Data Science Capstone Project

Generated by Doxygen 1.9.8

Chapter 1

Components Module Overview

This module contains all high-level pipeline functionality:

- **Corpus** — text dataset ingestion (BookSum, NarrativeQA)
- **Relation Extractor** — NLP-based relation extraction
- **Metrics** — evaluation metrics (QuestEval, BooookScore, etc.)
- **Fact Storage** — Knowledge graph operations and semantic triple sanitization
- **Book Conversion** — Gutenberg preprocessing helpers (EPUB, HTML, TEI)

These components operate on data retrieved via the connectors and support the full text-processing and knowledge-graph pipeline. Each class remains independent, with dependencies injected through the global Session.

Chapter 2

Connectors Module Overview

This module provides lightweight adapters for each supported database:

- **relational.py** — MySQL/PostgreSQL connector (Factory Method via from_env)
- **document.py** — MongoDB connector
- **graph.py** — Neo4j connector
- **llm.py** — LLM connector via LangChain
- **base.py** — shared abstract base classes

All connectors expose a consistent operational interface but differ in their underlying database engines. They are instantiated once inside the global Session, ensuring stable configuration and preventing duplicate connections.

Chapter 3

Core Module Overview

This module contains the application's orchestration layer:

- the global **Session** (singleton)
- the **Boss** and **Worker** coordination scripts

3.1 Pattern

- **Separation of Concerns** — Session manages dependency wiring so connectors and components remain focused on their own tasks

3.2 Usage

```python from src.core.context import session

session.relational_db.execute_query("SELECT 1") session.main_graph.add_triple("Alice", "interactsWith", "Bob")
```

Chapter 4

Source Directory Overview

All application code lives inside this directory, organized into clear responsibility-based modules:

- **core**/ — orchestration logic (Session, Boss, Worker)
- **connectors**/ — adapters for relational, document, and graph databases
- **components**/ — pipeline logic (NLP, metrics, corpus processing, book parsing)
- **main.py** / **util.py** — entry points and shared logging / error-handling utilities

This layout separates low-level adapters, high-level pipeline operations, and global orchestrators, improving maintainability and readability.

Chapter 5

Database Example Datasets

This directory contains small, deterministic example files used by the database-related tests. Each subfolder corresponds to a specific connector type implemented in the project:

- **relational**/ – SQL queries or fixture datasets for MySQL/PostgreSQL.
- **document**/ – JSON or BSON-like structures for MongoDB tests.
- **graph**/ – Cypher queries for Neo4j graph tests.

These examples are intentionally minimal so tests:

- run quickly,
- avoid external dependencies,
- stay stable across environments,
- and clearly demonstrate the expected input format for each connector.

All files here are for testing only and do not represent production data.

Chapter 6

Test Suite Overview

This project uses pytest with `pytest-order` and `pytest-dependency` to ensure deterministic test sequencing.

Test groups:

- **test_db_basic** / **test_db_files** — relational, document, and graph connector tests.
- **test_kg_triples** — KnowledgeGraph parsing, ID to name mapping, and graph helpers.
- **examples-db**, **examples-llm** — minimal deterministic datasets used by the tests.

Run the suite with:

```
pytest -m "not smoke" tests/
```

Or (if Docker is set up):

```
make docker-all-dbs
make docker-test
```

Chapter 7

Namespace Index

7.1 Package List

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

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

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

Chapter 9

Class Index

9.1 Class List

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

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

Chapter 11

Namespace Documentation

11.1 BlazorApp Namespace Reference

Namespaces

- namespace Controllers
- namespace Hubs
- namespace Models

11.2 BlazorApp.Controllers Namespace Reference

Classes

- class MetricsController

11.3 BlazorApp.Hubs Namespace Reference

Classes

- class MetricsHub

11.4 BlazorApp.Models Namespace Reference

Classes

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

11.5 conftest Namespace Reference

Functions

- None pytest_addoption (Any parser)

  Command-line flags for pytest.
- pytest.param optional_param (str name, str package)

  Return a pytest.param that is skipped if the given package is missing.
- Generator[Session, None, None] session (pytest.FixtureRequest request)

  Fixture to create session.
- Generator[RelationalConnector, None, None] relational_db (Session session)

  Fixture to get relational database connection.
- Generator[DocumentConnector, None, None] docs_db (Session session)

  Fixture to get document database connection.
- Generator[GraphConnector, None, None] graph_db (Session session)

  Fixture to get document database connection.
- Generator[KnowledgeGraph, None, None] main_graph (pytest.FixtureRequest request, GraphConnector graph_db, Session session)

  Fixture to get document database connection.

11.5.1 Function Documentation

11.5.1.1 docs_db()

```
Generator[DocumentConnector, None, None] docs_db (
            Session session )
```

Fixture to get document database connection.

11.5.1.2 graph_db()

```
Generator[GraphConnector, None, None] graph_db (
            Session session )
```

Fixture to get document database connection.

11.5.1.3 main_graph()

```
Generator[KnowledgeGraph, None, None] main_graph (
            pytest.FixtureRequest request,
            GraphConnector graph_db,
            Session session )
```

Fixture to get document database connection.

11.5.1.4 optional_param()

```
pytest.param optional_param (
            str name,
            str package )
```

Return a pytest.param that is skipped if the given package is missing.

Parameters

name	The fixture name to include in the parameter list.
package	The name of a Python package to check for.

Returns

PyTest parameter with the skip flag set if package is not installed.

11.5.1.5 pytest_addoption()

```
None pytest_addoption (
             Any parser )
```

Command-line flags for pytest.

Usage: pytest –log-success –no-log-colors

11.5.1.6 relational_db()

```
Generator[RelationalConnector, None, None] relational_db (
             Session session )
```

Fixture to get relational database connection.

11.5.1.7 session()

```
Generator[Session, None, None] session (
             pytest.FixtureRequest request )
```

Fixture to create session.

11.6 src Namespace Reference

Namespaces

- namespace charts
- namespace components
- namespace connectors
- namespace core
- namespace main
- namespace util

11.7 src.charts Namespace Reference

Classes

- class Plot

  Static plotting helpers for visualization.

11.8 src.components Namespace Reference

Namespaces

- namespace book_conversion
- namespace corpus
- namespace fact_storage
- namespace metrics
- namespace relation_extraction

11.9 src.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")

11.9.1 Variable Documentation

11.9.1.1 nlp

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

11.9.1.2 sentencizer

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

11.10 src.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()

11.10.1 Function Documentation

11.10.1.1 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.

11.10.1.2 load_booksum()

```
load_booksum ( )
```

11.10.1.3 load_narrativeqa()

```
load_narrativeqa ( )
```

11.10.1.4 merge_dataframes()

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

11.10.1.5 normalize_title()

```
normalize_title (
              t )
```

11.10.1.6 to_df_booksum()

```
to_df_booksum (
              ds )
```

11.10.1.7 to_df_nqa()

```
to_df_nqa (
              ds )
```

11.10.2 Variable Documentation

11.10.2.1 df

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

11.10.2.2 df_booksum

```
df_booksum = load_booksum()
```

11.10.2.3 df_nqa

```
df_nqa = load_narrativeqa()
```

11.10.2.4 index

```
index
```

11.10.2.5 m

```
m = Metrics()
```

11.11 src.components.fact_storage Namespace Reference

Classes

- class KnowledgeGraph

  Manages a single graph within Neo4j.
- class Triple

Functions

- str sanitize_node (str label)

  Clean node name for Cypher safety.
- str sanitize_relation (str label, str mode="UPPER_CASE", str default_relation="RELATED_TO")

  Clean and normalize relation label for knowledge graphs.

Variables

- _nlp = None

11.11.1 Function Documentation

11.11.1.1 sanitize_node()

```
str sanitize_node (
             str label )
```

Clean node name for Cypher safety.

- Joins lists/tuples into single string
- Replaces invalid characters with underscores
- Trims leading/trailing underscores and spaces Used by KG systems before inserting nodes.

Parameters

label	Raw node name (subject / object)

Returns

Sanitized string suitable for node property

Exceptions

ValueError	If result is empty after sanitization

11.11.1.2 sanitize_relation()

```
str sanitize_relation (
            str label,
            str  mode = "UPPER_CASE",
            str  default_relation = "RELATED_TO" )
```

Clean and normalize relation label for knowledge graphs.

Supports two output modes:

- UPPER_CASE: Neo4j convention (e.g., RELATED_TO)
- camelCase: OWL/RDF convention (e.g., relatedTo)

Process:

- Replaces invalid characters with underscores
- Applies mode-specific casing rules
- Falls back to normalized default if empty or invalid start

Relations must start with alphabetic character. Default relation is automatically normalized to match mode.

Parameters

label	Raw relation label (string)
mode	Output format: "UPPER_CASE" or "camelCase"
default_relation	Fallback relation name (auto-normalized to mode)

Returns

Sanitized relation label in specified mode

Exceptions

ValueError	If mode is invalid

11.11.2 Variable Documentation

11.11.2.1 _nlp

```
_nlp = None  [protected]
```

11.12 src.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.

11.12.1 Function Documentation

11.12.1.1 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.

11.12.1.2 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.

11.12.1.3 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/QuestEval/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.

11.13 src.components.relation_extraction Namespace Reference

Classes

- class RelationExtractor

  Abstract base class for Relation Extraction (RE) models.
- class RelationExtractorOpenIE

  Wrapper for Stanford OpenIE using the Stanza library.
- class RelationExtractorREBEL

  Relation Extractor using the REBEL generative model (Seq2Seq).
- class RelationExtractorTextacy

  Lightweight extraction using Spacy and Textacy (SVO).

11.14 src.connectors Namespace Reference

Namespaces

- namespace base
- namespace document
- namespace graph
- namespace llm
- namespace relational

11.15 src.connectors.base Namespace Reference

Classes

- class Connector

  Abstract base class for external connectors.
- class DatabaseConnector

  Abstract base class for database engine connectors.

11.16 src.connectors.document 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]

11.16.1 Function Documentation

11.16.1.1 _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.

11.16.1.2 _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.

11.16.1.3 _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.

11.16.1.4 _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.

11.16.1.5 _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.

11.16.1.6 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_operations wants to call execute_query, but continue using its existing db handle after execute_query disconnects.

11.16.2 Variable Documentation

11.16.2.1 MongoHandle

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

11.17 src.connectors.graph Namespace Reference

Classes

- class GraphConnector

  Connector for Neo4j (graph database).

Functions

- DataFrame _filter_to_db (DataFrame df, str database_name)

  Filter a DataFrame by database context.
- DataFrame _tuples_to_df (List[Tuple[Any,...]] tuples, List[str] meta)

  Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.
- DataFrame _normalize_elements (DataFrame df)

  Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.

11.17.1 Function Documentation

11.17.1.1 _filter_to_db()

```
DataFrame _filter_to_db (
            DataFrame df,
            str database_name )  [protected]
```

Filter a DataFrame by database context.

- Keeps nodes where 'db' matches the current database name and _init is False/absent.
- Keeps relationships if either endpoint node (by elementId) matches the same db.
- Works on unflattened frames where cells are dicts (node/rel maps).
- Safely ignores missing fields; drops a top-level '_init' column if present.

Parameters

df	DataFrame containing node and relationship rows.
database_name	The name of the current pseudo-database.

Returns

Filtered DataFrame restricted to the active database.

11.17.1.2 _normalize_elements()

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

Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.

- Accepts the DataFrame output of src.connectors.graph.GraphConnector.execute_query.
- Explodes dict-cast elements from columns into rows, resulting in 1 node or relation per row.
- Normalizes node and relation properties as columns. `element_id`, `element_type` are shared.
- Node-only properties (e.g. labels) are None for relationships, and likewise for relations (e.g. start_node).
- Returns an empty DataFrame for no results.

Parameters

df	DataFrame containing dict-cast nodes and relationships.

Returns

DataFrame suitable for downstream filtering and analysis.

11.17.1.3 _tuples_to_df()

```
DataFrame _tuples_to_df (
            List[Tuple[Any, ...]] tuples,
            List[str] meta )  [protected]
```

Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.

- Accepts the `tuples` output of db.cypher_query().
- Automatically unwraps NeoModel Node/Relationship objects into plain dicts via `.__properties__`.
- Adds an 'element_type' property distinguishing nodes vs relationships. Example Query: MATCH (a)-[r]->(b) RETURN a AS node_1, r AS edge, b AS node_2; Result: DataFrame with `node_1` and `node_2` columns containing Nodes converted to dicts, and an `edge` column containing a dict-cast relationship (element_type: relation).

Parameters

tuples	List of Neo4j query result tuples.
meta	List of element aliases returned by the query, and used here as column names.

Returns

DataFrame with requested columns.

11.18 src.connectors.llm Namespace Reference

Classes

- class LLMConnector

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

Functions

- List[Dict[str, Any]] normalize_to_dict (Dict[str, Any]|List[Dict[str, Any]] data, List[str] keys)

  Normalize nested/compacted LLM output into flat dicts.

11.18.1 Function Documentation

11.18.1.1 normalize_to_dict()

```
List[Dict[str, Any]] normalize_to_dict (
            Dict[str, Any] | List[Dict[str, Any]] data,
            List[str] keys )
```

Normalize nested/compacted LLM output into flat dicts.

Handles token-saving patterns:

- Nested relation-object pairs: {"s":"X", [{"r":"R1","o":"O1"}, ...]}
- List subjects with nested r-o: {"s":["X","Y"], [{"r":"R","o":"O"}, ...]}
- Cartesian products: {"s":["X","Y"], "r":["R1","R2"], "o":["O1","O2"]} Assumes input is already parsed (json.loads called by caller).

Parameters

data	Parsed LLM output (dict or list of dicts)
keys	Expected keys (e.g., ["s", "r", "o"])

Returns

List of flat dicts with all keys present

Exceptions

ValueError	If input format cannot be parsed

11.19 src.connectors.relational Namespace Reference

Classes

- 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).

11.20 src.core Namespace Reference

Namespaces

- namespace boss

Boss microservice for orchestrating distributed task processing.
- namespace context
- namespace stages
- namespace worker

  Generic Flask worker microservice for distributed task processing.

11.21 src.core.boss Namespace Reference

Boss microservice for orchestrating distributed task processing.

Functions

- 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.
- None create_boss_thread (str DB_NAME, int BOSS_PORT, str COLLECTION)
- 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

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

11.21.1 Detailed Description

Boss microservice for orchestrating distributed task processing.

Manages task distribution to workers and tracks completion order.

11.21.2 Function Documentation

11.21.2.1 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.

11.21.2.2 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.

11.21.2.3 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.

11.21.2.4 create_boss_thread()

```
None create_boss_thread (
             str DB_NAME,
             int BOSS_PORT,
             str COLLECTION )
```

11.21.2.5 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.

11.21.2.6 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.

11.21.2.7 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.

11.21.2.8 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.

11.21.3 Variable Documentation

11.21.3.1 MongoHandle

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

11.22 src.core.context Namespace Reference

Classes

- class Session

  Stores active database connections and configuration settings.

Functions

- Session get_session (∗Any args, ∗∗Any kwargs)

  Lazily creates a session on first call, otherwise returns the existing session.
- Any __getattr__ (str name)

  Lazy attribute resolution for module-level imports.

Variables

- Session session

  The global instance of the singleton Session class.

11.22.1 Function Documentation

11.22.1.1 __getattr__()

```
Any __getattr__ (
              str name )
```

Lazy attribute resolution for module-level imports.

- Only called when normal attribute lookup fails (i.e., name not in module globals).
- Enables lazy session creation: `from src.core.context import session`
- Regular imports (Session, get_session, etc.) bypass this entirely.

Parameters

name	The attribute name being accessed.

Returns

The session singleton if 'session' is requested.

Exceptions

AttributeError	If an unknown/undefined attribute is requested.

11.22.1.2 get_session()

```
Session get_session (
             *Any args,
             **Any kwargs )
```

Lazily creates a session on first call, otherwise returns the existing session.

Note

Will ignore any arguments passed after creation.

Parameters

∗*args*	Positional arguments forwarded to Session().
∗∗*kwargs*	Keyword arguments forwarded to Session().

Returns

The global instance of the Session class.

11.22.2 Variable Documentation

11.22.2.1 session

```
Session session
```

The global instance of the singleton Session class.

11.23 src.core.stages Namespace Reference

Functions

- str task_01_convert_epub (str epub_path, Optional[EPUBToTEI] converter=None)

  Will revisit later - Book classes need refactoring ###.
- task_02_parse_chapters (tei_path, book_chapters, book_id, story_id, start_str, end_str)
- task_03_chunk_story (story, max_chunk_length=1500)
- task_10_random_chunk (chunks)
- task_10_sample_chunks (chunks, n_sample)
- task_11_send_chunk (c, collection_name, book_title)
- task_12_relation_extraction_rebel (text, max_tokens=1024, parse_tuples=True)
- task_12_relation_extraction_openie (text, memory='4G', parse_tuples=True)
- task_12_relation_extraction_textacy (text, parse_tuples=True)
- task_13_concatenate_triples (extracted)
- task_14_relation_extraction_llm (triples_string, text)
- str task_15_sanitize_triples_llm (str llm_output)
- task_20_send_triples (triples)
- group_21_1_describe_graph (top_n=3)
- group_21_2_send_statistics ()

- group_21_3_post_statistics ()
- task_22_verbalize_triples (mode="triple")
- task_30_summarize_llm (triples_string)

  Prompt LLM to generate summary.
- task_31_send_summary (summary, collection_name, chunk_id)
- task_40_post_summary (book_id, book_title, summary)

  Send book info to Blazor.
- task_40_post_payload (book_id, book_title, summary, gold_summary, chunk, bookscore, questeval)

  Send metrics to Blazor.

