1 specific_queries

```
dict specific_queries  [static]
```

Initial value:

```
=  {
        "MYSQL": [
            "SELECT DATABASE();",  # Single value, name of the current database.
            "SHOW DATABASES;",  # List of databases the secondary user can access.
        ]  # List of all databases in the database engine.
    }
```

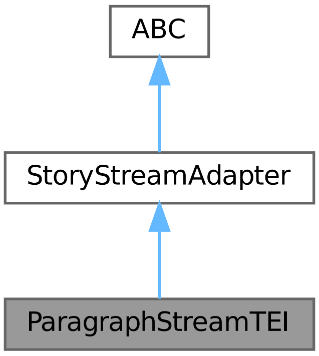
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py

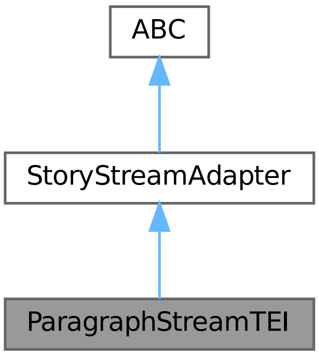
6.17 ParagraphStreamTEI Class Reference

Streams paragraphs from a TEI file as Chunk objects.

Inheritance diagram for ParagraphStreamTEI:



Collaboration diagram for ParagraphStreamTEI:



Public Member Functions

- None __init__ (self, str tei_path, int book_id, int story_id, list[str] allowed_chapters=None, str start_inclusive="", str end_inclusive="")

  Create a ParagraphStreamTEI object.
- Iterator[Chunk] stream_segments (self)

  Yields sanitized parts of a book.
- List[Chunk] pre_compute_segments (self)

  Splits the target book into paragraphs.

Public Member Functions inherited from StoryStreamAdapter

- Iterator[Chunk] stream_paragraphs (self)

    Concrete helper method to split segments into paragraphs.
- Iterator[str] stream_sentences (self)

    Concrete helper method to split paragraphs into sentences.

Public Attributes

- tei_path
- book_id
- story_id
- allowed_chapters
- start_inclusive
- end_inclusive
- lines
- root
- chunks
- xml_namespace

Static Public Attributes

- dict xml_namespace = {"tei": "http://www.tei-c.org/ns/1.0"}
- str encoding = "utf-8"

6.17.1 Detailed Description

Streams paragraphs from a TEI file as Chunk objects.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 __init__()

```
None __init__ (
              self,
             str tei_path,
             int book_id,
             int story_id,
             list[str]  allowed_chapters = None,
             str  start_inclusive = "",
             str  end_inclusive = "" )
```

Create a ParagraphStreamTEI object.

Parameters

tei_path	Path to an existing TEI XML file.
book_id	ID for this book.
story_id	ID for this story (may be same as book_id).
allowed_chapters	A list of valid chapter titles. Must exactly match the contents of head.
start_inclusive	(Optional) Unique string representing the start of the book.
end_inclusive	(Optional) Unique string representing the end of the book.

6.17.3 Member Function Documentation

6.17.3.1 pre_compute_segments()

```
List[Chunk] pre_compute_segments (
              self )
```

Splits the target book into paragraphs.

Yields Chunk objects for each paragraph (

) in the TEI file. Uses etree Element.sourceline to approximate start/end line in TEI. Supports optional start_inclusive / end_inclusive boundaries to slice text and stop iteration. Computes progress percentages using character counts:

- story_percent: progress through the entire story
- chapter_percent: progress through the current chapter Populates self.chunks so they can be streamed as requested by interface

6.17.3.2 stream_segments()

```
Iterator[Chunk] stream_segments (
              self )
```

Yields sanitized parts of a book.

- Story segments usually correspond to chapters.
- They serve as borders between chunking operations, ensuring chunks do not span multiple chapters. Implementation is handled by child classes BookStream, etc.
- Segments should be pre-cleaned and must contain 1 paragraph per line with all other newlines removed.

Reimplemented from StoryStreamAdapter.

6.17.4 Member Data Documentation

6.17.4.1 allowed_chapters

```
allowed_chapters
```

6.17.4.2 book_id

```
book_id
```

6.17.4.3 chunks

```
chunks
```

6.17.4.4 encoding

```
str encoding = "utf-8"  [static]
```

6.17.4.5 end_inclusive

```
end_inclusive
```

6.17.4.6 lines

```
lines
```

6.17.4.7 root

```
root
```

6.17.4.8 start_inclusive

```
start_inclusive
```

6.17.4.9 story_id

```
story_id
```

6.17.4.10 tei_path

```
tei_path
```

6.17.4.11 xml_namespace [1/2]

```
dict xml_namespace = {"tei":  "http://www.tei-c.org/ns/1.0"}  [static]
```

6.17.4.12 xml_namespace [2/2]

```
xml_namespace
```

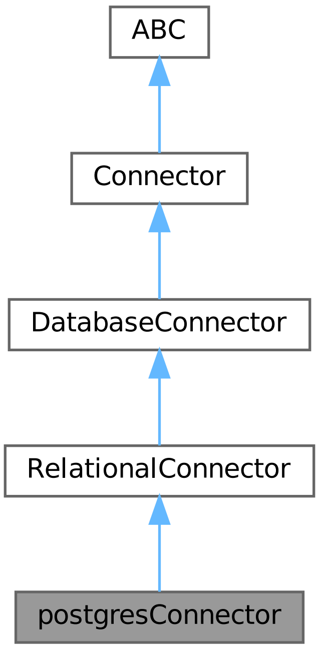
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

