

Data Science Capstone Project

Generated by Doxygen 1.9.8

1 Namespace Index	1
1.1 Package List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 BlazorApp Namespace Reference	9
5.2 BlazorApp.Controllers Namespace Reference	9
5.3 BlazorApp.Hubs Namespace Reference	9
5.4 BlazorApp.Models Namespace Reference	9
5.5 components Namespace Reference	10
5.6 components.book_conversion Namespace Reference	10
5.6.1 Variable Documentation	10
5.6.1.1 nlp	10
5.6.1.2 sentencizer	10
5.7 components.connectors Namespace Reference	11
5.8 components.corpus Namespace Reference	11
5.8.1 Function Documentation	11
5.8.1.1 fuzzy_merge_titles()	11
5.8.1.2 load_booksum()	12
5.8.1.3 load_narrativeqa()	12
5.8.1.4 merge_dataframes()	12
5.8.1.5 normalize_title()	12
5.8.1.6 to_df_booksum()	12
5.8.1.7 to_df_nqa()	13
5.8.2 Variable Documentation	13
5.8.2.1 df	13
5.8.2.2 df_booksum	13
5.8.2.3 df_nqa	13
5.8.2.4 index	13
5.8.2.5 m	13
5.9 components.document_storage Namespace Reference	13
5.9.1 Function Documentation	14
5.9.1.1 _docs_to_df()	14
5.9.1.2 _find_compatible_nested_key()	14
5.9.1.3 _flatten_recursive()	15
5.9.1.4 _sanitize_document()	15

5.9.1.5 <code>_sanitize_json()</code>	16
5.9.1.6 <code>mongo_handle()</code>	16
5.9.2 Variable Documentation	16
5.9.2.1 <code>MongoHandle</code>	16
5.10 <code>components.fact_storage</code> Namespace Reference	17
5.10.1 Function Documentation	17
5.10.1.1 <code>_filter_to_db()</code>	17
5.10.1.2 <code>_normalize_elements()</code>	17
5.10.1.3 <code>_tuples_to_df()</code>	18
5.11 <code>components.metrics</code> Namespace Reference	18
5.11.1 Function Documentation	19
5.11.1.1 <code>chunk_bookscore()</code>	19
5.11.1.2 <code>run_bookscore()</code>	19
5.11.1.3 <code>run_questeval()</code>	21
5.12 <code>components.semantic_web</code> Namespace Reference	22
5.13 <code>components.text_processing</code> Namespace Reference	22
5.13.1 Variable Documentation	22
5.13.1.1 <code>nlp</code>	22
5.13.1.2 <code>sentencizer</code>	22
5.14 <code>src</code> Namespace Reference	23
5.15 <code>src.flask</code> Namespace Reference	23
5.15.1 Detailed Description	24
5.15.2 Function Documentation	24
5.15.2.1 <code>create_app()</code>	24
5.15.2.2 <code>get_task_info()</code>	24
5.15.2.3 <code>load_boss_config()</code>	25
5.15.2.4 <code>load_imports()</code>	25
5.15.2.5 <code>load_mongo_config()</code>	25
5.15.2.6 <code>mark_task_in_progress()</code>	26
5.15.2.7 <code>notify_boss()</code>	26
5.15.2.8 <code>process_task()</code>	26
5.15.2.9 <code>save_task_result()</code>	27
5.15.3 Variable Documentation	27
5.15.3.1 <code>app</code>	27
5.15.3.2 <code>args</code>	27
5.15.3.3 <code>boss_url</code>	28
5.15.3.4 <code>daemon</code>	28
5.15.3.5 <code>help</code>	28
5.15.3.6 <code>host</code>	28
5.15.3.7 <code>MongoHandle</code>	28
5.15.3.8 <code>parser</code>	28
5.15.3.9 <code>PORT</code>	28