11.23.1 Function Documentation

11.23.1.1 group_21_1_describe_graph()

```
group_21_1_describe_graph (
              top_n = 3 )
```

11.23.1.2 group_21_2_send_statistics()

```
group_21_2_send_statistics ( )
```

11.23.1.3 group_21_3_post_statistics()

```
group_21_3_post_statistics ( )
```

11.23.1.4 task_01_convert_epub()

```
str task_01_convert_epub (
             str epub_path,
             Optional[EPUBToTEI]  converter = None )
```

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

11.23.1.5 task_02_parse_chapters()

```
task_02_parse_chapters (
              tei_path,
              book_chapters,
              book_id,
              story_id,
              start_str,
              end_str )
```

11.23.1.6 task_03_chunk_story()

```
task_03_chunk_story (
              story,
              max_chunk_length = 1500 )
```

11.23.1.7 task_10_random_chunk()

```
task_10_random_chunk (
              chunks )
```

11.23.1.8 task_10_sample_chunks()

```
task_10_sample_chunks (
              chunks,
              n_sample )
```

11.23.1.9 task_11_send_chunk()

```
task_11_send_chunk (
              c,
              collection_name,
              book_title )
```

11.23.1.10 task_12_relation_extraction_openie()

```
task_12_relation_extraction_openie (
              text,
              memory = '4G',
              parse_tuples = True )
```

11.23.1.11 task_12_relation_extraction_rebel()

```
task_12_relation_extraction_rebel (
              text,
              max_tokens = 1024,
              parse_tuples = True )
```

11.23.1.12 task_12_relation_extraction_textacy()

```
task_12_relation_extraction_textacy (
              text,
              parse_tuples = True )
```

11.23.1.13 task_13_concatenate_triples()

```
task_13_concatenate_triples (
              extracted )
```

11.23.1.14 task_14_relation_extraction_llm()

```
task_14_relation_extraction_llm (
              triples_string,
              text )
```

11.23.1.15 task_15_sanitize_triples_llm()

```
str task_15_sanitize_triples_llm (
             str llm_output )
```

11.23.1.16 task_20_send_triples()

```
task_20_send_triples (
              triples )
```

11.23.1.17 task_22_verbalize_triples()

```
task_22_verbalize_triples (
              mode = "triple" )
```

11.23.1.18 task_30_summarize_llm()

```
task_30_summarize_llm (
              triples_string )
```

Prompt LLM to generate summary.

11.23.1.19 task_31_send_summary()

```
task_31_send_summary (
              summary,
              collection_name,
              chunk_id )
```

11.23.1.20 task_40_post_payload()

```
task_40_post_payload (
              book_id,
              book_title,
              summary,
              gold_summary,
              chunk,
              bookscore,
              questeval )
```

Send metrics to Blazor.

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

11.23.1.21 task_40_post_summary()

```
task_40_post_summary (
                book_id,
                book_title,
                summary )
```

Send book info to Blazor.

- Post to Blazor metrics page

11.24 src.core.worker Namespace Reference

Generic Flask worker microservice for distributed task processing.

Functions

- None 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

11.24.1 Detailed Description

Generic Flask worker microservice for distributed task processing.

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

11.24.2 Function Documentation

11.24.2.1 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.

11.24.2.2 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.

11.24.2.3 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.

11.24.2.4 load_imports()

```
load_imports (
              func )
```

Pre-warm the task by importing requirements.

Parameters

func	The function to perform a dummy call on.

11.24.2.5 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.

11.24.2.6 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).

11.24.2.7 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').

11.24.2.8 process_task()

```
None 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.
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.

11.24.2.9 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.

11.24.3 Variable Documentation

11.24.3.1 app

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

11.24.3.2 args

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

11.24.3.3 boss_url

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

11.24.3.4 daemon

```
daemon
```

11.24.3.5 help

```
help
```

11.24.3.6 host

```
host
```

11.24.3.7 MongoHandle

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

11.24.3.8 parser

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

11.24.3.9 PORT

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

11.24.3.10 port

```
port
```

11.24.3.11 required

```
required
```

11.24.3.12 target

```
target
```

11.24.3.13 task_queue

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

11.24.3.14 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.

11.24.3.15 True

```
True
```

11.24.3.16 use_reloader

```
use_reloader
```

11.25 src.main Namespace Reference

Functions

- pipeline_A (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_B (collection_name, chunks, book_title)

  Extracts triples from a random chunk.
- pipeline_C (json_triples)

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

  Generate chunk summary.
- None pipeline_E (str summary, str book_title, str book_id, str chunk="", str gold_summary="", float bookscore=None, float questeval=None)

  Compute metrics and send available data to Blazor.
- full_pipeline (collection_name, epub_path, book_chapters, start_str, end_str, book_id, story_id, book_title)
- old_main (collection_name)

Variables

- DB_NAME = os.environ["DB_NAME"]
- BOSS_PORT = int(os.environ["PYTHON_PORT"])
- COLLECTION = os.environ["COLLECTION_NAME"]
- bool load_from_checkpoint = False
- str checkpoint_path = "./datasets/checkpoint.pkl"
- exist_ok
- int story_id = 1
- int book_id = 2
- str book_title = "The Phoenix and the Carpet"
- data = pickle.load(f_read)
- triples = data["triples"]
- chunk = data["chunk"]
- chunks
- chunk_id = chunk.get_chunk_id()
- triples_string = pipeline_C(triples)
- summary = pipeline_D(COLLECTION, triples_string, chunk.get_chunk_id())
- response = post_process_full_story(BOSS_PORT, story_id, task_type)

11.25.1 Function Documentation

11.25.1.1 full_pipeline()

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

11.25.1.2 old_main()

```
old_main (
              collection_name )
```

11.25.1.3 pipeline_A()

```
pipeline_A (
              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)

11.25.1.4 pipeline_B()

```
pipeline_B (
              collection_name,
              chunks,
              book_title )
```

Extracts triples from a random chunk.

- JSON triples (NLP & LLM)

11.25.1.5 pipeline_C()

```
pipeline_C (
              json_triples )
```

Generates a LLM summary using Neo4j triples.

- Neo4j graph database
- Blazor graph page

11.25.1.6 pipeline_D()

```
pipeline_D (
              collection_name,
              triples_string,
              chunk_id )
```

Generate chunk summary.

11.25.1.7 pipeline_E()

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

Compute metrics and send available data to Blazor.

11.25.2 Variable Documentation

11.25.2.1 book_id

```
int book_id = 2
```

11.25.2.2 book_title

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

11.25.2.3 BOSS_PORT

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

11.25.2.4 checkpoint_path

```
str checkpoint_path = "./datasets/checkpoint.pkl"
```

11.25.2.5 chunk

```
chunk = data["chunk"]
```

11.25.2.6 chunk_id

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

11.25.2.7 chunks

```
chunks
```

Initial value:

```
=  pipeline_A(
            epub_path="./tests/examples-pipeline/epub/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,
        )
```

11.25.2.8 COLLECTION

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

11.25.2.9 data

```
data = pickle.load(f_read)
```

11.25.2.10 DB_NAME

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

11.25.2.11 exist_ok

```
exist_ok
```

11.25.2.12 load_from_checkpoint

```
bool load_from_checkpoint = False
```

11.25.2.13 response

```
response = post_process_full_story(BOSS_PORT, story_id, task_type)
```

11.25.2.14 story_id

```
int story_id = 1
```

11.25.2.15 summary

```
summary = pipeline_D(COLLECTION, triples_string, chunk.get_chunk_id())
```

11.25.2.16 triples

```
triples = data["triples"]
```

11.25.2.17 triples_string

```
triples_string = pipeline_C(triples)
```

11.26 src.util Namespace Reference

Classes

- class Log

  The Log class standardizes console output.

Functions

- 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.

11.26.1 Function Documentation

11.26.1.1 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 src.connectors.relational.RelationalConnector.test_operations

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.

11.26.1.2 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.

11.27 tests Namespace Reference

Namespaces

- namespace test_db_basic
- namespace test_db_files
- namespace test_kg_triples
- namespace test_pipeline

11.28 tests.test_db_basic Namespace Reference

Functions

- 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.

11.28.1 Function Documentation

11.28.1.1 test_db_docs_comprehensive()

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

Tests if the GraphConnector is working as intended.

11.28.1.2 test_db_docs_minimal()

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

Tests if the DocumentConnector has a valid connection string.

11.28.1.3 test_db_graph_comprehensive()

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

Tests if the GraphConnector is working as intended.

11.28.1.4 test_db_graph_minimal()

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

Tests if the GraphConnector has a valid connection string.

11.28.1.5 test_db_relational_comprehensive()

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

Tests if the GraphConnector is working as intended.

11.28.1.6 test_db_relational_minimal()

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

Tests if the RelationalConnector has a valid connection string.

11.29 tests.test_db_files Namespace Reference

Functions

- 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_cypher_example_4 (GraphConnector graph_db)

  Test event graph with property mutations and multi-hop traversal.
- None _exec_query_file (DatabaseConnector db_fixture, str filename, List[str] valid_files)

  Run queries from a local file through the database.

11.29.1 Function Documentation

11.29.1.1 _exec_query_file()

```
None _exec_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.

11.29.1.2 load_examples_relational()

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

Fixture to create relational tables using engine-specific syntax.

11.29.1.3 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.

11.29.1.4 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.

11.29.1.5 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.

11.29.1.6 test_cypher_example_4()

```
None test_cypher_example_4 (
            GraphConnector graph_db )
```

Test event graph with property mutations and multi-hop traversal.

Validates MERGE property updates (2-wave assignment), relationship chains in DAG structure, consistent rel_type with varied properties, and multi-hop path queries. Tests that properties added via SET after initial CREATE are properly stored.

11.29.1.7 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.

11.29.1.8 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.

11.29.1.9 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.

11.29.1.10 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.

11.29.1.11 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.

11.30 tests.test_kg_triples Namespace Reference

Functions

- None test_knowledge_graph_triples (KnowledgeGraph main_graph)

  Test KnowledgeGraph triple operations using add_triple and get_all_triples.
- Generator[KnowledgeGraph, None, None] nature_scene_graph (KnowledgeGraph main_graph)

  Create a scene graph with multiple location-based communities for testing.
- None test_get_subgraph_by_nodes (KnowledgeGraph nature_scene_graph)

  Test filtering triples by specific node IDs.
- None test_get_neighborhood (KnowledgeGraph nature_scene_graph)

  Test k-hop neighborhood expansion around a central node.
- None test_get_random_walk_sample (KnowledgeGraph nature_scene_graph)

  Test random walk sampling starting from specified nodes.
- None test_get_neighborhood_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for k-hop neighborhood expansion.
- None test_get_random_walk_sample_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for random walk sampling.
- None test_detect_community_clusters_minimal (KnowledgeGraph nature_scene_graph)

  Test basic community detection functionality.
- None test_detect_community_clusters_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for community detection with various parameters.
- None test_ranked_degree (KnowledgeGraph nature_scene_graph)

  Test filtering triples by ranked node degree.
- None test_ranked_degree_ties (KnowledgeGraph main_graph)

  Test that degree ranking correctly handles ties with minimal data.
- node_nlp_cases ()

  Fixtures focusing on spaCy NLP cleaning (Stopword/Part-of-speech removal).
- node_regex_cases ()

  Fixtures focusing on Regex replacement and stripping.
- relation_casing_cases ()

  Fixtures for testing UPPER_CASE vs camelCase modes.
- relation_fallback_cases ()

  Fixtures specifically testing the fallback logic (when input is invalid/numeric).
- test_sanitize_node_nlp_capabilities (node_nlp_cases)

  Test that NLP logic correctly strips POS tags (DET, PRON, PART).
- test_sanitize_node_regex_cleaning (node_regex_cases)

  Test that Regex logic handles symbols and whitespace correctly.
- test_sanitize_relation_modes (relation_casing_cases, mode)

  Test standard relation normalization for both supported modes.
- test_sanitize_relation_fallbacks (relation_fallback_cases)

  Test the 'safety net' fallback logic for relations.
- test_sanitize_relation_default_normalization ()

  Edge case: Ensure the default_relation itself is sanitized if used.

11.30.1 Function Documentation

11.30.1.1 nature_scene_graph()

```
Generator[KnowledgeGraph, None, None] nature_scene_graph (
            KnowledgeGraph main_graph )
```

Create a scene graph with multiple location-based communities for testing.

Graph structure represents a park with distinct areas:

- Playground: swings, slide, kids
- Bench area: bench, parents
- Forest: trees, rock, path
- School building: doors, windows, classroom Each area forms a natural community for GraphRAG testing.

11.30.1.2 node_nlp_cases()

```
node_nlp_cases ( )
```

Fixtures focusing on spaCy NLP cleaning (Stopword/Part-of-speech removal).

11.30.1.3 node_regex_cases()

```
node_regex_cases ( )
```

Fixtures focusing on Regex replacement and stripping.

11.30.1.4 relation_casing_cases()

```
relation_casing_cases ( )
```

Fixtures for testing UPPER_CASE vs camelCase modes.

11.30.1.5 relation_fallback_cases()

```
relation_fallback_cases ( )
```

Fixtures specifically testing the fallback logic (when input is invalid/numeric).

11.30.1.6 test_detect_community_clusters_comprehensive()

```
None test_detect_community_clusters_comprehensive (
            KnowledgeGraph nature_scene_graph )
```

Comprehensive test for community detection with various parameters.

Tests:

- Multi-level hierarchical detection
- community_list structure for hierarchical summarization
- Invalid method handling
- Community stability and coverage

11.30.1.7 test_detect_community_clusters_minimal()

```
None test_detect_community_clusters_minimal (
            KnowledgeGraph nature_scene_graph )
```

Test basic community detection functionality.

Validates that detect_community_clusters assigns community_id properties to nodes and that get_community_↩ subgraph retrieves triples within a community. Tests both Leiden and Louvain methods.

11.30.1.8 test_get_neighborhood()

```
None test_get_neighborhood (
            KnowledgeGraph nature_scene_graph )
```

Test k-hop neighborhood expansion around a central node.

Validates that get_neighborhood correctly finds all triples within k hops of a starting node.

11.30.1.9 test_get_neighborhood_comprehensive()

```
None test_get_neighborhood_comprehensive (
            KnowledgeGraph nature_scene_graph )
```

Comprehensive test for k-hop neighborhood expansion.

Tests edge cases and advanced features:

- depth=0 (no expansion)
- Disconnected nodes
- Maximum depth reaching entire connected component
- Cycle handling (no infinite loops)
- Consistent results across multiple calls

11.30.1.10 test_get_random_walk_sample()

```
None test_get_random_walk_sample (
            KnowledgeGraph nature_scene_graph )
```

Test random walk sampling starting from specified nodes.

Validates that get_random_walk_sample generates a representative subgraph by following random paths through the graph.

11.30.1.11 test_get_random_walk_sample_comprehensive()

```
None test_get_random_walk_sample_comprehensive (
            KnowledgeGraph nature_scene_graph )
```

Comprehensive test for random walk sampling.

Tests edge cases and advanced features:

- Empty start_nodes list (should sample from any node)
- Dead-end nodes (leaf nodes with no outgoing edges)
- Walk length limits are respected
- Deterministic subset property
- Stochasticity verification

11.30.1.12 test_get_subgraph_by_nodes()

```
None test_get_subgraph_by_nodes (
            KnowledgeGraph nature_scene_graph )
```

Test filtering triples by specific node IDs.

Validates that get_subgraph_by_nodes correctly filters triples where either subject or object matches the provided node list.

11.30.1.13 test_knowledge_graph_triples()

```
None test_knowledge_graph_triples (
            KnowledgeGraph main_graph )
```

Test KnowledgeGraph triple operations using add_triple and get_all_triples.

Validates the KnowledgeGraph wrapper for semantic triple management:

- add_triple() creates nodes and relationships
- get_all_triples() retrieves triples as element IDs
- get_triple_properties() constructs a DataFrame with element properties as columns
- triples_to_names() maps IDs to human-readable names

11.30.1.14 test_ranked_degree()

```
None test_ranked_degree (
            KnowledgeGraph nature_scene_graph )
```

Test filtering triples by ranked node degree.

Validates that get_by_ranked_degree correctly returns triples whose endpoints belong to nodes within the specified degree rank range.

11.30.1.15 test_ranked_degree_ties()

```
None test_ranked_degree_ties (
            KnowledgeGraph main_graph )
```

Test that degree ranking correctly handles ties with minimal data.

Verifies that nodes with equal degrees receive the same rank and querying for non-existent ranks returns empty DataFrame.

11.30.1.16 test_sanitize_node_nlp_capabilities()

```
test_sanitize_node_nlp_capabilities (
              node_nlp_cases )
```

Test that NLP logic correctly strips POS tags (DET, PRON, PART).

Validates that 'sanitize_node' loads the global _nlp object and correctly filters linguistic tokens.

11.30.1.17 test_sanitize_node_regex_cleaning()

```
test_sanitize_node_regex_cleaning (
              node_regex_cases )
```

Test that Regex logic handles symbols and whitespace correctly.

Ensures strict node naming conventions:

- No non-alphanumeric chars (except underscore/space)
- No leading/trailing garbage

11.30.1.18 test_sanitize_relation_default_normalization()

```
test_sanitize_relation_default_normalization ( )
```

Edge case: Ensure the default_relation itself is sanitized if used.

If the input is empty, we return the default. But if the default provided is 'bad input', the function must clean the default before returning it.

11.30.1.19 test_sanitize_relation_fallbacks()

```
test_sanitize_relation_fallbacks (
              relation_fallback_cases )
```

Test the 'safety net' fallback logic for relations.

The function requires relations to start with an alphabetic character. If the input is garbage (e.g., '123' or '>>'), it must revert to the default_relation, and that default relation *must* also be normalized to the requested mode.

11.30.1.20 test_sanitize_relation_modes()