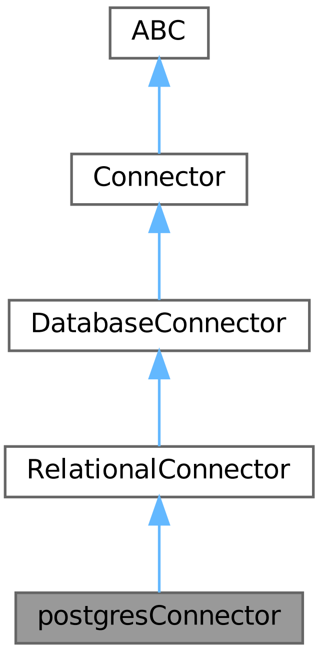
6.18 postgresConnector Class Reference

A relational database connector configured for PostgreSQL.

Inheritance diagram for postgresConnector:



Collaboration diagram for postgresConnector:



Public Member Functions

- None __init__ (self, bool verbose=False)

    Configures the relational connector.

Public Member Functions inherited from RelationalConnector

- "RelationalConnector" from_env (cls, bool verbose=False)

    Decides what type of relational connector to create using the .env file.
- None change_database (self, str new_database)

    Update the connection URI to reference a different database in the same engine.
- bool test_connection (self, bool raise_error=True)

    Establish a basic connection to the database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

    Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] execute_query (self, str query)

    Send a single command to the database connection.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

    Automatically generate and run a query for the specified table using SQLAlchemy.
- None create_database (self, str database_name)

    Use the current database connection to create a sibling database in this engine.
- None drop_database (self, str database_name="")

    Delete all data stored in a particular database.
- bool database_exists (self, str database_name)

    Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.

Static Public Attributes

- dict specific_queries

Additional Inherited Members

Public Attributes inherited from RelationalConnector

- database_name
- verbose
- connection_string
- db_type

Public Attributes inherited from DatabaseConnector

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions inherited from RelationalConnector

- List[str] _split_combined (self, str multi_query)

  Divides a string into non-divisible SQL queries using `sqlparse`.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

6.18.1 Detailed Description

A relational database connector configured for PostgreSQL.

Note

Should be hidden from the user using a factory method.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 __init__()

```
None __init__ (
               self,
             bool  verbose = False )
```

Configures the relational connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from RelationalConnector.

6.18.3 Member Data Documentation

6.18.3.1 specific_queries

```
dict specific_queries  [static]
```

Initial value:

```
=  {
        "POSTGRES": [
            "SELECT current_database();",  # Single value, name of the current database.
            "SELECT datname FROM pg_database;",  # List of ALL databases, even ones we cannot access.
        ]  # List of all databases in the database engine.
    }
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py

6.19 PRF1Metric Class Reference

Properties

- string Name [get, set]
- double Precision [get, set]
- double Recall [get, set]
- double F1Score [get, set]

6.19.1 Property Documentation

6.19.1.1 F1Score

```
double F1Score [get], [set]
```

6.19.1.2 Name

```
string Name [get], [set]
```

6.19.1.3 Precision

```
double Precision [get], [set]
```

6.19.1.4 Recall

```
double Recall [get], [set]
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/PRF1Metric.cs

6.20 QAItem Class Reference

Properties

- string Question `[get, set]`
- string GoldAnswer `[get, set]`
- string GeneratedAnswer `[get, set]`
- bool? IsCorrect `[get, set]`
- double? Accuracy `[get, set]`

6.20.1 Property Documentation

6.20.1.1 Accuracy

```
double? Accuracy [get], [set]
```

6.20.1.2 GeneratedAnswer

```
string GeneratedAnswer [get], [set]
```

6.20.1.3 GoldAnswer

```
string GoldAnswer  [get], [set]
```

6.20.1.4 IsCorrect

```
bool?  IsCorrect  [get], [set]
```

6.20.1.5 Question

```
string Question  [get], [set]
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAItem.cs

6.21 QAMetric Class Reference

Properties

- List< QAItem > QAItems = new() `[get, set]`
- double AverageAccuracy `[get]`

6.21.1 Property Documentation

6.21.1.1 AverageAccuracy