5.15.3.10 port	28
5.15.3.11 required	28
5.15.3.12 target	28
5.15.3.13 task_queue	29
5.15.3.14 task_worker	29
5.15.3.15 True	29
5.15.3.16 use_reloader	29
5.16 src.main Namespace Reference	29
5.16.1 Function Documentation	31
5.16.1.1 assign_task_to_worker()	31
5.16.1.2 chunk_single()	31
5.16.1.3 clear_task_data()	32
5.16.1.4 convert_from_csv()	32
5.16.1.5 convert_single()	32
5.16.1.6 create_app()	32
5.16.1.7 full_pipeline()	33
5.16.1.8 graph_triple_files()	33
5.16.1.9 load_worker_config()	33
5.16.1.10 old_main()	33
5.16.1.11 output_single()	33
5.16.1.12 pipeline_1()	34
5.16.1.13 pipeline_2()	34
5.16.1.14 pipeline_3()	34
5.16.1.15 pipeline_4()	34
5.16.1.16 pipeline_5a()	35
5.16.1.17 pipeline_5b()	35
5.16.1.18 post_chunk_status()	35
5.16.1.19 post_process_full_story()	36
5.16.1.20 post_story_status()	36
5.16.1.21 process_single()	37
5.16.1.22 test_relation_extraction()	37
5.16.2 Variable Documentation	37
5.16.2.1 app	37
5.16.2.2 book_id	37
5.16.2.3 book_title	37
5.16.2.4 BOSS_PORT	37
5.16.2.5 chapters	37
5.16.2.6 chunk	38
5.16.2.7 chunk_id	38
5.16.2.8 chunks	38
5.16.2.9 COLLECTION	38
5.16.2.10 collection	38

5.16.2.11 daemon	38
5.16.2.12 DB_NAME	38
5.16.2.13 end	38
5.16.2.14 mongo_db	38
5.16.2.15 MongoHandle	39
5.16.2.16 response	39
5.16.2.17 response_files	39
5.16.2.18 run_app	39
5.16.2.19 session	39
5.16.2.20 start	39
5.16.2.21 story_id	39
5.16.2.22 summary	39
5.16.2.23 target	39
5.16.2.24 task_types	39
5.16.2.25 tei	40
5.16.2.26 triple_files	40
5.16.2.27 triples	40
5.16.2.28 triples_string	40
5.16.2.29 worker_urls	40
5.17 src.setup Namespace Reference	40
5.17.1 Variable Documentation	40
5.17.1.1 session	40
5.18 src.util Namespace Reference	41
5.18.1 Function Documentation	41
5.18.1.1 all_none()	41
5.18.1.2 check_values()	41
5.18.1.3 df_natural_sorted()	42
5.19 tests Namespace Reference	42
5.20 tests.conftest Namespace Reference	42
5.20.1 Function Documentation	42
5.20.1.1 pytest_addoption()	42
5.20.1.2 session()	43
5.21 tests.test_components Namespace Reference	43
5.21.1 Function Documentation	44
5.21.1.1 _test_query_file()	44
5.21.1.2 docs_db()	44
5.21.1.3 graph_db()	44
5.21.1.4 load_examples_relational()	45
5.21.1.5 nature_scene_graph()	45
5.21.1.6 relational_db()	45
5.21.1.7 test_cypher_example_1()	45
5.21.1.8 test_cypher_example_2()	45