```
test_sanitize_relation_modes (
              relation_casing_cases,
              mode )
```

Test standard relation normalization for both supported modes.

Filters the fixture data to match the parameterized mode and verifies string transformation logic.

11.31 tests.test_pipeline Namespace Reference

Functions

- book_data (request)

  Fixtures.
- book_1_data ()

  Example data for Book 1: Five Children and It.
- book_2_data ()

  Example data for Book 2: The Phoenix and the Carpet - realistic pipeline data.
- llm_data (request)

  Realistic or malformed LLM response edge cases.
- llm_edge_case_1 ()

  Save tokens by reusing the original subject.
- llm_edge_case_2 ()

  Save tokens by providing multiple subjects.
- llm_edge_case_3 ()

  Combine subject: List[str] and relation-object: List[Dict[str, str]].
- llm_edge_case_4 ()

  Same-length lists are also parsable.
- llm_edge_case_5 ()

  Mismatched-length lists: inferred as Cartesian Product.
- llm_edge_case_6 ()

  Matched-length lists: inferred as Columnar (Zip).
- test_job_01_convert_epub (book_data)

  Tests.
- test_job_02_parse_chapters (book_data)

  Test TEI -> Story parsing for multiple books.
- test_job_03_chunk_story (book_data)

  Test Story -> chunks splitting for multiple books.

- test_job_10_sample_chunks (book_data)

  Test sampling multiple chunks from a list.
- test_job_10_random_chunk (book_data)

  Test selecting a single random chunk.
- test_job_11_send_chunk (docs_db, book_data)

  Test inserting chunk into MongoDB collection.
- test_job_13_concatenate_triples (book_data)

  Test converting extracted triples to newline-delimited string.
- test_job_15_sanitize_triples_llm (book_data)

  Test parsing LLM output JSON into triples list.
- test_job_15_comprehensive (llm_data)

  Test parsing malformed LLM output.
- test_job_20_send_triples (main_graph, book_data)

  Test inserting triples into knowledge graph.
- test_job_21_describe_graph (main_graph, book_data)

  Test generating edge count summary of knowledge graph.
- test_job_22_verbalize_triples (main_graph, book_data)

  Test converting high-degree triples to string format.
- test_job_31_send_summary (docs_db, book_data)

  Test updating chunk with summary in MongoDB.
- test_pipeline_A_minimal (book_data)

  Minimal aggregate test.
- test_pipeline_A_from_csv ()

  Read example CSV and run pipeline_A for each row.
- test_pipeline_C_minimal (main_graph, book_data)

  Test running pipeline_C with smoke test data.
- test_pipeline_E_minimal_summary_only (book_data)

  Test running pipeline_E with summary-only mode.
- test_pipeline_E_minimal_full_payload (book_data)

  Test running pipeline_E with full payload including metrics.

11.31.1 Function Documentation

11.31.1.1 book_1_data()

```
book_1_data ( )
```

Example data for Book 1: Five Children and It.

11.31.1.2 book_2_data()

```
book_2_data ( )
```

Example data for Book 2: The Phoenix and the Carpet - realistic pipeline data.

11.31.1.3 book_data()

```
book_data (
              request )
```

Fixtures.

11.31.1.4 llm_data()

```
llm_data (
             request )
```

Realistic or malformed LLM response edge cases.

11.31.1.5 llm_edge_case_1()

```
llm_edge_case_1 ( )
```

Save tokens by reusing the original subject.

11.31.1.6 llm_edge_case_2()

```
llm_edge_case_2 ( )
```

Save tokens by providing multiple subjects.

11.31.1.7 llm_edge_case_3()

```
llm_edge_case_3 ( )
```

Combine subject: List[str] and relation-object: List[Dict[str, str]].

11.31.1.8 llm_edge_case_4()

```
llm_edge_case_4 ( )
```

Same-length lists are also parsable.

11.31.1.9 llm_edge_case_5()

```
llm_edge_case_5 ( )
```

Mismatched-length lists: inferred as Cartesian Product.

11.31.1.10 llm_edge_case_6()

```
llm_edge_case_6 ( )
```

Matched-length lists: inferred as Columnar (Zip).

11.31.1.11 test_job_01_convert_epub()

```
test_job_01_convert_epub (
              book_data )
```

Tests.

Test EPUB -> TEI conversion for multiple books.

11.31.1.12 test_job_02_parse_chapters()

```
test_job_02_parse_chapters (
              book_data )
```

Test TEI -> Story parsing for multiple books.

11.31.1.13 test_job_03_chunk_story()

```
test_job_03_chunk_story (
              book_data )
```

Test Story -> chunks splitting for multiple books.

11.31.1.14 test_job_10_random_chunk()

```
test_job_10_random_chunk (
              book_data )
```

Test selecting a single random chunk.

11.31.1.15 test_job_10_sample_chunks()

```
test_job_10_sample_chunks (
              book_data )
```

Test sampling multiple chunks from a list.

11.31.1.16 test_job_11_send_chunk()

```
test_job_11_send_chunk (
              docs_db,
              book_data )
```

Test inserting chunk into MongoDB collection.

11.31.1.17 test_job_13_concatenate_triples()

```
test_job_13_concatenate_triples (
              book_data )
```

Test converting extracted triples to newline-delimited string.

11.31.1.18 test_job_15_comprehensive()

```
test_job_15_comprehensive (
              llm_data )
```

Test parsing malformed LLM output.

11.31.1.19 test_job_15_sanitize_triples_llm()

```
test_job_15_sanitize_triples_llm (
              book_data )
```

Test parsing LLM output JSON into triples list.

11.31.1.20 test_job_20_send_triples()

```
test_job_20_send_triples (
              main_graph,
              book_data )
```

Test inserting triples into knowledge graph.

11.31.1.21 test_job_21_describe_graph()

```
test_job_21_describe_graph (
              main_graph,
              book_data )
```

Test generating edge count summary of knowledge graph.

11.31.1.22 test_job_22_verbalize_triples()

```
test_job_22_verbalize_triples (
              main_graph,
              book_data )
```

Test converting high-degree triples to string format.

11.31.1.23 test_job_31_send_summary()

```
test_job_31_send_summary (
            docs_db,
            book_data )
```

Test updating chunk with summary in MongoDB.

11.31.1.24 test_pipeline_A_from_csv()

```
test_pipeline_A_from_csv ( )
```

Read example CSV and run pipeline_A for each row.

- Excel -> Save As -> CSV (UTF-8)
- Pandas will convert all blanks to None, so we must undo using fillna.

11.31.1.25 test_pipeline_A_minimal()

```
test_pipeline_A_minimal (
            book_data )
```

Minimal aggregate test.

Test running the aggregate pipeline_A on a single book.

11.31.1.26 test_pipeline_C_minimal()

```
test_pipeline_C_minimal (
            main_graph,
            book_data )
```

Test running pipeline_C with smoke test data.

11.31.1.27 test_pipeline_E_minimal_full_payload()

```
test_pipeline_E_minimal_full_payload (
            book_data )
```

Test running pipeline_E with full payload including metrics.

11.31.1.28 test_pipeline_E_minimal_summary_only()

```
test_pipeline_E_minimal_summary_only (
            book_data )
```

Test running pipeline_E with summary-only mode.

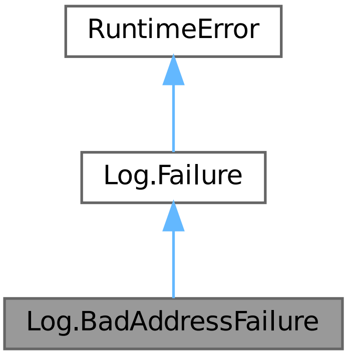
Chapter 12

Class Documentation

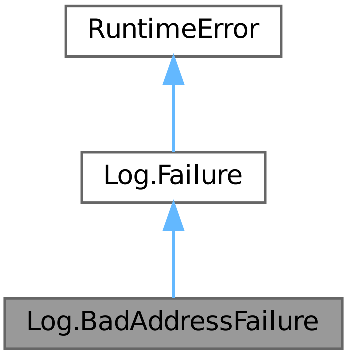
12.1 Log.BadAddressFailure Class Reference

Raised when a database connection string or address is invalid.

Inheritance diagram for Log.BadAddressFailure:



Collaboration diagram for Log.BadAddressFailure:



Public Member Functions

- __init__ (self, str source_prefix, str connection_string)

Public Member Functions inherited from Log.Failure

- __str__ (self)

Additional Inherited Members

Public Attributes inherited from Log.Failure

- prefix
- msg

12.1.1 Detailed Description

Raised when a database connection string or address is invalid.

- We support 4+ database engines and 2 endpoint frameworks (Blazor & Flask), each of which has a different error type when unable to connect.
- To avoid flooding the console with these, this error should not be chained.
- Usage: raise BadAddressFailure(source_prefix, connection_string) from None

12.1.2 Constructor & Destructor Documentation

12.1.2.1 __init__()

```
__init__ (
              self,
             str source_prefix,
             str connection_string )
```

Reimplemented from Log.Failure.

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

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

12.2 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)

12.2.1 Constructor & Destructor Documentation

12.2.1.1 __init__()

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

12.2.2 Member Function Documentation

12.2.2.1 stream_chapters()

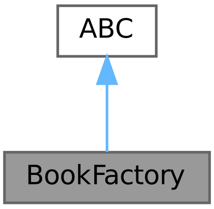
```
Iterator[Tuple[str, Dict[str, Any]]] stream_chapters (
              self )
```

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

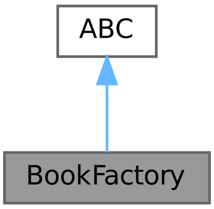
- /home/runner/work/dsci-capstone/dsci-capstone/src/components/book_conversion.py

12.3 BookFactory Class Reference

Inheritance diagram for BookFactory:



Collaboration diagram for BookFactory:



Public Member Functions

- Book create_book (self)

12.3.1 Member Function Documentation

12.3.1.1 create_book()

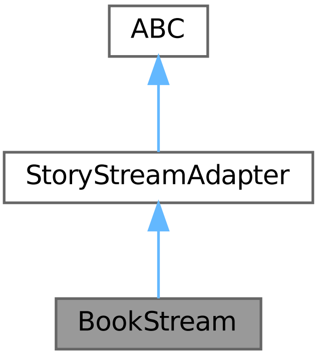
```
Book create_book (
               self )
```

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

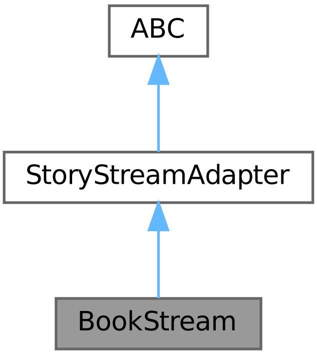
- /home/runner/work/dsci-capstone/dsci-capstone/src/components/book_conversion.py

12.4 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

12.4.1 Constructor & Destructor Documentation

12.4.1.1 __init__()

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

12.4.2 Member Function Documentation

12.4.2.1 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.

12.4.3 Member Data Documentation

12.4.3.1 book

```
book
```

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

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

12.5 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

12.5.1 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.

12.5.2 Constructor & Destructor Documentation

12.5.2.1 __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.

12.5.3 Member Function Documentation

12.5.3.1 __repr__()

```
str __repr__ (
              self )
```

12.5.3.2 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.

12.5.3.3 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.

12.5.3.4 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.

12.5.4 Member Data Documentation

12.5.4.1 book_id

```
book_id
```

12.5.4.2 chapter_number

```
chapter_number
```

12.5.4.3 chapter_percent

```
chapter_percent
```

12.5.4.4 line_end

```
line_end
```

12.5.4.5 line_start

```
line_start
```

12.5.4.6 story_id

```
story_id
```

12.5.4.7 story_percent

```
story_percent
```

12.5.4.8 text

```
text
```

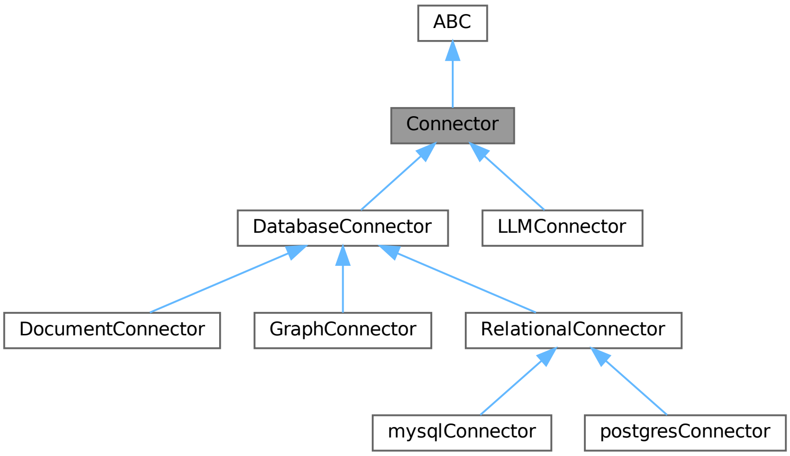
The documentation for this class was generated from the following file:

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

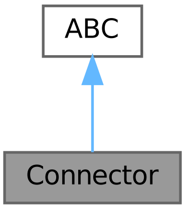
12.6 Connector Class Reference

Abstract base class for external connectors.

Inheritance diagram for Connector:



Collaboration diagram for Connector:



Public Member Functions

- bool test_operations (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.

12.6.1 Detailed Description

Abstract base class for external connectors.

Note

Credentials are specified in the .env file.

Derived classes should implement:

- **init**
- src.connectors.base.Connector.test_operations
- src.connectors.base.Connector.execute_query
- src.connectors.base.Connector.execute_file

12.6.2 Member Function Documentation

12.6.2.1 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 LLMConnector, DocumentConnector, GraphConnector, and RelationalConnector.

12.6.2.2 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.

12.6.2.3 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, DocumentConnector, LLMConnector, RelationalConnector, and GraphConnector.

12.6.2.4 test_operations()

```
bool test_operations (
              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 DocumentConnector, GraphConnector, LLMConnector, and RelationalConnector.

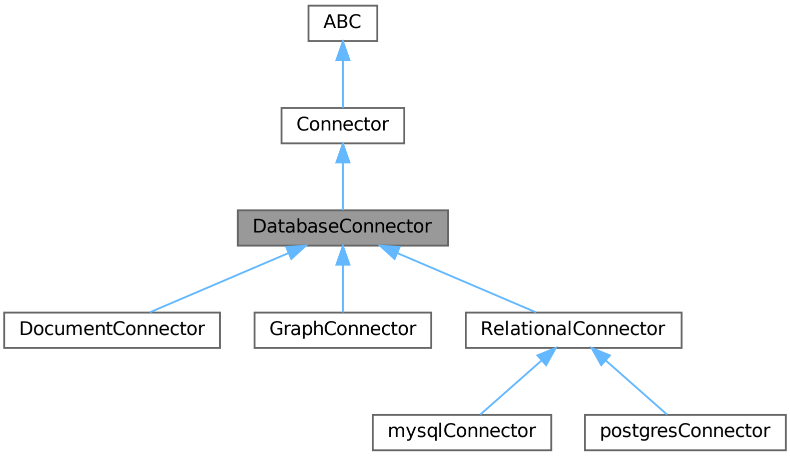
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/base.py

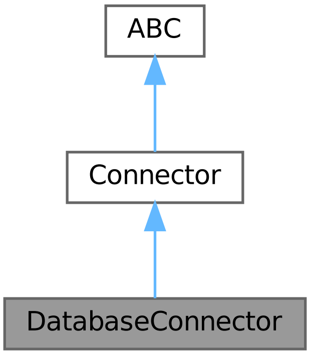
12.7 DatabaseConnector Class Reference

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.
- 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_operations (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.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Any result)

  Checks if the result of a query is valid (i.e.

12.7.1 Detailed Description

Abstract base class for database engine connectors.

Derived classes should implement:

- src.connectors.base.DatabaseConnector.__init__
- src.connectors.base.DatabaseConnector.test_operations
- src.connectors.base.DatabaseConnector.execute_query
- src.connectors.base.DatabaseConnector._split_combined
- src.connectors.base.DatabaseConnector.get_dataframe
- src.connectors.base.DatabaseConnector.create_database
- src.connectors.base.DatabaseConnector.drop_database
- src.connectors.base.DatabaseConnector.change_database
- src.connectors.base.DatabaseConnector.database_exists

12.7.2 Constructor & Destructor Documentation

12.7.2.1 __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 src.connectors.base.DatabaseConnector.configure() is called.

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

12.7.3 Member Function Documentation

12.7.3.1 _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).

12.7.3.2 _parsable_to_df()

```
bool _parsable_to_df (
              self,
             Any result )   [protected]
```

Checks if the result of a query is valid (i.e.

can be converted to a Pandas DataFrame).

Parameters

result	The result of a SQL, Cypher, or JSON query.

Returns

Whether the object is parsable to DataFrame.

Reimplemented in DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.3 _returns_data()

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

Checks if a query is structured in a way that returns real data, and not status messages.

Parameters

query	A single query string.

Returns

Whether the query is intended to fetch data (true) or might return a status message (false).

Reimplemented in DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.4 _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 DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.5 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 DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.6 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.

12.7.3.7 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 DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.8 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 in DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.9 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.

12.7.3.10 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.

12.7.3.11 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.

12.7.3.12 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 DocumentConnector, RelationalConnector, and GraphConnector.