```
double AverageAccuracy  [get]
```

6.21.1.2 QAItems

```
List<QAItem> QAItems = new()  [get], [set]
```

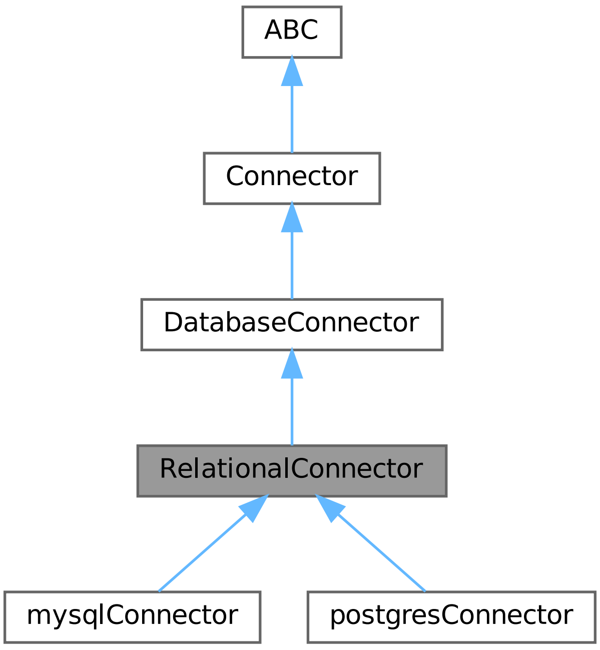
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAMetric.cs

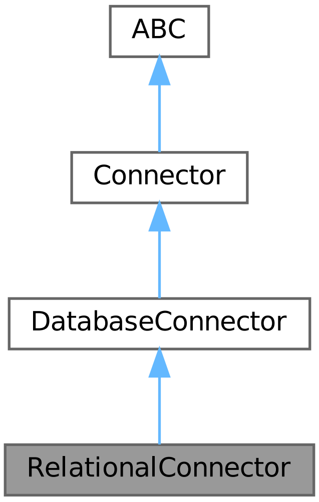
6.22 RelationalConnector Class Reference

Connector for relational databases (MySQL, PostgreSQL).

Inheritance diagram for RelationalConnector:



Collaboration diagram for RelationalConnector:



Public Member Functions

- None __init__ (self, bool verbose, List[str] specific_queries)

  Creates a new database connector.
- "RelationalConnector" from_env (cls, bool verbose=False)

  Decides what type of relational connector to create using the .env file.
- None change_database (self, str new_database)

  Update the connection URI to reference a different database in the same engine.
- bool test_connection (self, bool raise_error=True)

  Establish a basic connection to the database, and test full functionality.
- bool check_connection (self, str log_source, bool raise_error)

  Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] execute_query (self, str query)

  Send a single command to the database connection.
- Optional[DataFrame] get_dataframe (self, str name, List[str] columns=[ ])

  Automatically generate and run a query for the specified table using SQLAlchemy.
- None create_database (self, str database_name)

  Use the current database connection to create a sibling database in this engine.
- None drop_database (self, str database_name="")

  Delete all data stored in a particular database.
- bool database_exists (self, str database_name)

  Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None configure (self, str DB, str database_name)

  Read connection settings from the .env file.
- Generator[None, None, None] temp_database (self, str database_name)

  Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] execute_combined (self, str multi_query)

  Run several database commands in sequence.
- List[Optional[DataFrame]] execute_file (self, str filename)

  Run several database commands from a file.

Public Attributes

- database_name
- verbose
- connection_string
- db_type

Public Attributes inherited from DatabaseConnector

- verbose

  Whether to print debug messages.
- db_type
- db_engine
- username
- password
- host
- port
- connection_string

Protected Member Functions

- List[str] _split_combined (self, str multi_query)

  Divides a string into non-divisible SQL queries using `sqlparse`.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

6.22.1 Detailed Description

Connector for relational databases (MySQL, PostgreSQL).

Uses SQLAlchemy to abstract complex database operations. Hard-coded queries are used for testing purposes, and depend on the specific engine.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 __init__()

```
None __init__ (
                 self,
                bool verbose,
                List[str] specific_queries )
```

Creates a new database connector.

Use components.connectors.RelationalConnector.from_env instead (this is called by derived classes).

Parameters

verbose	Whether to print success and failure messages.
specific_queries	A list of helpful SQL queries.

Reimplemented from DatabaseConnector.

Reimplemented in mysqlConnector, and postgresConnector.

6.22.3 Member Function Documentation

6.22.3.1 _split_combined()

```
List[str] _split_combined (
                 self,
                str multi_query )  [protected]
```

Divides a string into non-divisible SQL queries using `sqlparse`.

Parameters

multi_query	A string containing multiple queries.

Returns

A list of single-query strings.

Reimplemented from DatabaseConnector.

6.22.3.2 change_database()

```
None change_database (
                 self,
                str new_database )
```

Update the connection URI to reference a different database in the same engine.

Parameters

new_database	The name of the database to connect to.

Reimplemented from DatabaseConnector.

6.22.3.3 check_connection()

```
bool check_connection (
              self,
             str log_source,
             bool raise_error )
```

Minimal connection test to determine if our connection string is valid.

Connect to our relational database using SQLAlchemy's engine.begin()

Parameters

log_source	The Log class prefix indicating which method is performing the check.
raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.22.3.4 create_database()

```
None create_database (
              self,
             str database_name )
```

Use the current database connection to create a sibling database in this engine.

Parameters

database_name	The name of the new database to create.

Exceptions

Log.Failure	If we fail to create the requested database for any reason.

Reimplemented from DatabaseConnector.

6.22.3.5 database_exists()

```
bool database_exists (
            self,
            str database_name )
```

Search for an existing database using the provided name.

Parameters

database_name	The name of a database to search for.

Returns

Whether the database is visible to this connector.

Reimplemented from DatabaseConnector.

6.22.3.6 drop_database()

```
None drop_database (
            self,
            str  database_name = "" )
```

Delete all data stored in a particular database.

Parameters

database_name	The name of an existing database.

Exceptions

Log.Failure	If we fail to drop the target database for any reason.

Reimplemented from DatabaseConnector.

6.22.3.7 execute_query()