5.21.1.9 test_cypher_example_3()	46
5.21.1.10 test_cypher_example_4()	46
5.21.1.11 test_db_docs_comprehensive()	46
5.21.1.12 test_db_docs_minimal()	46
5.21.1.13 test_db_graph_comprehensive()	46
5.21.1.14 test_db_graph_minimal()	46
5.21.1.15 test_db_relational_comprehensive()	47
5.21.1.16 test_db_relational_minimal()	47
5.21.1.17 test_detect_community_clusters_comprehensive()	47
5.21.1.18 test_detect_community_clusters_minimal()	47
5.21.1.19 test_get_neighborhood()	47
5.21.1.20 test_get_neighborhood_comprehensive()	48
5.21.1.21 test_get_random_walk_sample()	48
5.21.1.22 test_get_random_walk_sample_comprehensive()	48
5.21.1.23 test_get_subgraph_by_nodes()	48
5.21.1.24 test_knowledge_graph_triples()	49
5.21.1.25 test_mongo_example_1()	49
5.21.1.26 test_mongo_example_2()	49
5.21.1.27 test_mongo_example_3()	49
5.21.1.28 test_sql_example_1()	50
5.21.1.29 test_sql_example_2()	50
6 Class Documentation	51
6.1 Book Class Reference	51
6.1.1 Constructor & Destructor Documentation	51
6.1.1.1 __init__()	51
6.1.2 Member Function Documentation	51
6.1.2.1 stream_chapters()	51
6.2 BookFactory Class Reference	52
6.2.1 Member Function Documentation	52
6.2.1.1 create_book()	52
6.3 BookStream Class Reference	53
6.3.1 Constructor & Destructor Documentation	54
6.3.1.1 __init__()	54
6.3.2 Member Function Documentation	54
6.3.2.1 stream_segments()	54
6.3.3 Member Data Documentation	54
6.3.3.1 book	54
6.4 Chunk Class Reference	54
6.4.1 Detailed Description	55
6.4.2 Constructor & Destructor Documentation	55
6.4.2.1 __init__()	55

6.4.3 Member Function Documentation	56
6.4.3.1 __repr__()	56
6.4.3.2 char_count()	56
6.4.3.3 get_chunk_id()	56
6.4.3.4 to_mongo_dict()	57
6.4.4 Member Data Documentation	57
6.4.4.1 book_id	57
6.4.4.2 chapter_number	57
6.4.4.3 chapter_percent	57
6.4.4.4 line_end	57
6.4.4.5 line_start	57
6.4.4.6 story_id	57
6.4.4.7 story_percent	57
6.4.4.8 text	58
6.5 Connector Class Reference	58
6.5.1 Detailed Description	59
6.5.2 Member Function Documentation	59
6.5.2.1 check_connection()	59
6.5.2.2 configure()	60
6.5.2.3 execute_file()	60
6.5.2.4 execute_query()	60
6.5.2.5 test_connection()	61
6.6 DatabaseConnector Class Reference	61
6.6.1 Detailed Description	64
6.6.2 Constructor & Destructor Documentation	64
6.6.2.1 __init__()	64
6.6.3 Member Function Documentation	64
6.6.3.1 _is_single_query()	64
6.6.3.2 _parsable_to_df()	65
6.6.3.3 _returns_data()	65
6.6.3.4 _split_combined()	66
6.6.3.5 change_database()	66
6.6.3.6 configure()	66
6.6.3.7 create_database()	67
6.6.3.8 database_exists()	67
6.6.3.9 drop_database()	67
6.6.3.10 execute_combined()	68
6.6.3.11 execute_file()	68
6.6.3.12 execute_query()	69
6.6.3.13 get_dataframe()	69
6.6.3.14 temp_database()	70
6.6.4 Member Data Documentation	70

6.6.4.1 connection_string	70
6.6.4.2 db_engine	70
6.6.4.3 db_type	70
6.6.4.4 host	70
6.6.4.5 password	70
6.6.4.6 port	70
6.6.4.7 username	71
6.6.4.8 verbose	71
6.7 DocumentConnector Class Reference	71
6.7.1 Detailed Description	74
6.7.2 Constructor & Destructor Documentation	74
6.7.2.1 __init__()	74
6.7.3 Member Function Documentation	74
6.7.3.1 _parsable_to_df()	74
6.7.3.2 _returns_data()	75
6.7.3.3 _split_combined()	75
6.7.3.4 change_database()	76
6.7.3.5 check_connection()	76
6.7.3.6 create_database()	76
6.7.3.7 database_exists()	77
6.7.3.8 delete_dummy()	77
6.7.3.9 drop_database()	77
6.7.3.10 execute_query()	78
6.7.3.11 get_dataframe()	78
6.7.3.12 get_unmanaged_handle()	79
6.7.3.13 test_connection()	79
6.7.4 Member Data Documentation	80
6.7.4.1 _auth_suffix	80
6.7.4.2 connection_string	80
6.7.4.3 database_name	80
6.7.4.4 verbose	80
6.8 EPUBToTEI Class Reference	80
6.8.1 Detailed Description	81
6.8.2 Constructor & Destructor Documentation	81
6.8.2.1 __init__()	81
6.8.3 Member Function Documentation	81
6.8.3.1 _prune_bad_tags()	81
6.8.3.2 _sanitize_ids()	82
6.8.3.3 clean_tei()	82
6.8.3.4 convert_to_tei()	82
6.8.4 Member Data Documentation	82
6.8.4.1 clean_tei_content	82