12.7.3.13 get_dataframe()

```
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

Reimplemented in DocumentConnector, GraphConnector, and RelationalConnector.

12.7.3.14 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.

12.7.4 Member Data Documentation

12.7.4.1 connection_string

```
connection_string
```

12.7.4.2 db_engine

```
db_engine
```

12.7.4.3 db_type

```
db_type
```

12.7.4.4 host

```
host
```

12.7.4.5 password

```
password
```

12.7.4.6 port

```
port
```

12.7.4.7 username

```
username
```

12.7.4.8 verbose

```
verbose
```

Whether to print debug messages.

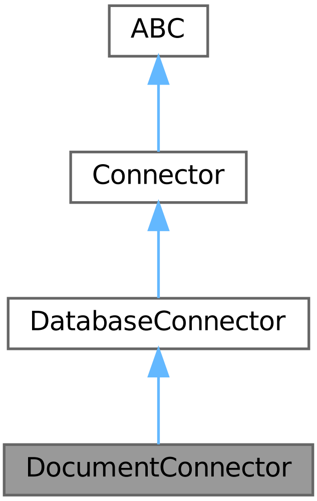
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/base.py

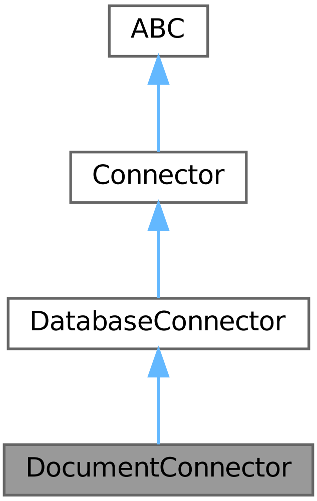
12.8 DocumentConnector Class Reference

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_operations (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=True)

  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.
- 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.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Any result)

  Checks if the result of a query is valid (i.e.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

Protected Attributes

- _auth_suffix

12.8.1 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).

12.8.2 Constructor & Destructor Documentation

12.8.2.1 __init__()

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

Creates a new MongoDB connector.

Parameters

verbose	Whether to print debug messages.

Reimplemented from DatabaseConnector.

12.8.3 Member Function Documentation

12.8.3.1 _parsable_to_df()

```
bool _parsable_to_df (
              self,
             Any result )  [protected]
```

Checks if the result of a query is valid (i.e.

can be converted to a Pandas DataFrame).

- Handles cursor-style dicts (with 'cursor' or 'firstBatch'), list-of-dict results, and single-document results.
- Excludes pure status/meta responses like {'ok': 1}.

Parameters

result	The result of a JSON query.

Returns

Whether the object is parsable to DataFrame.

Reimplemented from DatabaseConnector.

12.8.3.2 _returns_data()

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

Checks if a query is structured in a way that returns real data, and not status messages.

Determines whether a MongoDB command should yield cursor data.

- Uses an exclusion list - commands that definitely return a status.
- Everything else falls through to execution for validation.

Parameters

query	A single pre-validated JSON query string.

Returns

Whether the query is intended to fetch data (true) or might return a status message (false).

Reimplemented from DatabaseConnector.

12.8.3.3 _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.

12.8.3.4 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.

12.8.3.5 check_connection()

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

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.

12.8.3.6 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.

12.8.3.7 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.

12.8.3.8 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.

12.8.3.9 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.

12.8.3.10 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.

12.8.3.11 get_dataframe()

```
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

Sorted DataFrame containing the requested data

Exceptions

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

Reimplemented from DatabaseConnector.

12.8.3.12 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.

12.8.3.13 test_operations()

```
bool test_operations (
              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.

12.8.4 Member Data Documentation

12.8.4.1 _auth_suffix

```
_auth_suffix  [protected]
```

12.8.4.2 connection_string

```
connection_string
```

12.8.4.3 database_name

```
database_name
```

12.8.4.4 verbose

```
verbose
```

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

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/document.py

12.9 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.

12.9.1 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.

12.9.2 Constructor & Destructor Documentation

12.9.2.1 __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

12.9.3 Member Function Documentation

12.9.3.1 _prune_bad_tags()

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

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

12.9.3.2 _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_'.

12.9.3.3 clean_tei()

```
None clean_tei (
             self )
```

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

12.9.3.4 convert_to_tei()

```
None convert_to_tei (
             self )
```

Uses Pandoc to draft a TEI string from EPUB.

12.9.4 Member Data Documentation

12.9.4.1 clean_tei_content

```
clean_tei_content
```

12.9.4.2 encoding

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

12.9.4.3 epub_path

```
epub_path
```

12.9.4.4 pandoc_xml_path

```
pandoc_xml_path
```

12.9.4.5 raw_tei_content

```
raw_tei_content
```

12.9.4.6 tei_path

```
tei_path
```

12.9.4.7 xml_namespace

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

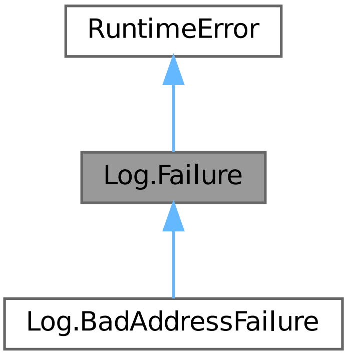
The documentation for this class was generated from the following file:

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

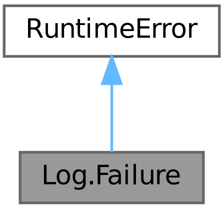
12.10 Log.Failure Class Reference

User-facing base class for custom error handling.

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

12.10.1 Detailed Description

User-facing base class for custom error handling.

- Builder Pattern - User can combine and chain standard message strings from the Log class.
- Prefixes (e.g., "GRAPH DB:", "FILE IO:") are redundant with tracebacks but improve readability by highlighting the semantic source of the error - not just a line number.
- Enforces a consistent color scheme across all raised errors for quick scanning.

12.10.2 Constructor & Destructor Documentation

12.10.2.1 __init__()

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

Reimplemented in Log.BadAddressFailure.

12.10.3 Member Function Documentation

12.10.3.1 __str__()

```
__str__ (
            self )
```

12.10.4 Member Data Documentation

12.10.4.1 msg

```
msg
```

12.10.4.2 prefix

```
prefix
```

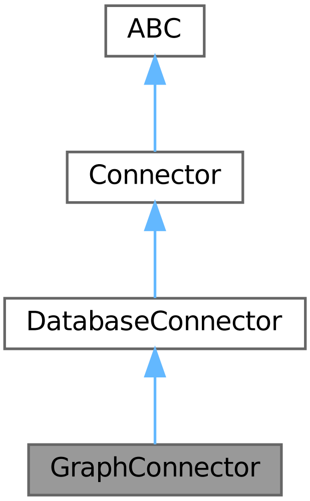
The documentation for this class was generated from the following file:

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

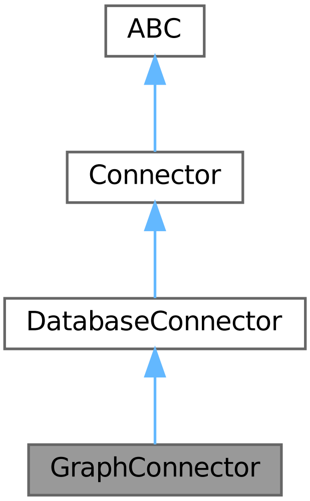
12.11 GraphConnector Class Reference

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_operations (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=True)

  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.
- 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.
- bool graph_exists (self, str graph_name)

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

  Delete the initial dummy node from the database.
- 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.

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 CQL queries, ignoring comments.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Tuple[Any, Any] result)

  Checks if the result of a Neo4j query is valid (i.e.
- None _execute_retag_db (self)

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

Protected Attributes

- _graph_name

12.11.1 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.

12.11.2 Constructor & Destructor Documentation

12.11.2.1 __init__()

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

Creates a new Neo4j connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from DatabaseConnector.

12.11.3 Member Function Documentation

12.11.3.1 _execute_retag_db()

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

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

12.11.3.2 _fetch_latest()

```
Tuple[Optional[List[Tuple[Any, ...]]], Optional[List[str]]] _fetch_latest (
              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.

12.11.3.3 _parsable_to_df()

```
bool _parsable_to_df (
              self,
             Tuple[Any, Any] result )  [protected]
```

Checks if the result of a Neo4j query is valid (i.e.

can be converted to a Pandas DataFrame).

- Validates shape: (records, meta)
- Validates content: rows are iterable, elements are dict-like or have .__properties__ (NeoModel Node/Rel)

Parameters

result	The result of a Cypher query.

Returns

Whether the object is parsable to DataFrame.

Reimplemented from DatabaseConnector.

12.11.3.4 _returns_data()

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

Checks if a query is structured in a way that returns real data, and not status messages.

Determines whether a Cypher query should yield records.

- RETURN must be present as a keyword (not in a string value) to return data.
- YIELD is used for stored procedures, and might return data.

Parameters

query	A single pre-validated CQL query string.

Returns

Whether the query is intended to fetch data (true) or might return a status message (false).

Reimplemented from DatabaseConnector.

12.11.3.5 _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.

12.11.3.6 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.

12.11.3.7 check_connection()

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

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.

12.11.3.8 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 src.connectors.base.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.

12.11.3.9 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.

12.11.3.10 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.

12.11.3.11 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.

12.11.3.12 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.

12.11.3.13 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. Internal helper functions need unfiltered results.

Returns

DataFrame containing the result of the query, or None

Exceptions

Log.Failure	If the query fails to execute.

Reimplemented from DatabaseConnector.

12.11.3.14 get_dataframe()

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

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

- Fetches all nodes and relationships belonging to the active database + graph name.

- Includes public attributes, element_id, labels, and element_type.
- Uses execute_query() for DataFrame conversion and filtering.
- Does not explode lists or nested values.

Parameters

name	The name of an existing graph or subgraph.
columns	A list of column names to keep.

Returns

Sorted DataFrame containing the requested data.

Exceptions

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

Reimplemented from DatabaseConnector.

12.11.3.15 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.

12.11.3.16 graph_exists()

```
bool graph_exists (
              self,
             str graph_name )
```

Search for an existing graph using the provided name.

Parameters

graph_name	The name of a graph to search for.

Returns

Whether the graph is visible to this connector.

12.11.3.17 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.

12.11.3.18 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.

12.11.3.19 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.

12.11.3.20 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.

12.11.3.21 test_operations()

```
bool test_operations (
             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.

12.11.4 Member Data Documentation

12.11.4.1 _graph_name

```
_graph_name  [protected]
```

12.11.4.2 connection_string

```
connection_string
```

12.11.4.3 database_name

```
database_name
```

12.11.4.4 verbose

```
verbose
```

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

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/graph.py

12.12 KnowledgeGraph Class Reference

Manages a single graph within Neo4j.

Public Member Functions

- None __init__ (self, str name, GraphConnector database, bool verbose=False)
- None add_triple (self, str subject, str relation, str object_)

  Add a semantic triple to the graph using raw Cypher.
- None add_triples_json (self, List[Triple] triples_json)

  Add several semantic triples to the graph from pre-verified JSON.
- DataFrame get_all_triples (self)

  Return all triples in the specified graph as a pandas DataFrame.
- Optional[DataFrame] get_triple_properties (self)

  Pivot the graph elements DataFrame to expose node and relationship properties as columns.
- DataFrame triples_to_names (self, DataFrame df_ids, bool drop_ids=False, Optional[DataFrame] df_↩ lookup=None)

  Maps a DataFrame containing element ID columns to human-readable names.
- DataFrame find_element_names (self, DataFrame df_ids, List[str] name_columns, List[str] id_columns, str element_type, str name_property, bool drop_ids=False, Optional[DataFrame] df_lookup=None)

  Helper function which maps element IDs to human-readable names.
- DataFrame get_subgraph_by_nodes (self, List[str] node_ids, List[str] id_columns=["subject_id", "object_id"])

  Return all triples where subject or object is in the specified node list.
- DataFrame get_neighborhood (self, str node_id, int depth=1)

  Get k-hop neighborhood around a central node.
- DataFrame get_degree_range (self, int min_degree=1, int max_degree=-1, List[str] id_columns=["subject_id", "object_id"])

  Return triples associated with nodes whose degree lies within the specified bounds.
- DataFrame get_by_ranked_degree (self, int best_rank=1, int worst_rank=-1, bool enforce_count=False, List[str] id_columns=["subject_id", "object_id"])

  Return triples associated with nodes whose degree rank lies in the specified range.
- DataFrame get_random_walk_sample (self, List[str] start_nodes, int walk_length, int num_walks=1)

  Sample subgraph using directed random walk traversal starting from specified nodes.
- DataFrame get_community_subgraph (self, int community_id)

  Return all triples belonging to a specific GraphRAG community.
- None detect_community_clusters (self, str method="leiden", bool multi_level=False, int max_levels=10)

  Run community detection on the graph as described by the GraphRAG paper.
- str to_triples_string (self, Optional[DataFrame] triple_names_df=None, str mode="triple")

  Convert triples to string representation in various formats.
- str to_contextualized_string (self, Optional[List[str]] focus_nodes=None, int top_n=5)

  Convert triples to contextualized string grouped by focus nodes.

- str to_narrative (self, str strategy="path", Optional[str] start_node=None, int max_triples=50)

  Convert graph to narrative text using specified strategy.
- Dict[str, Any] get_summary_stats (self)

  Return summary statistics about the graph structure.
- DataFrame get_edge_counts (self, int top_n=-1)

  Return node names and their edge counts, ordered by edge count descending.
- str get_node_context (self, str node_id, bool include_neighbors=True)

  Return natural language description of a node and its relationships.
- DataFrame get_relation_summary (self)

  Return summary of relationship types and their frequencies.
- 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.

Public Attributes

- graph_name

  The name of this graph.
- database

  Reference to a pre-configured graph database wrapper.
- verbose

  Whether to print debug messages.

Protected Attributes

- _first_insert

  Flag to drop any existing graph when the first triple is added.

12.12.1 Detailed Description

Manages a single graph within Neo4j.

- Handles safe conversion of LLM output to structured triples.
- Provides helper functions to add and retrieve triples.

12.12.2 Constructor & Destructor Documentation

12.12.2.1 __init__()

```
None __init__ (
              self,
             str name,
             GraphConnector database,
             bool  verbose = False )
```

12.12.3 Member Function Documentation

12.12.3.1 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 src.connectors.llm.LLMConnector.normalize_triples.

Exceptions

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

12.12.3.2 add_triples_json()

```
None add_triples_json (
             self,
            List[Triple] triples_json )
```

Add several semantic triples to the graph from pre-verified JSON.

Note

JSON should be pre-normalized using src.connectors.llm.normalize_triples.

Parameters

triples_json	A list of Triple dictionaries containing keys: 's', 'r', and 'o'.

Exceptions

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

12.12.3.3 detect_community_clusters()

```
None detect_community_clusters (
              self,
             str  method = "leiden",
             bool  multi_level = False,
             int  max_levels = 10 )
```

Run community detection on the graph as described by the GraphRAG paper.

- Assigns a `community_id` property to all nodes, and optionally `level_id`.
- Partitions the graph's nodes into topic-coherent communities.
- Afterwards, you can call `get_community_subgraph()` to extract each community's triples for summarization. Clustering Methods
- Leiden (recommended) - improvement of Louvain ensuring well-connected, stable communities; supports multi-level hierarchy.
- Louvain - quickly groups nodes but may yield fragmented subcommunities.

Parameters

method	The community detection algorithm to run. Options: "leiden" (default) or "louvain".
multi_level	Whether to record hierarchical levels (`level_id`) for multi-scale summarization.
max_levels	Maximum hierarchy depth to compute (default: 10).

Exceptions

Log.Failure	If GDS is unavailable or any query fails.

12.12.3.4 find_element_names()

```
DataFrame find_element_names (
              self,
             DataFrame df_ids,
             List[str] name_columns,
             List[str] id_columns,
             str element_type,
             str name_property,
             bool  drop_ids = False,
             Optional[DataFrame]  df_lookup = None )
```

Helper function which maps element IDs to human-readable names.

Note

- Requires the provided nodes or edges to still exist in the graph database; otherwise must specify df_lookup.

Parameters

df_ids	DataFrame with required columns: *id_columns*.
name_columns	Required list of column names to create.
id_columns	Required list of columns containing element IDs.
element_type	Whether to match nodes or relationships. Value must be "node" or "relationship".
name_property	Required element property from *df_lookup* to use as the display name.
drop_ids	Whether to remove *id_columns* from results.
df_lookup	Optional DataFrame fetched from src.connectors.graph.GraphConnector.get_dataframe with required columns: element_id, elemenet_type, and *name_property*.

Returns

DataFrame with added columns: *name_columns*.

Exceptions

Log.Failure	If mapping fails or required IDs are missing.

12.12.3.5 get_all_triples()

```
DataFrame get_all_triples (
              self )
```

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

Returns

Returns (subject, relation, object) columns only.

Exceptions

Log.Failure	If the query fails to retrieve or process the DataFrame.

12.12.3.6 get_by_ranked_degree()

```
DataFrame get_by_ranked_degree (
              self,
             int  best_rank = 1,
             int  worst_rank = -1,
             bool  enforce_count = False,
             List[str]  id_columns = ["subject_id", "object_id"] )
```

Return triples associated with nodes whose degree rank lies in the specified range.

- Computes degree (edge count) for all nodes.
- Sorts nodes by degree descending, assigns ranks, and selects those with best_rank <= rank <= worst_rank.
- Returns all triples where subject_id or object_id matches a selected node.

Parameters

best_rank	Minimum degree rank. Inclusive.
worst_rank	Maximum degree rank (-1 = maximum degree) to include. Inclusive.
enforce_count	Always return (worst_rank - best_rank + 1) rows (fallback to node_id order).
id_columns	List of columns to compare against. Can be 'subject_id', 'object_id', or both.

Returns

DataFrame containing the triples for ranked nodes; columns: subject_id, relation_id, object_id.

Exceptions

Log.Failure	If the graph cannot be queried.
ValueError	If best_rank or worst_rank values are invalid.

12.12.3.7 get_community_subgraph()

```
DataFrame get_community_subgraph (
              self,
             int community_id )
```

Return all triples belonging to a specific GraphRAG community.

- Communities are densely connected subgraphs detected via clustering algorithms.
- This enables GraphRAG-style hierarchical summarization where each community can be summarized independently. Requires nodes to have a 'community_id' property assigned.
- Afterwards, you may run a summary step which generates community summaries for each cluster (as described in the paper).

Parameters

community↩ _id	The identifier of the community to retrieve.

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

Log.Failure	If the query fails to retrieve the requested DataFrame or community detection has not been run.

12.12.3.8 get_degree_range()