```
Optional[DataFrame] execute_query (
            self,
            str query )
```

Send a single command to the database connection.

Note

If a result is returned, it will be converted to a DataFrame.

Parameters

query	A single query to perform on the database.

Returns

DataFrame containing the result of the query, or None

Exceptions

Log.Failure	If the query fails to execute.

Reimplemented from DatabaseConnector.

6.22.3.8 from_env()

```
"RelationalConnector" from_env (
              cls,
             bool   verbose = False )
```

Decides what type of relational connector to create using the .env file.

Parameters

verbose	Whether to print success and failure messages.

Exceptions

Log.Failure	If the .env file contains an invalid DB_ENGINE value.

6.22.3.9 get_dataframe()

```
Optional[DataFrame] get_dataframe (
              self,
             str name,
             List[str]   columns = [] )
```

Automatically generate and run a query for the specified table using SQLAlchemy.

Parameters

name	The name of an existing table or collection in the database.
columns	A list of column names to keep.

Returns

Sorted DataFrame containing the requested data, or None

Exceptions

Log.Failure	If we fail to create the requested DataFrame for any reason.

Reimplemented from DatabaseConnector.

6.22.3.10 test_connection()

```
bool test_connection (
              self,
             bool   raise_error = True )
```

Establish a basic connection to the database, and test full functionality.

Can be configured to fail silently, which enables retries or external handling.

Parameters

raise_error	Whether to raise an error on connection failure.

Returns

Whether the connection test was successful.

Exceptions

Log.Failure	If raise_error is True and the connection test fails to complete.

Reimplemented from Connector.

6.22.4 Member Data Documentation

6.22.4.1 connection_string

```
connection_string
```

6.22.4.2 database_name

```
database_name
```

6.22.4.3 db_type

```
db_type
```

6.22.4.4 verbose

```
verbose
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py

6.23 RelationExtractor Class Reference

Public Member Functions

- __init__ (self, model_name="Babelscape/rebel-large", max_tokens=1024)
- extract (self, str text, bool parse_tuples=False)

Public Attributes

- tokenizer
- model
- max_tokens
- tuple_delim

6.23.1 Constructor & Destructor Documentation

6.23.1.1 __init__()

```
__init__ (
              self,
              model_name = "Babelscape/rebel-large",
              max_tokens = 1024 )
```

6.23.2 Member Function Documentation

6.23.2.1 extract()

```
extract (
              self,
             str text,
             bool  parse_tuples = False )
```

6.23.3 Member Data Documentation

6.23.3.1 max_tokens

```
max_tokens
```

6.23.3.2 model

```
model
```

6.23.3.3 tokenizer

```
tokenizer
```

6.23.3.4 tuple_delim

```
tuple_delim
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py

6.24 ScalarMetric Class Reference

Properties

- string Name `[get, set]`
- double Value `[get, set]`

6.24.1 Property Documentation

6.24.1.1 Name

```
string Name [get], [set]
```

6.24.1.2 Value

```
double Value [get], [set]
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs

6.25 Session Class Reference

Stores active database connections and configuration settings.

Public Member Functions

- __new__ (cls, ∗args, ∗∗kwargs)

  Creates a new session at first access, otherwise uses the existing session.
- __init__ (self, verbose=False)

  Initializes the session using the .env file.
- test_database_connections (self)

  Configure the databases and verify they are working correctly.
- reset (self)

  Deletes all created databases and tables.

Public Attributes

- verbose

  Initializes the session using the .env file.
- relational_db

  Stores RDF-compliant semantic triples.
- docs_db

  Stores input text, pre-processed chunks, JSON intermediates, and final output.
- graph_db

  Main storage for entities (nodes) and relations (edges).

Static Protected Attributes

- _instance = None

  Creates a new session at first access, otherwise uses the existing session.

6.25.1 Detailed Description

Stores active database connections and configuration settings.

- This class implements Singleton design, so only one session can be created.
- However, the session config can still be updated using the normal constructor.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 __init__()

```
__init__ (
              self,
              verbose = False )
```

Initializes the session using the .env file.

- The relational database connector is created using a Factory Method, choosing mysql or postgres based on the .env file.
- The document database connector is created normally since mongo is the only supported option.
- The graph database connector is created normally since neo4j is the only supported option.

6.25.3 Member Function Documentation

6.25.3.1 __new__()

```
__new__ (
              cls,
             * args,
             ** kwargs )
```

Creates a new session at first access, otherwise uses the existing session.

6.25.3.2 reset()

```
reset (
              self )
```

Deletes all created databases and tables.

6.25.3.3 test_database_connections()

```
test_database_connections (
              self )
```

Configure the databases and verify they are working correctly.

6.25.4 Member Data Documentation

6.25.4.1 _instance

```
_instance = None  [static], [protected]
```

Creates a new session at first access, otherwise uses the existing session.

6.25.4.2 docs_db

```
docs_db
```

Stores input text, pre-processed chunks, JSON intermediates, and final output.

6.25.4.3 graph_db

```
graph_db
```

Main storage for entities (nodes) and relations (edges).

6.25.4.4 relational_db

```
relational_db
```

Stores RDF-compliant semantic triples.

6.25.4.5 verbose

```
verbose
```

Initializes the session using the .env file.

- The relational database connector is created using a Factory Method, choosing mysql or postgres based on the .env file.
  - The document database connector is created normally since mongo is the only supported option.
  - The graph database connector is created normally since neo4j is the only supported option.

Enables or disables the components from printing debug info.

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/setup.py

6.26 Story Class Reference

Public Member Functions

- None __init__ (self, StoryStreamAdapter reader)
- Iterator[Chunk] stream_chunks (self)
- None pre_split_chunks (self, int max_chunk_length)

  Splits paragraphs into chunks.

Public Attributes

- reader

Protected Member Functions

- None _merge_chunks (self, List[Chunk] segs, int max_len)
- Chunk _make_single (self, Chunk seg, str text, int max_len, Optional[Chunk] start=None)

6.26.1 Constructor & Destructor Documentation

6.26.1.1 __init__()

```
None __init__ (
             self,
            StoryStreamAdapter reader )
```

6.26.2 Member Function Documentation

6.26.2.1 _make_single()

```
Chunk _make_single (
              self,
             Chunk seg,
             str text,
             int max_len,
             Optional[Chunk]  start = None )  [protected]
```

6.26.2.2 _merge_chunks()

```
None _merge_chunks (
              self,
             List[Chunk] segs,
             int max_len )  [protected]
```

6.26.2.3 pre_split_chunks()

```
None pre_split_chunks (
              self,
             int max_chunk_length )
```

Splits paragraphs into chunks.

- Populates self.chunks with Chunk objects that obey max_chunk_length.
- Combines adjacent paragraphs when possible.
- Falls back to splitting by sentences if one paragraph is too long.

6.26.2.4 stream_chunks()

```
Iterator[Chunk] stream_chunks (
              self )
```

6.26.3 Member Data Documentation

6.26.3.1 reader

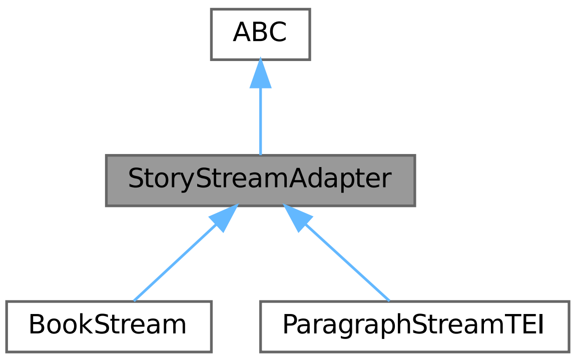
```
reader
```

The documentation for this class was generated from the following file:

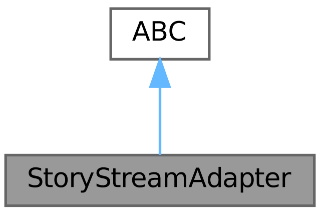
- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.27 StoryStreamAdapter Class Reference

Inheritance diagram for StoryStreamAdapter:



Collaboration diagram for StoryStreamAdapter:



Public Member Functions

- Iterator[Chunk] stream_segments (self)

    Yields sanitized parts of a book.
- Iterator[Chunk] stream_paragraphs (self)

    Concrete helper method to split segments into paragraphs.
- Iterator[str] stream_sentences (self)

    Concrete helper method to split paragraphs into sentences.

6.27.1 Member Function Documentation

6.27.1.1 stream_paragraphs()

```
Iterator[Chunk] stream_paragraphs (
              self )
```

Concrete helper method to split segments into paragraphs.

The Chunk class is repurposed here so we pass location info. Depending on the Story.pre_split_chunks implementation, this might be unnecessary.

6.27.1.2 stream_segments()

```
Iterator[Chunk] stream_segments (
              self )
```

Yields sanitized parts of a book.

- Story segments usually correspond to chapters.
- They serve as borders between chunking operations, ensuring chunks do not span multiple chapters. Implementation is handled by child classes BookStream, etc.
- Segments should be pre-cleaned and must contain 1 paragraph per line with all other newlines removed.

Reimplemented in ParagraphStreamTEI, and BookStream.

6.27.1.3 stream_sentences()

```
Iterator[str] stream_sentences (
              self )
```

Concrete helper method to split paragraphs into sentences.

Mostly for debugging.

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.28 SummaryData Class Reference

Properties

- string BookID `[get, set]`
- string BookTitle `[get, set]`
- string SummaryText `[get, set]`
- string GoldSummaryText `[get, set]`
- SummaryMetrics Metrics = new() `[get, set]`
- List< QAMetric > QAResults = new() `[get, set]`

6.28.1 Property Documentation

6.28.1.1 BookID

```
string BookID  [get], [set]
```

6.28.1.2 BookTitle

```
string BookTitle  [get], [set]
```

6.28.1.3 GoldSummaryText

```
string GoldSummaryText  [get], [set]
```

6.28.1.4 Metrics

```
SummaryMetrics Metrics = new()  [get], [set]
```

6.28.1.5 QAResults

```
List<QAMetric> QAResults = new()  [get], [set]
```

6.28.1.6 SummaryText

```
string SummaryText  [get], [set]
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryData.cs

6.29 SummaryMetrics Class Reference

Static Public Member Functions

- static SummaryMetrics GetDefault ()

Properties

- List< PRF1Metric > PRF1Metrics = new() `[get, set]`
- QAMetric QA = new() `[get, set]`
- List< ScalarMetric > ScalarMetrics = new() `[get, set]`

6.29.1 Member Function Documentation

6.29.1.1 GetDefault()