```
DataFrame get_degree_range (
             self,
            int  min_degree = 1,
            int  max_degree = -1,
            List[str]  id_columns = ["subject_id", "object_id"] )
```

Return triples associated with nodes whose degree lies within the specified bounds.

- Degree is defined as the number of relationships where a node appears as start_node_id or end_node_id.
- Selects all nodes satisfying min_degree <= degree <= max_degree and returns triples incident to those nodes.

Parameters

max_degree	Maximum number of edges allowed for a node to be included (-1 = infer highest edge count).
min_degree	Minimum number of edges required for a node to be included.
id_columns	List of columns to compare against. Can be 'subject_id', 'object_id', or both.

Returns

DataFrame containing an arbitrary number of triples for nodes in the specified degree range.

Exceptions

Log.Failure	If the graph fails to load or degree computation fails.
ValueError	If min_degree or max_degree values are invalid.

12.12.3.9 get_edge_counts()

```
DataFrame get_edge_counts (
             self,
            int  top_n = -1 )
```

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 -1 (all nodes).

Returns

DataFrame with columns: node_id, edge_count

Exceptions

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

12.12.3.10 get_neighborhood()

```
DataFrame get_neighborhood (
              self,
             str node_id,
             int  depth = 1 )
```

Get k-hop neighborhood around a central node.

Returns all triples within k hops of the specified node. A 1-hop neighborhood includes all direct neighbors, 2-hop includes neighbors-of-neighbors, etc.

Parameters

node←↩ _id	The element ID of the central node.
depth	Number of hops to traverse (default: 1).

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

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

12.12.3.11 get_node_context()

```
str get_node_context (
              self,
             str node_id,
             bool  include_neighbors = True )
```

Return natural language description of a node and its relationships.

Generates a human-readable summary of a single node suitable for LLM context. Example: "Alice is connected to 5 entities. She knows Bob and Charlie, works at Company, lives in City, and follows Dave."

Parameters

node_id	The element ID of the node to describe.
include_neighbors	Whether to list neighbor node IDs (default: True).

Returns

Natural language description of the node.

Exceptions

Log.Failure	If the node does not exist in the graph.

12.12.3.12 get_random_walk_sample()

```
DataFrame get_random_walk_sample (
              self,
             List[str] start_nodes,
             int walk_length,
             int  num_walks = 1 )
```

Sample subgraph using directed random walk traversal starting from specified nodes.

- More diverse than degree-based filtering (nodes with many edges) and better preserves graph structure.
- Each walk starts from a random node in start_nodes and continues for walk_length steps.

Parameters

start_nodes	List of node IDs to use as starting points.
walk_length	Number of steps in each random walk.
num_walks	Number of random walks to perform (default: 1).

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

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

12.12.3.13 get_relation_summary()

```
DataFrame get_relation_summary (
              self )
```

Return summary of relationship types and their frequencies.

Provides an overview of what types of relationships exist in the graph and how common each type is. Useful for understanding graph schema.

Returns

DataFrame with columns: relation_type, count, example_triple

12.12.3.14 get_subgraph_by_nodes()

```
DataFrame get_subgraph_by_nodes (
              self,
             List[str] node_ids,
             List[str]  id_columns = ["subject_id", "object_id"] )
```

Return all triples where subject or object is in the specified node list.

Parameters

node_ids	List of node element IDs to filter by.
id_columns	List of columns to compare against. Can be 'subject_id', 'object_id', or both.

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

Log.Failure	If the query fails to retrieve the requested DataFrame.
KeyError	If the provided column names are invalid.

12.12.3.15 get_summary_stats()

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

Return summary statistics about the graph structure.

Provides metadata useful for LLM context, including:

- node_count: Total number of nodes
- edge_count: Total number of relationships
- relation_types: List of unique relationship types
- avg_degree: Average node degree (edges per node)
- top_nodes: List of most connected nodes (top 5 by degree)
- density: Graph density (actual edges / possible edges)

  Returns

  Dictionary containing graph statistics.

12.12.3.16 get_triple_properties()

```
Optional[DataFrame] get_triple_properties (
              self )
```

Pivot the graph elements DataFrame to expose node and relationship properties as columns.

- Builds a joined view of properties from both nodes (n1, n2) and the relationship (r).
- Removes redundant fields such as: db, kg, element_type, start_node_id, and end_node_id.
- Usage: n1.element_id, r.rel_type, n2.name, etc.

  Returns

  DataFrame where each row represents one triple (n1, r, n2).

Exceptions

Log.Failure	If the elements DataFrame cannot be loaded or pivoting fails.

12.12.3.17 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.

12.12.3.18 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.

12.12.3.19 to_contextualized_string()

```
str to_contextualized_string (
              self,
             Optional[List[str]]  focus_nodes = None,
             int  top_n = 5 )
```

Convert triples to contextualized string grouped by focus nodes.

Groups triples by subject nodes and formats them with context headers. This provides better structure for LLM comprehension compared to flat triple lists. If focus_nodes is None, uses the top_n most connected nodes. Example output: Facts about Alice:

- knows Bob
- works_at Company
- lives_in City

Parameters

focus_nodes	List of node names to group by. If None, uses top_n by degree.
top_n	Number of top nodes to use if focus_nodes is None (default: 5).

Returns

Formatted string with contextualized triple groups.

12.12.3.20 to_narrative()

```
str to_narrative (
              self,
             str  strategy = "path",
             Optional[str]  start_node = None,
             int  max_triples = 50 )
```

Convert graph to narrative text using specified strategy.

Transforms structured triples into natural language narrative:

- "path": Follow edges sequentially from start_node, creating a story-like flow
- "cluster": Group related entities and describe them thematically
- "summary": High-level overview of graph contents and structure

Parameters

strategy	Narrative generation strategy: "path", "cluster", or "summary" (default: "path").
start_node	Starting node for "path" strategy. If None, uses highest-degree node.
max_triples	Maximum number of triples to include (default: 50).

Returns

Natural language narrative describing the graph.

Exceptions

ValueError	If strategy is not recognized.

12.12.3.21 to_triples_string()

```
str to_triples_string (
              self,
             Optional[DataFrame]  triple_names_df = None,
             str  mode = "triple" )
```

Convert triples to string representation in various formats.

Supports multiple output formats for LLM consumption:

- "natural": Human-readable sentences (e.g., "Alice employed by Bob.")
- "triple": Raw triple format (e.g., "Alice employedBy Bob")
- "json": JSON array of objects with s/r/o keys

Parameters

triple_names↩ _df	DataFrame with subject, relation, object columns. If None, uses all triples from this graph.
mode	Output format: "natural", "triple", or "json" (default: "triple").

Returns

String representation of triples in the specified format.

Exceptions

ValueError	If format is not recognized.

12.12.3.22 triples_to_names()

```
DataFrame triples_to_names (
              self,
             DataFrame df_ids,
             bool  drop_ids = False,
             Optional[DataFrame]  df_lookup = None )
```

Maps a DataFrame containing element ID columns to human-readable names.

Note

- Requires the provided nodes to still exist in the graph database; otherwise must specify df_lookup.

Parameters

df_ids	DataFrame with added columns: subject_id, relation_id, object_id.
drop_ids	Whether to remove columns from results: subject_id, relation_id, object_id.
df_lookup	Optional DataFrame fetched from src.connectors.graph.GraphConnector.get_dataframe with required columns: element_id, elemenet_type, name, and rel_type.

Returns

DataFrame with added columns: subject, relation, object.

Exceptions

Log.Failure	If mapping fails or required IDs are missing.

12.12.4 Member Data Documentation

12.12.4.1 _first_insert

```
_first_insert  [protected]
```

Flag to drop any existing graph when the first triple is added.

12.12.4.2 database

```
database
```

Reference to a pre-configured graph database wrapper.

12.12.4.3 graph_name

```
graph_name
```

The name of this graph.

Matches node.kg for all nodes in the graph database.

12.12.4.4 verbose

```
verbose
```

Whether to print debug messages.

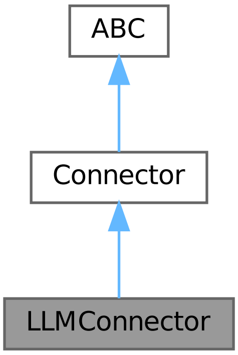
The documentation for this class was generated from the following file:

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

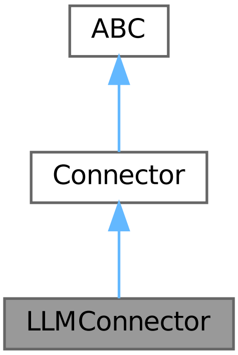
12.13 LLMConnector Class Reference

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

Inheritance diagram for LLMConnector:



Collaboration diagram for LLMConnector:



Public Member Functions

- None __init__ (self, float temperature=0, str system_prompt="You are a helpful assistant.")

  Initialize the connector.
- None configure (self)

  Initialize the LangChain LLM using environment credentials.
- bool test_operations (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)

  Send a trivial prompt to verify LLM connectivity.
- 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
- llm
- system_prompt

12.13.1 Detailed Description

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

Note

> The method src.connectors.llm.LLMConnector.execute_query simplifies the prompt process.

To implement various configurations, either set properties directly or create another LLMConnector instance. Useful config options: temperature, system_prompt, llm, model_name. We prefer creating a separate wrapper instance for reusable hard-coded configurations.

12.13.2 Constructor & Destructor Documentation

12.13.2.1 __init__()

```
None __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.

12.13.3 Member Function Documentation

12.13.3.1 check_connection()

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

Send a trivial prompt to verify LLM connectivity.

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 prompt executed successfully.

Exceptions

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

Reimplemented from Connector.

12.13.3.2 configure()

```
None 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

12.13.3.3 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.

12.13.3.4 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.

12.13.3.5 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.

12.13.3.6 test_operations()

```
bool test_operations (
              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 prompt executed successfully.

Exceptions

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

Reimplemented from Connector.

12.13.4 Member Data Documentation

12.13.4.1 llm

```
llm
```

12.13.4.2 model_name

```
model_name
```

12.13.4.3 system_prompt

```
system_prompt
```

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

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/llm.py

12.14 Log Class Reference

The Log class standardizes console output.

Collaboration diagram for Log:

Log
msg_good_df_parse
msg_good_df_parse
Log

Classes

- class BadAddressFailure

  Raised when a database connection string or address is invalid.
- class Failure

  User-facing base class for custom error handling.

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="WARN: ", 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.
- None time_message (str prefix="[TIME] ", str msg="", bool verbose=True)

  A time message begins with a light blue prefix.
- None chart_message (str prefix="[CHART] ", str msg="", bool verbose=True)

  A chart message begins with a gray prefix.
- None chart (str title, str filename, bool verbose=True)

  Print the time taken to complete a function.
- None elapsed_time (str name, float seconds, str call_chain, bool verbose=True)

  Print the time taken to complete a function.
- str format_call_chain (List[FrameInfo] stack, str name)

  Sanitize and concatenate the full call stack for console output.
- Callable[..., Any] time (Callable[..., Any] func)

Logs the time elapsed for a function call.

- Generator[None, None, None] timer (str name=None)

  Context manager for recording the execution time of code blocks.
- DataFrame get_timing_summary ()

  Returns timing results as a pandas DataFrame.
- DataFrame get_merged_timing (str file_path="./logs/elapsed_time.csv")

  Reads the existing file, deletes rows matching this run_id, and adds current data.
- None dump_timing_csv (str file_path="./logs/elapsed_time.csv")

  Save timing results to a CSV file, appending if it already exists.
- clear_timing_data ()

  Clears all recorded timing data.
- print_timing_summary ()

  Prints a formatted timing summary grouped by function.

Static Public Attributes

- bool USE_COLORS = True

  Enable ANSI colors in output.
- bool RECORD_TIME = True

  Enable time-logging with the 'Log.time' decorator.
- bool FULL_DF = False

  Print the entire DataFrame to console.
- 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 CYAN = "\033[96m"

  ANSI code for light blue.
- str GRAY = "\033[90m"

  ANSI code for light gray.
- 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 TIME_COLOR = CYAN

  ANSI color applied to the prefix of time-elapsed messages.
- str CHART_COLOR = GRAY

  ANSI color applied to the prefix of chart generation messages.
- str MSG_COLOR = BRIGHT

  ANSI color applied to the body of every Log message.
- f msg_chart_saved = lambda title, filename"Saved chart '{title}' to {filename}"
- f msg_elapsed_time = lambda name, seconds"{name} took {seconds:.3f}s"

- int run_id = 1
- str t_dump = "[DUMP] "
- f msg_time_dump = lambda file_path"Saved time records to '{file_path}'"
- 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_ops = "OPERATE: "
- str test_basic = "CONNECT: "
- 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}'"
- tuple msg_fail_manage_gr
- 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}'"
- Log msg_good_df_parse = lambda df.msg_result(df)
- f msg_bad_df_parse = lambda query"Failed to convert query result to DataFrame:\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: "
- f msg_none_df = lambda collection_type, collection_name"Unable to fetch DataFrame from {collection_type} '{collection_name}' - None"
- str sub_gr = "SUBGRAPH: "
- str gr_rag = "RAG: "
- f msg_bad_triples = lambda graph_name"No triples found for graph {graph_name}"

Static Protected Attributes

- list _timing_results = [ ]

12.14.1 Detailed Description

The Log class standardizes console output.

12.14.2 Member Function Documentation

12.14.2.1 chart()

```
None chart (
            str title,
            str filename,
            bool  verbose = True )  [static]
```

Print the time taken to complete a function.

Parameters

title	The title of the chart.
filename	Where the chart was saved to.
verbose	Whether to actually print. Saves space and reduces nested if statements.

12.14.2.2 chart_message()

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

A chart message begins with a gray 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.

12.14.2.3 clear_timing_data()

```
clear_timing_data ( )  [static]
```

Clears all recorded timing data.

12.14.2.4 dump_timing_csv()

```
None dump_timing_csv (
             str  file_path = "./logs/elapsed_time.csv" )  [static]
```

Save timing results to a CSV file, appending if it already exists.

Parameters

file_path	Where the saved CSV will be located.

Returns

DataFrame with columns: function, elapsed, call_chain

12.14.2.5 elapsed_time()

```
None elapsed_time (
             str name,
             float seconds,
             str call_chain,
             bool  verbose = True )  [static]
```

Print the time taken to complete a function.

Parameters

name	The name of the function.
seconds	The number of seconds (will be rounded to 3 decimals)
call_chain	The full stack of function calls (for record-keeping)
verbose	Whether to actually print. Saves space and reduces nested if statements.

12.14.2.6 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

12.14.2.7 fail_legacy()

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

A legacy failure message begins with a Red X.

Parameters

msg	The message to print.

12.14.2.8 format_call_chain()

```
str format_call_chain (
             List[FrameInfo] stack,
             str name )  [static]
```

Sanitize and concatenate the full call stack for console output.

Parameters

stack	The frame stack obtained by inspect.stack().
name	The name of the caller function.

Returns

A string representing the full call chain.

12.14.2.9 get_merged_timing()

```
DataFrame get_merged_timing (
            str  file_path = "./logs/elapsed_time.csv" )  [static]
```

Reads the existing file, deletes rows matching this run_id, and adds current data.

Returns

DataFrame with columns: function, elapsed, call_chain, run_id

12.14.2.10 get_timing_summary()

```
DataFrame get_timing_summary ( )  [static]
```

Returns timing results as a pandas DataFrame.

Returns

DataFrame with columns: function, elapsed, call_chain, run_id

12.14.2.11 print_timing_summary()

```
print_timing_summary ( )  [static]
```

Prints a formatted timing summary grouped by function.

12.14.2.12 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.

12.14.2.13 success_legacy()

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

A legacy success message begins with a Green Plus.

Parameters

msg	The message to print.

12.14.2.14 time()

```
Callable[..., Any] time (
             Callable[..., Any] func )  [static]
```

Logs the time elapsed for a function call.

- Uses an inner wrapper function to capture ∗args and ∗∗kwargs.

Parameters

func	The function to wrap.

Returns

The wrapped function that logs time and forwards the result.

12.14.2.15 time_message()

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

A time message begins with a light blue 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.

12.14.2.16 timer()

```
Generator[None, None, None] timer (
             str  name = None )  [static]
```

Context manager for recording the execution time of code blocks.