```
static SummaryMetrics GetDefault ( )  [static]
```

6.29.2 Property Documentation

6.29.2.1 PRF1Metrics

```
List<PRF1Metric> PRF1Metrics = new()  [get], [set]
```

6.29.2.2 QA

```
QAMetric QA = new()  [get], [set]
```

6.29.2.3 ScalarMetrics

```
List<ScalarMetric> ScalarMetrics = new()  [get], [set]
```

The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs

Chapter 7

File Documentation

7.1 /home/runner/work/dsci-capstone/dsci-capstone/components/book↩ _conversion.py File Reference

Classes

- class Chunk

    Lightweight container for a span of story text.
- class StoryStreamAdapter
- class Story
- class ParagraphStreamTEI

    Streams paragraphs from a TEI file as Chunk objects.
- class Book
- class BookStream
- class BookFactory
- class EPUBToTEI

    Converts EPUB files to XML format (TEI specification).

Namespaces

- namespace components
- namespace components.book_conversion

Variables

- nlp = spacy.blank("en")
- sentencizer = nlp.add_pipe("sentencizer")

7.2 /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py File Reference

Classes

- class Connector

  Abstract base class for external connectors.
- class DatabaseConnector

  Abstract base class for database engine connectors.
- class RelationalConnector

  Connector for relational databases (MySQL, PostgreSQL).
- class mysqlConnector

  A relational database connector configured for MySQL.
- class postgresConnector

  A relational database connector configured for PostgreSQL.

Namespaces

- namespace components
- namespace components.connectors

7.3 /home/runner/work/dsci-capstone/dsci-capstone/components/corpus.py File Reference

Namespaces

- namespace components
- namespace components.corpus

Functions

- load_booksum ()
- to_df_booksum (ds)
- load_narrativeqa ()
- to_df_nqa (ds)
- normalize_title (t)
- merge_dataframes (df1, df2, suffix1, suffix2, key_columns)
- fuzzy_merge_titles (df1, df2, suffix1, suffix2, key="title", threshold=90, scorer=fuzz.token_sort_ratio)

  Perform a two-way fuzzy merge between two DataFrames on a text column (e.g., book titles).

Variables

- df_booksum = load_booksum()
- df_nqa = load_narrativeqa()
- df = fuzzy_merge_titles(df_booksum, df_nqa, "_booksum", "_nqa", key="title", threshold=70)
- index
- m = Metrics()

7.4 /home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py File Reference

Classes

- class DocumentConnector

  Connector for MongoDB (document database)

Namespaces

- namespace components
- namespace components.document_storage

Functions

- MongoHandle mongo_handle (str host, str alias)

  Establish a temporary connection to MongoDB.
- DataFrame _flatten_recursive (DataFrame df)

  Explode all list columns and flatten dict columns until only scalars remain.
- str _sanitize_json (str text)

  Remove comments and other non-JSON content from a MongoDB query string.
- Dict[str, Any] _sanitize_document (Dict[str, Any] doc, Dict[str, Set[Type[Any]]] type_registry)

  Normalize document fields to consistent types for DataFrame construction.
- DataFrame _docs_to_df (List[Dict[str, Any]] docs, bool merge_unspecified=True)

  Convert raw MongoDB documents to a Pandas DataFrame.
- str _find_compatible_nested_key (Type[Any] value_type, Dict[str, Set[Type[Any]]] nested_schema, bool merge_unspecified)

  Find a nested column compatible with the given primitive type.

Variables

- MongoHandle = Generator["Database[Any]", None, None]

7.5 /home/runner/work/dsci-capstone/dsci-capstone/components/fact_storage.py File Reference

Classes

- class GraphConnector

  Connector for Neo4j (graph database).

Namespaces

- namespace components
- namespace components.fact_storage

Functions

- Tuple[Optional[List[Tuple[Any,...]]], Optional[List[str]]] filter_valid (List[Tuple[Any,...]] results, List[str] meta, str db_name)

  Filter Cypher query results (nodes or relationships) by database context.

7.6 /home/runner/work/dsci-capstone/dsci-capstone/components/metrics.py File Reference

Classes

- class Metrics

  Utility class for computing and posting evaluation metrics.

Namespaces

- namespace components
- namespace components.metrics

Functions

- Dict[str, Any] run_questeval (Dict[str, Any] chunk, ∗str qeval_task="summarization", bool use_cuda=False, bool use_question_weighter=True)

  Run QuestEval metric calculation.
- Dict[str, Any] run_bookscore (Dict[str, Any] chunk, ∗str model="gpt-3.5-turbo", int batch_size=10, bool use←_v2=True)

  Run BooookScore metric for long-form summarization.
- str chunk_bookscore (str book_text, str book_title='book', int chunk_size=2048)

  Chunk a book into BooookScore segments.

7.7 /home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py File Reference

7.8 /home/runner/work/dsci-capstone/dsci-capstone/components/text_←processing.py File Reference

Classes

- class RelationExtractor
- class LLMConnector

  Connector for prompting and returning LLM output (raw text/JSON) via LangChain.

Namespaces

- namespace components
- namespace components.text_processing

Variables

- nlp = spacy.blank("en")
- sentencizer = nlp.add_pipe("sentencizer")

7.9 /home/runner/work/dsci-capstone/dsci-capstone/components/__↩ init__.py File Reference

Namespaces

- namespace components

7.10 /home/runner/work/dsci-capstone/dsci-capstone/src/__init__.py File Reference

Namespaces

- namespace src

7.11 /home/runner/work/dsci-capstone/dsci-capstone/tests/__init__.py File Reference

Namespaces

- namespace tests

7.12 /home/runner/work/dsci-capstone/dsci-capstone/src/flask.py File Reference

Namespaces

- namespace src
- namespace src.flask

  Generic Flask worker microservice for distributed task processing.

Functions