Parameters

name	Optional name for the timed block. If not provided, uses caller function name. Usage: with Log.timer():

12.14.3 your code here

12.14.3.1 warn()

```
None warn (
             str  prefix = "WARN: ",
             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.

12.14.4 Member Data Documentation

12.14.4.1 _timing_results

```
list _timing_results = [ ]  [static], [protected]
```

12.14.4.2 bad_addr

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

12.14.4.3 bad_path

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

12.14.4.4 bad_val

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

12.14.4.5 BRIGHT

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

ANSI code for bright yellow / cream.

12.14.4.6 CHART_COLOR

```
str CHART_COLOR = GRAY  [static]
```

ANSI color applied to the prefix of chart generation messages.

12.14.4.7 conn_abc

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

12.14.4.8 create_db

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

12.14.4.9 CYAN

```
str CYAN = "\033[96m"  [static]
```

ANSI code for light blue.

12.14.4.10 db_conn_abc

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

12.14.4.11 db_exists

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

12.14.4.12 doc_db

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

12.14.4.13 drop_db

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

12.14.4.14 drop_gr

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

12.14.4.15 FAILURE_COLOR

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

ANSI color applied to the prefix of critical fail messages.

12.14.4.16 FULL_DF

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

Print the entire DataFrame to console.

12.14.4.17 get_df

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

12.14.4.18 get_unique

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

12.14.4.19 good_val

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

12.14.4.20 gr_db

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

12.14.4.21 gr_rag

```
str gr_rag = "RAG: "  [static]
```

12.14.4.22 GRAY

```
str GRAY = "\033[90m"  [static]
```

ANSI code for light gray.

12.14.4.23 GREEN

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

ANSI code for green text.

12.14.4.24 kg

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

12.14.4.25 msg_bad_addr

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

12.14.4.26 msg_bad_coll

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

12.14.4.27 msg_bad_df_parse

```
f msg_bad_df_parse = lambda query"Failed to convert query result to DataFrame:\n'{query}'"
[static]
```

12.14.4.28 msg_bad_exec_f

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

12.14.4.29 msg_bad_exec_q

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

12.14.4.30 msg_bad_graph

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

12.14.4.31 msg_bad_path

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

12.14.4.32 msg_bad_table

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

12.14.4.33 msg_bad_triples

```
f msg_bad_triples = lambda graph_name"No triples found for graph {graph_name}" [static]
```

12.14.4.34 msg_chart_saved

```
f msg_chart_saved = lambda title, filename"Saved chart '{title}' to {filename}" [static]
```

12.14.4.35 MSG_COLOR

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

ANSI color applied to the body of every Log message.

12.14.4.36 msg_compare

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

12.14.4.37 msg_db_connect

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

12.14.4.38 msg_db_current

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

12.14.4.39 msg_db_exists

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

12.14.4.40 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]
```

12.14.4.41 msg_elapsed_time

```
f msg_elapsed_time = lambda name, seconds"{name} took {seconds:.3f}s"  [static]
```

12.14.4.42 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}"
    )
```

12.14.4.43 msg_fail_manage_gr

```
tuple msg_fail_manage_gr  [static]
```

Initial value:

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

12.14.4.44 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]
```

12.14.4.45 msg_good_coll

```
tuple msg_good_coll  [static]
```

Initial value:

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

12.14.4.46 msg_good_df_parse

```
Log msg_good_df_parse = lambda df.msg_result(df)  [static]
```

12.14.4.47 msg_good_exec_f

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

12.14.4.48 msg_good_exec_q

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

12.14.4.49 msg_good_exec_qr

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

12.14.4.50 msg_good_graph

```
tuple msg_good_graph  [static]
```

Initial value:

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

12.14.4.51 msg_good_path

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

12.14.4.52 msg_good_table

```
tuple msg_good_table  [static]
```

Initial value:

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

12.14.4.53 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}"
    )
```

12.14.4.54 msg_none_df

```
f msg_none_df = lambda collection_type, collection_name"Unable to fetch DataFrame from {collection↩
_type} '{collection_name}' - None"  [static]
```

12.14.4.55 msg_result

```
tuple msg_result  [static]
```

Initial value:

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

12.14.4.56 msg_success_managed_db

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

12.14.4.57 msg_success_managed_gr

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

12.14.4.58 msg_swap_db

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

12.14.4.59 msg_swap_kg

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

12.14.4.60 msg_time_dump

```
f msg_time_dump = lambda file_path"Saved time records to '{file_path}'"  [static]
```

12.14.4.61 msg_unknown_error

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

12.14.4.62 pytest_db

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

12.14.4.63 RECORD_TIME

```
bool RECORD_TIME = True  [static]
```

Enable time-logging with the 'Log.time' decorator.

12.14.4.64 RED

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

ANSI code for red text.

12.14.4.65 rel_db

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

12.14.4.66 run_f

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

12.14.4.67 run_id

```
int run_id = 1  [static]
```

12.14.4.68 run_q

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

12.14.4.69 sub_gr

```
str sub_gr = "SUBGRAPH: "  [static]
```

12.14.4.70 SUCCESS_COLOR

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

ANSI color applied to the prefix of success messages.

12.14.4.71 swap_db

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

12.14.4.72 swap_kg

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

12.14.4.73 t_dump

```
str t_dump = "[DUMP] "  [static]
```

12.14.4.74 test_basic

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

12.14.4.75 test_df

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

12.14.4.76 test_info

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

12.14.4.77 test_ops

```
str test_ops = "OPERATE: "  [static]
```

12.14.4.78 test_tmp_db

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

12.14.4.79 TIME_COLOR

```
str TIME_COLOR = CYAN  [static]
```

ANSI color applied to the prefix of time-elapsed messages.

12.14.4.80 USE_COLORS

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

Enable ANSI colors in output.

12.14.4.81 WARNING_COLOR

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

ANSI color applied to the prefix of ignored fail messages.

12.14.4.82 WHITE

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

ANSI code to reset color.

12.14.4.83 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

12.15 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

12.15.1 Detailed Description

Utility class for computing and posting evaluation metrics.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 __init__()

```
None __init__ (
              self )
```

12.15.3 Member Function Documentation

12.15.3.1 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.

12.15.3.2 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.

12.15.3.3 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.

12.15.3.4 generate_example_metrics()

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

Create a placeholder payload with dummy values.

Returns

Full payload with nested metrics.

12.15.3.5 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.

12.15.3.6 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.

12.15.3.7 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.

12.15.4 Member Data Documentation

12.15.4.1 HOST

```
HOST
```

12.15.4.2 PORT

```
PORT
```

12.15.4.3 timeout_seconds

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

12.15.4.4 url

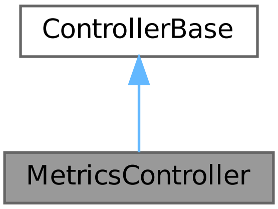
```
url
```

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

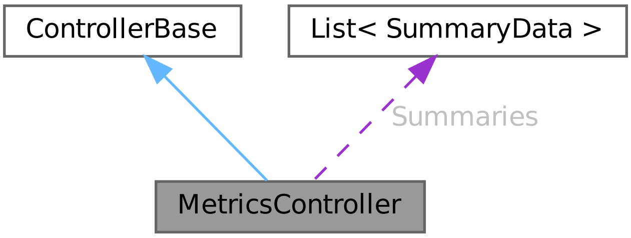
- /home/runner/work/dsci-capstone/dsci-capstone/src/components/metrics.py

12.16 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()

12.16.1 Constructor & Destructor Documentation

12.16.1.1 MetricsController()

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

12.16.2 Member Function Documentation

12.16.2.1 GetAll()

```
IActionResult GetAll ( )
```

12.16.2.2 GetIndex()

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

12.16.2.3 Post()

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

12.16.3 Member Data Documentation

12.16.3.1 _hubContext

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

12.16.3.2 _logger

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

12.16.3.3 Summaries

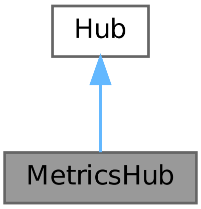
```
readonly List<SummaryData> Summaries = new()  [static], [private]
```

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

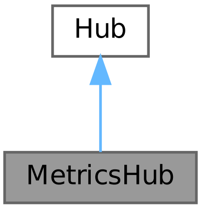
- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/MetricsController.cs

12.17 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

12.17.1 Constructor & Destructor Documentation

12.17.1.1 MetricsHub()

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

12.17.2 Member Function Documentation

12.17.2.1 OnConnectedAsync()

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

12.17.2.2 OnDisconnectedAsync()

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

12.17.3 Member Data Documentation

12.17.3.1 _logger

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

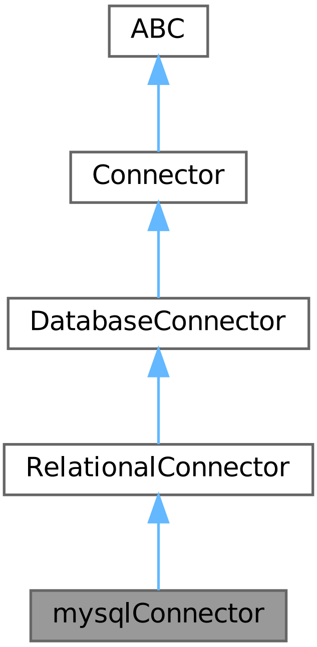
The documentation for this class was generated from the following file:

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

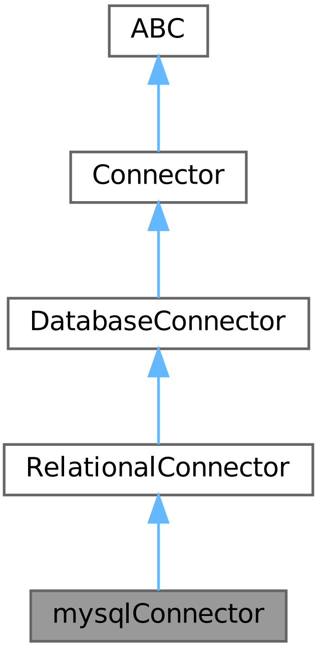
12.18 mysqlConnector Class Reference

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_operations (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=True)

  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.
- 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`.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Tuple[Any, Any] result)

  Checks if the result of a SQL query is valid (i.e.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

12.18.1 Detailed Description

A relational database connector configured for MySQL.

Note

Should be hidden from the user using a factory method.

12.18.2 Constructor & Destructor Documentation

12.18.2.1 __init__()

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

Configures the relational connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from RelationalConnector.

12.18.3 Member Data Documentation

12.18.3.1 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.
    }
```

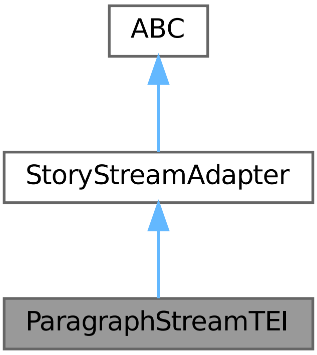
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/relational.py

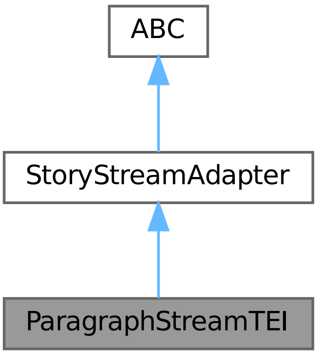
12.19 ParagraphStreamTEI Class Reference

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"

12.19.1 Detailed Description

Streams paragraphs from a TEI file as Chunk objects.

12.19.2 Constructor & Destructor Documentation

12.19.2.1 __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.

12.19.3 Member Function Documentation

12.19.3.1 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

12.19.3.2 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.

12.19.4 Member Data Documentation

12.19.4.1 allowed_chapters

```
allowed_chapters
```

12.19.4.2 book_id

```
book_id
```

12.19.4.3 chunks

```
chunks
```

12.19.4.4 encoding

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

12.19.4.5 end_inclusive

```
end_inclusive
```

12.19.4.6 lines

```
lines
```

12.19.4.7 root

```
root
```

12.19.4.8 start_inclusive

```
start_inclusive
```

12.19.4.9 story_id

```
story_id
```

12.19.4.10 tei_path

```
tei_path
```

12.19.4.11 xml_namespace [1/2]

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

12.19.4.12 xml_namespace [2/2]

```
xml_namespace
```

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

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

12.20 Plot Class Reference

Static plotting helpers for visualization.

Static Public Member Functions

- None time_elapsed_by_names (str filename="./logs/charts/avg_runtime.png")
  Plot average elapsed time per function name, averaging across runs.

12.20.1 Detailed Description

Static plotting helpers for visualization.

12.20.2 Member Function Documentation

12.20.2.1 time_elapsed_by_names()

```
None time_elapsed_by_names (
             str  filename = "./logs/charts/avg_runtime.png" )  [static]
```

Plot average elapsed time per function name, averaging across runs.

Parameters

filename	Where to save the generated chart

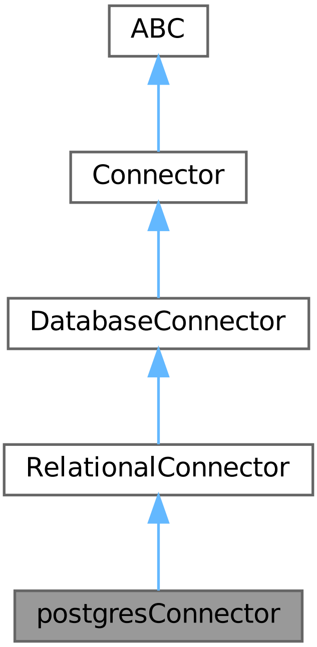
The documentation for this class was generated from the following file:

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

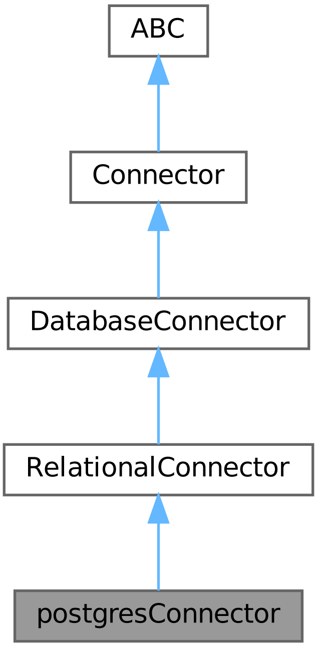
12.21 postgresConnector Class Reference

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_operations (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=True)

  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.
- 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`.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Tuple[Any, Any] result)

  Checks if the result of a SQL query is valid (i.e.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

12.21.1 Detailed Description

A relational database connector configured for PostgreSQL.

Note

Should be hidden from the user using a factory method.

12.21.2 Constructor & Destructor Documentation

12.21.2.1 __init__()

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

Configures the relational connector.

Parameters

verbose	Whether to print success and failure messages.

Reimplemented from RelationalConnector.

12.21.3 Member Data Documentation

12.21.3.1 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/src/connectors/relational.py

12.22 PRF1Metric Class Reference

Properties

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

12.22.1 Property Documentation

12.22.1.1 F1Score

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

12.22.1.2 Name

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

12.22.1.3 Precision

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

12.22.1.4 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

12.23 QAItem Class Reference

Properties

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

12.23.1 Property Documentation

12.23.1.1 Accuracy

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

12.23.1.2 GeneratedAnswer

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

12.23.1.3 GoldAnswer

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

12.23.1.4 IsCorrect

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

12.23.1.5 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

12.24 QAMetric Class Reference

Properties

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

12.24.1 Property Documentation

12.24.1.1 AverageAccuracy

```
double AverageAccuracy  [get]
```

12.24.1.2 QAItems

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

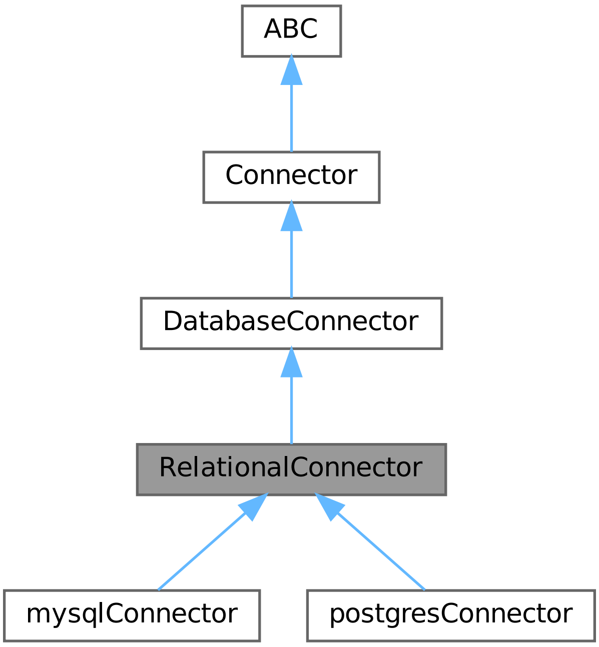
The documentation for this class was generated from the following file:

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

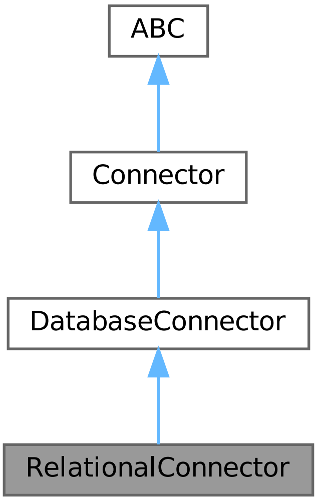
12.25 RelationalConnector Class Reference

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_operations (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=True)

  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.
- 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`.
- bool _returns_data (self, str query)

  Checks if a query is structured in a way that returns real data, and not status messages.
- bool _parsable_to_df (self, Tuple[Any, Any] result)

  Checks if the result of a SQL query is valid (i.e.

Protected Member Functions inherited from DatabaseConnector

- bool _is_single_query (self, str query)

  Checks if a string contains multiple queries.

12.25.1 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.

12.25.2 Constructor & Destructor Documentation

12.25.2.1 __init__()

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

Creates a new database connector.

Use src.connectors.relational.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.

12.25.3 Member Function Documentation

12.25.3.1 _parsable_to_df()

```
bool _parsable_to_df (
                self,
               Tuple[Any, Any] result )  [protected]
```

Checks if the result of a SQL query is valid (i.e.

can be converted to a Pandas DataFrame).

- SQLAlchemy CursorResult exposes .returns_rows and .keys().
- These must be fetched earlier, before the cursor is closed.

Parameters

result	The tuple result (rows, columns) of a SQL, Cypher, or JSON query.

Returns

Whether the object is parsable to DataFrame.

Reimplemented from DatabaseConnector.

12.25.3.2 _returns_data()

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

Checks if a query is structured in a way that returns real data, and not status messages.

Determines whether a SQL query should yield tabular data.

- Uses an exclusion list - commands that definitely return only status / row count.
- Everything else falls through to execution for validation.

Parameters

query	A single pre-validated SQL query string.

Returns

Whether the query is intended to fetch data (true) or might return a status message (false).

Reimplemented from DatabaseConnector.

12.25.3.3 _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.

12.25.3.4 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.

12.25.3.5 check_connection()

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

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.