- process_task (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] chunk_doc, str boss_url, Callable[[Dict[str, Any]], Dict[str, Any]] task_handler, Any task_kwargs=None)

    Perform the assigned task in a background thread.
- str load_mongo_config (str database)

    Load MongoDB configuration from environment variables.
- str load_boss_config ()

    Load boss service callback URL from environment variables.
- Tuple[Callable[[Dict[str, Any]], Dict[str, Any]], Dict[str, Any]] get_task_info (str task_name)

    Dynamically import and return the appropriate task handler function.
- load_imports (func)

    Pre-warm the task by importing requirements.
- None mark_task_in_progress (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name)

    Mark a task as in-progress in MongoDB before processing begins.
- None save_task_result (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] result)

    Save completed task results to MongoDB.
- None notify_boss (str boss_url, str chunk_id, str task_name, str status)

    Send completion notification to boss service.
- Flask create_app (str task_name, str boss_url)

    Create and configure Flask application for task processing.

Variables

- MongoHandle = Generator["Database[Any]", None, None]
- parser = argparse.ArgumentParser(description="Flask worker microservice")
- required
- True
- help
- args = parser.parse_args()
- str task_queue = Queue()
- target
- task_worker ()

    Background threading system for non-blocking task handling.
- daemon
- str boss_url = load_boss_config()
- PORT = int(os.environ[f"{args.task.upper()}_PORT"])
- Flask app = create_app(args.task, boss_url)
- host
- port
- use_reloader

7.13 /home/runner/work/dsci-capstone/dsci-capstone/src/main.py File Reference

Namespaces

- namespace src
- namespace src.main

Functions

- convert_single ()

  Converts one EPUB file to TEI format.
- convert_from_csv ()

  Converts several EPUB files to TEI format.
- chunk_single ()

  Creates a Story and many Chunks from a TEI file.
- test_relation_extraction ()

  Runs REBEL on a basic example; used for debugging.
- process_single ()

  Uses NLP and LLM to process an existing TEI file.
- graph_triple_files (session)

  Loads JSON into Neo4j to test the Blazor graph page.
- output_single (session)

  Generates a summary from triples stored in JSON, and posts data to Blazor.
- full_pipeline (session, collection_name, epub_path, book_chapters, start_str, end_str, book_id, story_id, book_title)
- old_main (session, collection_name)
- pipeline_1 (epub_path, book_chapters, start_str, end_str, book_id, story_id)

  Connects all components to convert an EPUB file to a book summary.
- pipeline_2 (session, collection_name, chunks, book_title)

  Extracts triples from a random chunk.
- pipeline_3 (session, triples)

  Generates a LLM summary using Neo4j triples.
- pipeline_4 (session, collection_name, triples_string, chunk_id)

  Generate chunk summary.
- pipeline_5a (summary, book_title, book_id)

  Send book info to Blazor.
- pipeline_5b (summary, book_title, book_id, chunk, gold_summary="", float bookscore=None, float questeval=None)

  Send metrics to Blazor.
- Dict[str, str] load_worker_config (List[str] task_types)

  Load worker service URLs from environment variables.
- None clear_task_data (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name)

  Clear any existing task data before assigning new task to worker.
- bool assign_task_to_worker (str worker_url, str database_name, str collection_name, str chunk_id)

  Assign a task to a worker microservice.
- Flask create_app (DocumentConnector docs_db, str database_name, str collection_name, Dict[str, str] worker_urls)

  Create and configure Flask application for boss service.
- requests.models.Response post_story_status (int boss_port, int story_id, str task, str status)

  Helpers to interact with the Flask boss thread.
- requests.models.Response post_chunk_status (int boss_port, str chunk_id, int story_id, str task, str status)

  Send a chunk-level update to the boss Flask app.
- requests.models.Response post_process_full_story (int boss_port, int story_id, str task_type)

  Process all chunks in MongoDB matching the provided story ID.

Variables

- str tei = "./datasets/examples/trilogy-wishes-1.tei"

  Will revisit later - Book classes need refactoring ###.
- str chapters
- str start = ""
- str end = "But I must say no more."
- list triple_files
- list response_files = ["./datasets/triples/chunk-160_story-1.txt"]
- MongoHandle = Generator["Database[Any]", None, None]
- session = Session(verbose=False)
- DB_NAME = os.environ["DB_NAME"]
- BOSS_PORT = int(os.environ["PYTHON_PORT"])
- COLLECTION = os.environ["COLLECTION_NAME"]
- mongo_db = session.docs_db.get_unmanaged_handle()
- collection = getattr(mongo_db, COLLECTION)
- list task_types = ["questeval", "bookscore"]
- Dict[str, str] worker_urls = load_worker_config(task_types)
- Flask app = create_app(session.docs_db, DB_NAME, COLLECTION, worker_urls)
- app run_app = lambda.run(host="0.0.0.0", port=BOSS_PORT, use_reloader=False)
- target
- daemon
- int story_id = 1
- int book_id = 2
- str book_title = "The Phoenix and the Carpet"
- chunks
- triples
- chunk
- chunk_id = chunk.get_chunk_id()
- triples_string = pipeline_3(session, triples)
- summary = pipeline_4(session, COLLECTION, triples_string, chunk.get_chunk_id())
- requests.models.Response response = post_process_full_story(BOSS_PORT, story_id, task_type)

7.14 /home/runner/work/dsci-capstone/dsci-capstone/src/setup.py File Reference

Classes

- class Session

  Stores active database connections and configuration settings.

Namespaces

- namespace src
- namespace src.setup

Variables

- session = Session()

7.15 /home/runner/work/dsci-capstone/dsci-capstone/src/util.py File Reference

Classes

- class Log

  The Log class standardizes console output.
- class Log.Failure

Namespaces

- namespace src
- namespace src.util

Functions

- all_none (∗args)

  Checks if all provided args are None.
- DataFrame df_natural_sorted (DataFrame df, List[str] ignored_columns=[ ], List[str] sort_columns=[ ])

  Sort a DataFrame in natural order using only certain columns.
- bool check_values (List[Any] results, List[Any] expected, bool verbose, str log_source, bool raise_error)

  Safely compare two lists of values.

7.16 /home/runner/work/dsci-capstone/dsci-capstone/tests/conftest.py File Reference

Namespaces

- namespace tests
- namespace tests.conftest

Functions

- pytest_addoption (parser)
- session (request)

  Fixture to create session.

7.17 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_↩ components.py File Reference

Namespaces

- namespace tests
- namespace tests.test_components

Functions

- RelationalConnector relational_db (Session session)

    Fixture to get relational database connection.
- DocumentConnector docs_db (Session session)

    Fixture to get document database connection.
- GraphConnector graph_db (Session session)

    Fixture to get document database connection.
- None test_db_relational_minimal (RelationalConnector relational_db)

    Tests if the RelationalConnector has a valid connection string.
- None test_db_docs_minimal (DocumentConnector docs_db)

    Tests if the DocumentConnector has a valid connection string.
- None test_db_graph_minimal (GraphConnector graph_db)

    Tests if the GraphConnector has a valid connection string.
- None test_db_relational_comprehensive (RelationalConnector relational_db)

    Tests if the GraphConnector is working as intended.
- None test_db_docs_comprehensive (DocumentConnector docs_db)

    Tests if the GraphConnector is working as intended.
- None test_db_graph_comprehensive (GraphConnector graph_db)

    Tests if the GraphConnector is working as intended.
- Generator[None, None, None] load_examples_relational (RelationalConnector relational_db)

    Fixture to create relational tables using engine-specific syntax.
- None test_sql_example_1 (RelationalConnector relational_db, Generator[None, None, None] load_examples_relational)

    Run queries contained within test files.
- None test_sql_example_2 (RelationalConnector relational_db, Generator[None, None, None] load_examples_relational)

    Run queries contained within test files.
- None test_mongo_example_1 (DocumentConnector docs_db)

    Run queries contained within test files.
- None test_mongo_example_2 (DocumentConnector docs_db)

    Run queries contained within test files.
- None test_mongo_example_3 (DocumentConnector docs_db)

    Run queries contained within test files.
- None test_cypher_example_1 (GraphConnector graph_db)

    Run queries contained within test files.
- None test_cypher_example_2 (GraphConnector graph_db)

    Test social network graph with relationships and mixed query patterns.
- None test_cypher_example_3 (GraphConnector graph_db)

    Test scene and dialogue graphs with proper isolation.
- None _test_query_file (DatabaseConnector db_fixture, str filename, List[str] valid_files)

    Run queries from a local file through the database.

7.18 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/_Imports.razor File Reference

7.19 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/App.razor File Reference

7.20 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Layout/MainLayout.razor File Reference

7.21 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Layout/NavMenu.razor File Reference

7.22 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Pages/Error.razor File Reference

7.23 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Pages/Graph.razor File Reference

7.24 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Pages/Home.razor File Reference

7.25 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Pages/Metrics.razor File Reference

7.26 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Components/Routes.razor File Reference

7.27 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↵ App/Controllers/MetricsController.cs File Reference

Classes

- class MetricsController

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Controllers

7.28 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Hubs/MetricsHub.cs File Reference

Classes

- class MetricsHub

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Hubs

7.29 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/PRF1Metric.cs File Reference

Classes

- class PRF1Metric

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

7.30 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/QAItem.cs File Reference

Classes

- class QAItem

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

7.31 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/QAMetric.cs File Reference

Classes

- class QAMetric

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

7.32 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/ScalarMetric.cs File Reference

Classes

- class ScalarMetric

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

7.33 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/SummaryData.cs File Reference

Classes

- class SummaryData

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

7.34 /home/runner/work/dsci-capstone/dsci-capstone/web-app/Blazor↩ App/Models/SummaryMetrics.cs File Reference

Classes

- class SummaryMetrics

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

Index