12.25.3.6 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.

12.25.3.7 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.

12.25.3.8 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.

12.25.3.9 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.

12.25.3.10 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.

12.25.3.11 get_dataframe()

```
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

Exceptions

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

Reimplemented from DatabaseConnector.

12.25.3.12 test_operations()

```
bool test_operations (
              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.

12.25.4 Member Data Documentation

12.25.4.1 connection_string

```
connection_string
```

12.25.4.2 database_name

```
database_name
```

12.25.4.3 db_type

```
db_type
```

12.25.4.4 verbose

```
verbose
```

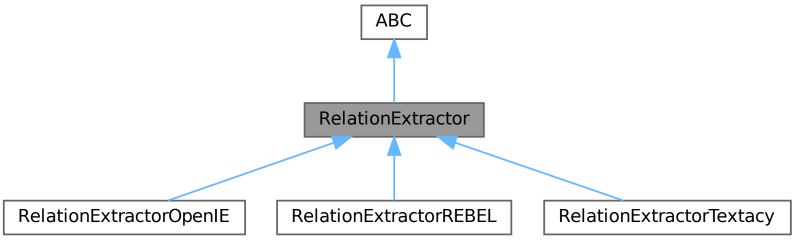
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/relational.py

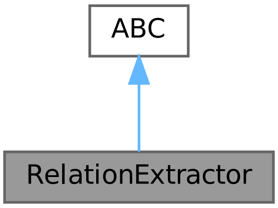
12.26 RelationExtractor Class Reference

Abstract base class for Relation Extraction (RE) models.

Inheritance diagram for RelationExtractor:



Collaboration diagram for RelationExtractor:



Public Member Functions

- List[Union[Tuple[str, str, str], str]] extract (self, str text, bool parse_tuples=True)

    Extract relations from the provided text.

Static Public Attributes

- str TUPLE_DELIM = " "

Protected Member Functions

- List[str] _format_triples_to_strings (self, List[Tuple[str, str, str]] triples)

  Helper to convert native tuples to standardized raw strings.

12.26.1 Detailed Description

Abstract base class for Relation Extraction (RE) models.

Derived classes must implement the extract method to process text and return a list of triples or raw strings. Backends (Spacy, Stanza, Transformers) should be lazy-loaded.

12.26.2 Member Function Documentation

12.26.2.1 _format_triples_to_strings()

```
List[str] _format_triples_to_strings (
             self,
            List[Tuple[str, str, str]] triples )  [protected]
```

Helper to convert native tuples to standardized raw strings.

12.26.2.2 extract()

```
List[Union[Tuple[str, str, str], str]] extract (
             self,
            str text,
            bool  parse_tuples = True )
```

Extract relations from the provided text.

Parameters

text	The raw input text to process.
parse_tuples	If False, returns a formatted string 'Subj Rel Obj'.

Returns

A list of triples (subj, rel, obj) or raw string outputs.

Reimplemented in RelationExtractorREBEL, RelationExtractorOpenIE, and RelationExtractorTextacy.

12.26.3 Member Data Documentation

12.26.3.1 TUPLE_DELIM

```
str TUPLE_DELIM = " "  [static]
```

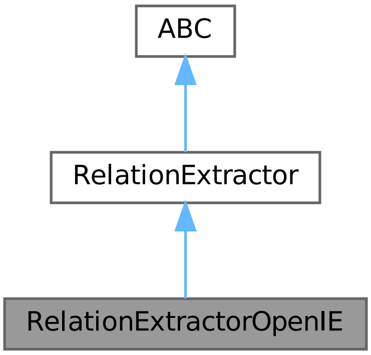
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/components/relation_extraction.py

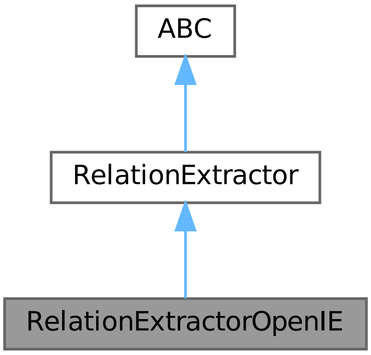
12.27 RelationExtractorOpenIE Class Reference

Wrapper for Stanford OpenIE using the Stanza library.

Inheritance diagram for RelationExtractorOpenIE:



Collaboration diagram for RelationExtractorOpenIE:



Public Member Functions

- None __init__ (self, memory='4G')

  Initialize the Stanza CoreNLP client interface.
- List[Union[Tuple[str, str, str], str]] extract (self, str text, bool parse_tuples=True)

  Extract triples using the Stanford OpenIE pipeline.

Public Attributes

- stanza
- CoreNLPClient
- memory

Protected Member Functions

- "CoreNLPClient" _get_client (self)

  Configure and instantiate the CoreNLP Client.

Protected Member Functions inherited from RelationExtractor

- List[str] _format_triples_to_strings (self, List[Tuple[str, str, str]] triples)

  Helper to convert native tuples to standardized raw strings.

Additional Inherited Members

Static Public Attributes inherited from RelationExtractor

- str TUPLE_DELIM = " "

12.27.1 Detailed Description

Wrapper for Stanford OpenIE using the Stanza library.

Note

> Requires Java (JDK 8+) and 'stanza' python package.

Ideal for "Exhaustive" and "Literal" extraction. Unlike generative models, this extracts spans directly from the text and handles coreference resolution internally.

12.27.2 Constructor & Destructor Documentation

12.27.2.1 __init__()

```
None __init__ (
              self,
              memory = '4G' )
```

Initialize the Stanza CoreNLP client interface.

Checks for the existence of the CoreNLP backend and installs it if missing. This is a blocking operation on the first run.

Parameters

memory	Java heap size string (e.g., '4G', '8G').

12.27.3 Member Function Documentation

12.27.3.1 _get_client()

```
"CoreNLPClient" _get_client (
              self )  [protected]
```

Configure and instantiate the CoreNLP Client.

Configuration targets "Exhaustive" and "Coref-Resolved" extraction:

- openie.resolve_coref: Uses the coref graph to replace pronouns (He -> Harry).
- openie.triple.strict: False allows for more loose/exhaustive extractions.
- openie.max_entailments_per_clause: Maximizes variations of triples returned.

  Returns

  An instance of stanza.server.CoreNLPClient.

12.27.3.2 extract()

```
List[Union[Tuple[str, str, str], str]] extract (
              self,
             str text,
             bool  parse_tuples = True )
```

Extract triples using the Stanford OpenIE pipeline.

Uses a context manager to spin up the Java server, process the text, and tear it down immediately to free resources. For production / batch processing, you might want to keep the client alive longer

Parameters

text	The raw narrative text.
parse_tuples	If False, concatenates the triples into a multi-line string.

Returns

A list of extracted relations.

Reimplemented from RelationExtractor.

12.27.4 Member Data Documentation

12.27.4.1 CoreNLPClient

```
CoreNLPClient
```

12.27.4.2 memory

```
memory
```

12.27.4.3 stanza

```
stanza
```

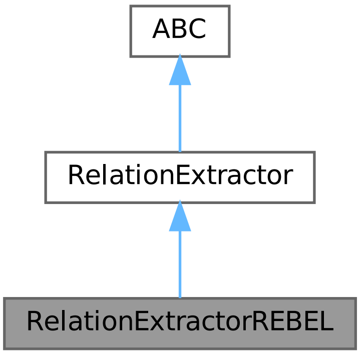
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/components/relation_extraction.py

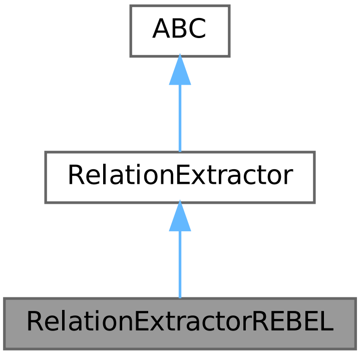
12.28 RelationExtractorREBEL Class Reference

Relation Extractor using the REBEL generative model (Seq2Seq).

Inheritance diagram for RelationExtractorREBEL:



Collaboration diagram for RelationExtractorREBEL:



Public Member Functions

- None __init__ (self, model_name="Babelscape/rebel-large", max_tokens=1024)

  Initialize the REBEL model and tokenizer.
- List[Union[Tuple[str, str, str], str]] extract (self, str text, bool parse_tuples=False)

  Perform extraction on the text using the generative model.

Public Attributes

- nlp
- sentencizer
- tokenizer
- model
- max_tokens
- tuple_delim

Additional Inherited Members

Static Public Attributes inherited from RelationExtractor

- str TUPLE_DELIM = " "

Protected Member Functions inherited from RelationExtractor

- List[str] _format_triples_to_strings (self, List[Tuple[str, str, str]] triples)

  Helper to convert native tuples to standardized raw strings.

12.28.1 Detailed Description

Relation Extractor using the REBEL generative model (Seq2Seq).

Note

> Requires 'torch', 'transformers', and 'spacy' installed.

REBEL treats RE as a translation task (Text -> Triples). It is powerful but can hallucinate or normalize entities (non-literal).

12.28.2 Constructor & Destructor Documentation

12.28.2.1 __init__()

```
None __init__ (
              self,
              model_name = "Babelscape/rebel-large",
              max_tokens = 1024 )
```

Initialize the REBEL model and tokenizer.

Note

> Lazy imports are used to prevent heavy libraries from loading unless this class is instantiated.

Parameters

model_name	The HuggingFace hub path for the model.
max_tokens	The maximum sequence length for the tokenizer.

12.28.3 Member Function Documentation

12.28.3.1 extract()

```
List[Union[Tuple[str, str, str], str]] extract (
              self,
             str text,
             bool  parse_tuples = False )
```

Perform extraction on the text using the generative model.

The text is first segmented into sentences because RE models degrade in performance on long, multi-sentence paragraphs.

Parameters

text	The input narrative text.
parse_tuples	If True, parses the generated string into structured tuples.

Returns

A list of extracted relations.

Reimplemented from RelationExtractor.

12.28.4 Member Data Documentation

12.28.4.1 max_tokens

```
max_tokens
```

12.28.4.2 model

```
model
```

12.28.4.3 nlp

```
nlp
```

12.28.4.4 sentencizer

```
sentencizer
```

12.28.4.5 tokenizer

```
tokenizer
```

12.28.4.6 tuple_delim

```
tuple_delim
```

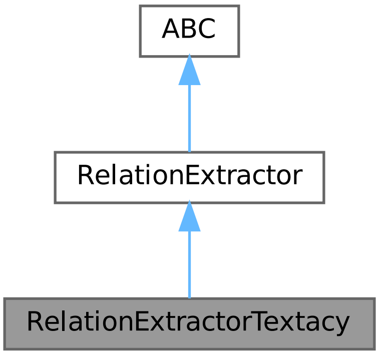
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/src/components/relation_extraction.py

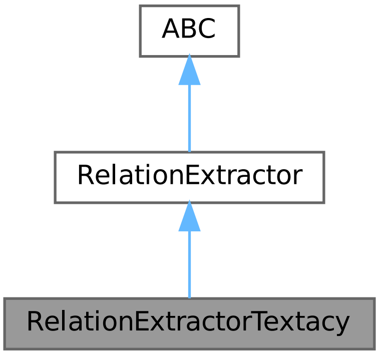
12.29 RelationExtractorTextacy Class Reference

Lightweight extraction using Spacy and Textacy (SVO).

Inheritance diagram for RelationExtractorTextacy:



Collaboration diagram for RelationExtractorTextacy:



Public Member Functions

- None __init__ (self)

  Initialize Spacy model for dependency parsing.
- List[Union[Tuple[str, str, str], str]] extract (self, str text, bool parse_tuples=True)

  Extract SVO triples.

Public Attributes

- nlp
- textacy

Additional Inherited Members

Static Public Attributes inherited from RelationExtractor

- str TUPLE_DELIM = " "

Protected Member Functions inherited from RelationExtractor

- List[str] _format_triples_to_strings (self, List[Tuple[str, str, str]] triples)

  Helper to convert native tuples to standardized raw strings.

12.29.1 Detailed Description

Lightweight extraction using Spacy and Textacy (SVO).

Note

> Requires 'spacy' and 'textacy'.

A pure-Python backup. Less exhaustive than OpenIE but faster and setup-free. Extracts Subject-Verb-Object patterns based on dependency parsing.

12.29.2 Constructor & Destructor Documentation

12.29.2.1 __init__()

```
None __init__ (
              self )
```

Initialize Spacy model for dependency parsing.

Note

> Defaults to 'en_core_web_sm'. Ensure this model is downloaded via `python -m spacy download en_core_web_sm`.

12.29.3 Member Function Documentation

12.29.3.1 extract()

```
List[Union[Tuple[str, str, str], str]] extract (
              self,
             str text,
             bool  parse_tuples = True )
```

Extract SVO triples.

Parameters

text	The raw input text.
parse_tuples	If False, concatenates the triples into a multi-line string.

Returns

A list of extracted relations.

Reimplemented from RelationExtractor.

12.29.4 Member Data Documentation

12.29.4.1 nlp

```
nlp
```

12.29.4.2 textacy

```
textacy
```

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

- /home/runner/work/dsci-capstone/dsci-capstone/src/components/relation_extraction.py

12.30 ScalarMetric Class Reference

Properties

- string Name `[get, set]`
- double Value `[get, set]`

12.30.1 Property Documentation

12.30.1.1 Name

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

12.30.1.2 Value

```
double Value [get], [set]
```

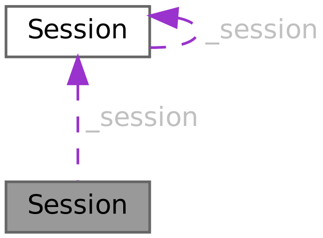
The documentation for this class was generated from the following file:

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

12.31 Session Class Reference

Stores active database connections and configuration settings.

Collaboration diagram for Session:



Public Member Functions

- Self __new__ (cls, ∗Any args, ∗∗Any kwargs)

  Creates a new session at first access, otherwise returns the existing session.
- None __init__ (self, bool verbose=False)

  Initializes the session using the .env file.

Public Attributes

- verbose

  Enables or disables the components from printing debug info.
- relational_db

  Stores RDF-compliant semantic triples.
- docs_db

  Stores input text, pre-processed chunks, JSON intermediates, and final output.
- graph_db

  Stores entities (nodes) and relations (edges).
- main_graph

  Main storage for initial pipeline.
- metrics

  The metrics class needs an instance to read the .env file.

Static Protected Attributes

- _instance = None
- bool _created = False
- Session _session = None

12.31.1 Detailed Description

Stores active database connections and configuration settings.

- This class implements Singleton design so only one session can be created.

12.31.2 Constructor & Destructor Documentation

12.31.2.1 __init__()

```
None __init__ (
              self,
             bool  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.

12.31.3 Member Function Documentation

12.31.3.1 __new__()

```
Self __new__ (
              cls,
             *Any args,
             **Any kwargs )
```

Creates a new session at first access, otherwise returns the existing session.

Parameters

**args*	Positional arguments forwarded to **init**().
***kwargs*	Keyword arguments forwarded to **init**().

Returns

The new global Session singleton.

12.31.4 Member Data Documentation

12.31.4.1 _created

```
bool _created = False  [static], [protected]
```

12.31.4.2 _instance

```
_instance = None  [static], [protected]
```

12.31.4.3 _session

```
Session _session = None  [static], [protected]
```

12.31.4.4 docs_db

```
docs_db
```

Stores input text, pre-processed chunks, JSON intermediates, and final output.

12.31.4.5 graph_db

```
graph_db
```

Stores entities (nodes) and relations (edges).

12.31.4.6 main_graph

```
main_graph
```

Main storage for initial pipeline.

12.31.4.7 metrics

```
metrics
```

The metrics class needs an instance to read the .env file.

12.31.4.8 relational_db

```
relational_db
```

Stores RDF-compliant semantic triples.

12.31.4.9 verbose

```
verbose
```

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/core/context.py

12.32 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)

12.32.1 Constructor & Destructor Documentation

12.32.1.1 __init__()

```
None __init__ (
                self,
               StoryStreamAdapter reader )
```

12.32.2 Member Function Documentation

12.32.2.1 _make_single()

```
Chunk _make_single (
                self,
               Chunk seg,
               str text,
               int max_len,
               Optional[Chunk]  start = None )  [protected]
```

12.32.2.2 _merge_chunks()

```
None _merge_chunks (
                self,
               List[Chunk] segs,
               int max_len )  [protected]
```

12.32.2.3 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.

12.32.2.4 stream_chunks()

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

12.32.3 Member Data Documentation

12.32.3.1 reader

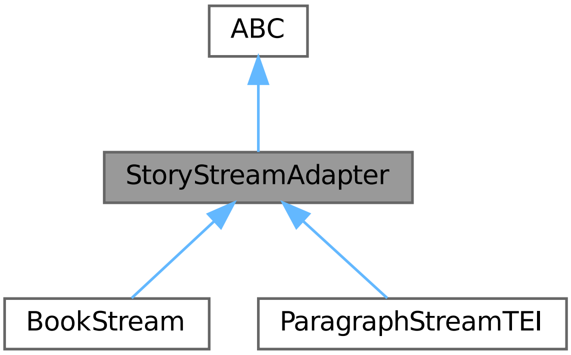
```
reader
```

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

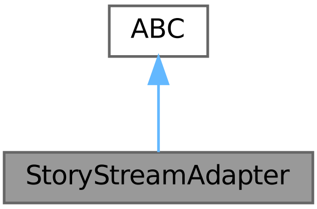
- /home/runner/work/dsci-capstone/dsci-capstone/src/components/book_conversion.py

12.33 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.

12.33.1 Member Function Documentation

12.33.1.1 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.

12.33.1.2 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.

12.33.1.3 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/src/components/book_conversion.py

12.34 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]

12.34.1 Property Documentation

12.34.1.1 BookID

```
string BookID [get], [set]
```

12.34.1.2 BookTitle

```
string BookTitle [get], [set]
```

12.34.1.3 GoldSummaryText

```
string GoldSummaryText [get], [set]
```

12.34.1.4 Metrics

```
SummaryMetrics Metrics = new() [get], [set]
```

12.34.1.5 QAResults

```
List<QAMetric> QAResults = new() [get], [set]
```

12.34.1.6 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

12.35 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]

12.35.1 Member Function Documentation

12.35.1.1 GetDefault()

```
static SummaryMetrics GetDefault ( )  [static]
```

12.35.2 Property Documentation

12.35.2.1 PRF1Metrics

```
List<PRF1Metric> PRF1Metrics = new()  [get], [set]
```

12.35.2.2 QA

```
QAMetric QA = new()  [get], [set]
```

12.35.2.3 ScalarMetrics

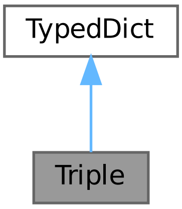
```
List<ScalarMetric> ScalarMetrics = new()  [get], [set]
```

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

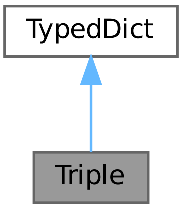
- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs

12.36 Triple Class Reference

Inheritance diagram for Triple:



Collaboration diagram for Triple:



Static Public Attributes

- str s
- str r
- str o

12.36.1 Member Data Documentation

12.36.1.1 o

```
str o  [static]
```

12.36.1.2 r

```
str r  [static]
```

12.36.1.3 s

```
str s  [static]
```

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

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

Chapter 13

File Documentation

13.1 /home/runner/work/dsci-capstone/dsci-capstone/conftest.py File Reference

Namespaces

- namespace conftest

Functions

- None pytest_addoption (Any parser)

  Command-line flags for pytest.
- pytest.param optional_param (str name, str package)

  Return a pytest.param that is skipped if the given package is missing.
- Generator[Session, None, None] session (pytest.FixtureRequest request)

  Fixture to create session.
- Generator[RelationalConnector, None, None] relational_db (Session session)

  Fixture to get relational database connection.
- Generator[DocumentConnector, None, None] docs_db (Session session)

  Fixture to get document database connection.
- Generator[GraphConnector, None, None] graph_db (Session session)

  Fixture to get document database connection.
- Generator[KnowledgeGraph, None, None] main_graph (pytest.FixtureRequest request, GraphConnector graph_db, Session session)

  Fixture to get document database connection.

13.2 /home/runner/work/dsci-capstone/dsci-capstone/src/charts.py File Reference

Classes

- class Plot

  Static plotting helpers for visualization.

Namespaces

- namespace src
- namespace src.charts

13.3 /home/runner/work/dsci-capstone/dsci-capstone/src/__init__.py File Reference

Namespaces

- namespace src

13.4 /home/runner/work/dsci-capstone/dsci-capstone/src/components/__init__.py File Reference

Namespaces

- namespace src
- namespace src.components

13.5 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/__init__.py File Reference

Namespaces

- namespace src
- namespace src.connectors

13.6 /home/runner/work/dsci-capstone/dsci-capstone/src/core/__init__.py File Reference

Namespaces

- namespace src
- namespace src.core

13.7 /home/runner/work/dsci-capstone/dsci-capstone/tests/__init__.py File Reference

Namespaces

- namespace tests

13.8 /home/runner/work/dsci-capstone/dsci-capstone/src/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 src
- namespace src.components
- namespace src.components.book_conversion

Variables

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

13.9 /home/runner/work/dsci-capstone/dsci-capstone/src/components/corpus.py File Reference

Namespaces

- namespace src
- namespace src.components
- namespace src.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()

13.10 /home/runner/work/dsci-capstone/dsci-capstone/src/components/fact_storage.py File Reference

Classes

- class Triple
- class KnowledgeGraph

  Manages a single graph within Neo4j.

Namespaces

- namespace src
- namespace src.components
- namespace src.components.fact_storage

Functions

- str sanitize_node (str label)

  Clean node name for Cypher safety.
- str sanitize_relation (str label, str mode="UPPER_CASE", str default_relation="RELATED_TO")

  Clean and normalize relation label for knowledge graphs.

Variables

- _nlp = None

13.11 /home/runner/work/dsci-capstone/dsci-capstone/src/components/metrics.py File Reference

Classes

- class Metrics

  Utility class for computing and posting evaluation metrics.

Namespaces

- namespace src
- namespace src.components
- namespace src.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.

13.12 /home/runner/work/dsci-capstone/dsci-capstone/src/components/README.md File Reference

13.13 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/←↩ README.md File Reference

13.14 /home/runner/work/dsci-capstone/dsci-capstone/src/core/←↩ README.md File Reference

13.15 /home/runner/work/dsci-capstone/dsci-capstone/src/README.md File Reference

13.16 /home/runner/work/dsci-capstone/dsci-capstone/tests/examples-db/README.md File Reference

13.17 /home/runner/work/dsci-capstone/dsci-capstone/tests/←↩ README.md File Reference

13.18 /home/runner/work/dsci-capstone/dsci-capstone/src/components/relation_extraction.py File Reference

Classes

- class RelationExtractor

  Abstract base class for Relation Extraction (RE) models.
- class RelationExtractorREBEL

  Relation Extractor using the REBEL generative model (Seq2Seq).
- class RelationExtractorOpenIE

  Wrapper for Stanford OpenIE using the Stanza library.
- class RelationExtractorTextacy

  Lightweight extraction using Spacy and Textacy (SVO).

Namespaces

- namespace src
- namespace src.components
- namespace src.components.relation_extraction

13.19 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/base.py File Reference

Classes

- class Connector

  Abstract base class for external connectors.
- class DatabaseConnector

  Abstract base class for database engine connectors.

Namespaces

- namespace src
- namespace src.connectors
- namespace src.connectors.base

13.20 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/document.py File Reference

Classes

- class DocumentConnector

  Connector for MongoDB (document database)

Namespaces

- namespace src
- namespace src.connectors
- namespace src.connectors.document

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]

13.21 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/graph.py File Reference

Classes

- class GraphConnector

  Connector for Neo4j (graph database).

Namespaces

- namespace src
- namespace src.connectors
- namespace src.connectors.graph

Functions

- DataFrame _filter_to_db (DataFrame df, str database_name)

  Filter a DataFrame by database context.
- DataFrame _tuples_to_df (List[Tuple[Any,...]] tuples, List[str] meta)

  Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.
- DataFrame _normalize_elements (DataFrame df)

  Convert Neo4j query results (nodes and relationships) into a Pandas DataFrame.

13.22 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/llm.py File Reference

Classes

- class LLMConnector

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

Namespaces

- namespace src
- namespace src.connectors
- namespace src.connectors.llm

Functions

- List[Dict[str, Any]] normalize_to_dict (Dict[str, Any]|List[Dict[str, Any]] data, List[str] keys)

  Normalize nested/compacted LLM output into flat dicts.

13.23 /home/runner/work/dsci-capstone/dsci-capstone/src/connectors/relational.py File Reference

Classes

- 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 src
- namespace src.connectors
- namespace src.connectors.relational

13.24 /home/runner/work/dsci-capstone/dsci-capstone/src/core/boss.py File Reference

Namespaces

- namespace src
- namespace src.core
- namespace src.core.boss

  Boss microservice for orchestrating distributed task processing.

Functions

- 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.
- None create_boss_thread (str DB_NAME, int BOSS_PORT, str COLLECTION)
- 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

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

13.25 /home/runner/work/dsci-capstone/dsci-capstone/src/core/context.py File Reference

Classes

- class Session

  Stores active database connections and configuration settings.

Namespaces

- namespace src
- namespace src.core
- namespace src.core.context

Functions

- Session get_session (∗Any args, ∗∗Any kwargs)

  Lazily creates a session on first call, otherwise returns the existing session.
- Any __getattr__ (str name)

  Lazy attribute resolution for module-level imports.

Variables

- Session session

  The global instance of the singleton Session class.

13.26 /home/runner/work/dsci-capstone/dsci-capstone/src/core/stages.py File Reference

Namespaces

- namespace src
- namespace src.core
- namespace src.core.stages

Functions

- str task_01_convert_epub (str epub_path, Optional[EPUBToTEI] converter=None)

    Will revisit later - Book classes need refactoring ###.
- task_02_parse_chapters (tei_path, book_chapters, book_id, story_id, start_str, end_str)
- task_03_chunk_story (story, max_chunk_length=1500)
- task_10_random_chunk (chunks)
- task_10_sample_chunks (chunks, n_sample)
- task_11_send_chunk (c, collection_name, book_title)
- task_12_relation_extraction_rebel (text, max_tokens=1024, parse_tuples=True)
- task_12_relation_extraction_openie (text, memory='4G', parse_tuples=True)
- task_12_relation_extraction_textacy (text, parse_tuples=True)
- task_13_concatenate_triples (extracted)
- task_14_relation_extraction_llm (triples_string, text)
- str task_15_sanitize_triples_llm (str llm_output)
- task_20_send_triples (triples)
- group_21_1_describe_graph (top_n=3)
- group_21_2_send_statistics ()
- group_21_3_post_statistics ()
- task_22_verbalize_triples (mode="triple")
- task_30_summarize_llm (triples_string)

    Prompt LLM to generate summary.
- task_31_send_summary (summary, collection_name, chunk_id)
- task_40_post_summary (book_id, book_title, summary)

    Send book info to Blazor.
- task_40_post_payload (book_id, book_title, summary, gold_summary, chunk, bookscore, questeval)

    Send metrics to Blazor.

13.27 /home/runner/work/dsci-capstone/dsci-capstone/src/core/worker.py File Reference

Namespaces

- namespace src
- namespace src.core
- namespace src.core.worker

    Generic Flask worker microservice for distributed task processing.

Functions

- None 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

13.28 /home/runner/work/dsci-capstone/dsci-capstone/src/main.py File Reference

Namespaces

- namespace src
- namespace src.main

Functions

- pipeline_A (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_B (collection_name, chunks, book_title)

  Extracts triples from a random chunk.
- pipeline_C (json_triples)

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

  Generate chunk summary.
- None pipeline_E (str summary, str book_title, str book_id, str chunk="", str gold_summary="", float bookscore=None, float questeval=None)

  Compute metrics and send available data to Blazor.
- full_pipeline (collection_name, epub_path, book_chapters, start_str, end_str, book_id, story_id, book_title)
- old_main (collection_name)

Variables

- DB_NAME = os.environ["DB_NAME"]
- BOSS_PORT = int(os.environ["PYTHON_PORT"])
- COLLECTION = os.environ["COLLECTION_NAME"]
- bool load_from_checkpoint = False
- str checkpoint_path = "./datasets/checkpoint.pkl"
- exist_ok
- int story_id = 1
- int book_id = 2
- str book_title = "The Phoenix and the Carpet"
- data = pickle.load(f_read)
- triples = data["triples"]
- chunk = data["chunk"]
- chunks
- chunk_id = chunk.get_chunk_id()
- triples_string = pipeline_C(triples)
- summary = pipeline_D(COLLECTION, triples_string, chunk.get_chunk_id())
- response = post_process_full_story(BOSS_PORT, story_id, task_type)

13.29 /home/runner/work/dsci-capstone/dsci-capstone/src/util.py File Reference

Classes

- class Log

  The Log class standardizes console output.
- class Log.Failure

  User-facing base class for custom error handling.
- class Log.BadAddressFailure

  Raised when a database connection string or address is invalid.

Namespaces

- namespace src
- namespace src.util

Functions

- 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.

13.30 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_db_↩ basic.py File Reference

Namespaces

- namespace tests
- namespace tests.test_db_basic

Functions

- 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.

13.31 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_db_↩ files.py File Reference

Namespaces

- namespace tests
- namespace tests.test_db_files

Functions

- 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_cypher_example_4 (GraphConnector graph_db)

  Test event graph with property mutations and multi-hop traversal.
- None _exec_query_file (DatabaseConnector db_fixture, str filename, List[str] valid_files)

  Run queries from a local file through the database.

13.32 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_kg_↩ triples.py File Reference

Namespaces

- namespace tests
- namespace tests.test_kg_triples

Functions

- None test_knowledge_graph_triples (KnowledgeGraph main_graph)

  Test KnowledgeGraph triple operations using add_triple and get_all_triples.
- Generator[KnowledgeGraph, None, None] nature_scene_graph (KnowledgeGraph main_graph)

  Create a scene graph with multiple location-based communities for testing.
- None test_get_subgraph_by_nodes (KnowledgeGraph nature_scene_graph)

  Test filtering triples by specific node IDs.
- None test_get_neighborhood (KnowledgeGraph nature_scene_graph)

  Test k-hop neighborhood expansion around a central node.
- None test_get_random_walk_sample (KnowledgeGraph nature_scene_graph)

  Test random walk sampling starting from specified nodes.
- None test_get_neighborhood_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for k-hop neighborhood expansion.
- None test_get_random_walk_sample_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for random walk sampling.
- None test_detect_community_clusters_minimal (KnowledgeGraph nature_scene_graph)

  Test basic community detection functionality.
- None test_detect_community_clusters_comprehensive (KnowledgeGraph nature_scene_graph)

  Comprehensive test for community detection with various parameters.
- None test_ranked_degree (KnowledgeGraph nature_scene_graph)

  Test filtering triples by ranked node degree.
- None test_ranked_degree_ties (KnowledgeGraph main_graph)

  Test that degree ranking correctly handles ties with minimal data.
- node_nlp_cases ()

  Fixtures focusing on spaCy NLP cleaning (Stopword/Part-of-speech removal).
- node_regex_cases ()

  Fixtures focusing on Regex replacement and stripping.
- relation_casing_cases ()

  Fixtures for testing UPPER_CASE vs camelCase modes.
- relation_fallback_cases ()

  Fixtures specifically testing the fallback logic (when input is invalid/numeric).
- test_sanitize_node_nlp_capabilities (node_nlp_cases)

  Test that NLP logic correctly strips POS tags (DET, PRON, PART).
- test_sanitize_node_regex_cleaning (node_regex_cases)

  Test that Regex logic handles symbols and whitespace correctly.
- test_sanitize_relation_modes (relation_casing_cases, mode)

  Test standard relation normalization for both supported modes.
- test_sanitize_relation_fallbacks (relation_fallback_cases)

  Test the 'safety net' fallback logic for relations.
- test_sanitize_relation_default_normalization ()

  Edge case: Ensure the default_relation itself is sanitized if used.

13.33 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_pipeline.py File Reference

Namespaces

- namespace tests
- namespace tests.test_pipeline

Functions

- book_data (request)

  Fixtures.
- book_1_data ()

  Example data for Book 1: Five Children and It.
- book_2_data ()

  Example data for Book 2: The Phoenix and the Carpet - realistic pipeline data.
- llm_data (request)

  Realistic or malformed LLM response edge cases.
- llm_edge_case_1 ()

  Save tokens by reusing the original subject.
- llm_edge_case_2 ()

  Save tokens by providing multiple subjects.
- llm_edge_case_3 ()

  Combine subject: List[str] and relation-object: List[Dict[str, str]].
- llm_edge_case_4 ()

  Same-length lists are also parsable.
- llm_edge_case_5 ()

  Mismatched-length lists: inferred as Cartesian Product.
- llm_edge_case_6 ()

  Matched-length lists: inferred as Columnar (Zip).
- test_job_01_convert_epub (book_data)

  Tests.
- test_job_02_parse_chapters (book_data)

  Test TEI -> Story parsing for multiple books.
- test_job_03_chunk_story (book_data)

  Test Story -> chunks splitting for multiple books.
- test_job_10_sample_chunks (book_data)

  Test sampling multiple chunks from a list.
- test_job_10_random_chunk (book_data)

  Test selecting a single random chunk.
- test_job_11_send_chunk (docs_db, book_data)

  Test inserting chunk into MongoDB collection.
- test_job_13_concatenate_triples (book_data)

  Test converting extracted triples to newline-delimited string.
- test_job_15_sanitize_triples_llm (book_data)

  Test parsing LLM output JSON into triples list.
- test_job_15_comprehensive (llm_data)

  Test parsing malformed LLM output.
- test_job_20_send_triples (main_graph, book_data)

  Test inserting triples into knowledge graph.

- test_job_21_describe_graph (main_graph, book_data)

  Test generating edge count summary of knowledge graph.
- test_job_22_verbalize_triples (main_graph, book_data)

  Test converting high-degree triples to string format.
- test_job_31_send_summary (docs_db, book_data)

  Test updating chunk with summary in MongoDB.
- test_pipeline_A_minimal (book_data)

  Minimal aggregate test.
- test_pipeline_A_from_csv ()

  Read example CSV and run pipeline_A for each row.
- test_pipeline_C_minimal (main_graph, book_data)

  Test running pipeline_C with smoke test data.
- test_pipeline_E_minimal_summary_only (book_data)

  Test running pipeline_E with summary-only mode.
- test_pipeline_E_minimal_full_payload (book_data)

  Test running pipeline_E with full payload including metrics.

13.34 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/_Imports.razor File Reference

13.35 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/App.razor File Reference

13.36 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/MainLayout.razor File Reference

13.37 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/NavMenu.razor File Reference

13.38 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Error.razor File Reference

13.39 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Graph.razor File Reference

13.40 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Home.razor File Reference

13.41 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Metrics.razor File Reference

13.42 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Routes.razor File Reference

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

Classes

- class MetricsController

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Controllers

13.44 /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

13.45 /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

13.46 /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

13.47 /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

13.48 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs File Reference

Classes

- class ScalarMetric

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

13.49 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryData.cs File Reference

Classes

- class SummaryData

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

13.50 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs File Reference

Classes

- class SummaryMetrics

Namespaces

- namespace BlazorApp
- namespace BlazorApp.Models

Index