6.8.4.2 encoding	82
6.8.4.3 epub_path	82
6.8.4.4 pandoc_xml_path	83
6.8.4.5 raw_tei_content	83
6.8.4.6 tei_path	83
6.8.4.7 xml_namespace	83
6.9 Log.Failure Class Reference	83
6.9.1 Constructor & Destructor Documentation	84
6.9.1.1 __init__()	84
6.9.2 Member Function Documentation	84
6.9.2.1 __str__()	84
6.9.3 Member Data Documentation	84
6.9.3.1 msg	84
6.9.3.2 prefix	84
6.10 GraphConnector Class Reference	85
6.10.1 Detailed Description	87
6.10.2 Constructor & Destructor Documentation	87
6.10.2.1 __init__()	87
6.10.3 Member Function Documentation	88
6.10.3.1 _execute_retag_db()	88
6.10.3.2 _fetch_latest()	88
6.10.3.3 _parsable_to_df()	88
6.10.3.4 _returns_data()	89
6.10.3.5 _split_combined()	89
6.10.3.6 change_database()	90
6.10.3.7 check_connection()	90
6.10.3.8 create_database()	90
6.10.3.9 database_exists()	91
6.10.3.10 delete_dummy()	91
6.10.3.11 drop_database()	91
6.10.3.12 drop_graph()	92
6.10.3.13 execute_query()	92
6.10.3.14 get_dataframe()	93
6.10.3.15 get_unique()	93
6.10.3.16 IS_DUMMY_()	94
6.10.3.17 NOT_DUMMY_()	94
6.10.3.18 SAME_DB_KG_()	94
6.10.3.19 temp_graph()	95
6.10.3.20 test_connection()	95
6.10.4 Member Data Documentation	95
6.10.4.1 _graph_name	95
6.10.4.2 connection_string	95

6.10.4.3 database_name	96
6.10.4.4 verbose	96
6.11 KnowledgeGraph Class Reference	96
6.11.1 Detailed Description	97
6.11.2 Constructor & Destructor Documentation	97
6.11.2.1 __init__()	97
6.11.3 Member Function Documentation	97
6.11.3.1 add_triple()	97
6.11.3.2 detect_community_clusters()	98
6.11.3.3 find_element_names()	99
6.11.3.4 get_all_triples()	99
6.11.3.5 get_community_subgraph()	100
6.11.3.6 get_edge_counts()	100
6.11.3.7 get_neighborhood()	101
6.11.3.8 get_node_context()	101
6.11.3.9 get_random_walk_sample()	102
6.11.3.10 get_relation_summary()	102
6.11.3.11 get_subgraph_by_nodes()	103
6.11.3.12 get_summary_stats()	103
6.11.3.13 get_triple_properties()	103
6.11.3.14 print_nodes()	104
6.11.3.15 print_triples()	104
6.11.3.16 to_contextualized_string()	104
6.11.3.17 to_narrative()	105
6.11.3.18 to_triple_string()	105
6.11.3.19 triples_to_names()	106
6.11.4 Member Data Documentation	106
6.11.4.1 database	106
6.11.4.2 graph_name	107
6.11.4.3 verbose	107
6.12 LLMConnector Class Reference	107
6.12.1 Detailed Description	109
6.12.2 Constructor & Destructor Documentation	109
6.12.2.1 __init__()	109
6.12.3 Member Function Documentation	109
6.12.3.1 check_connection()	109
6.12.3.2 configure()	110
6.12.3.3 execute_file()	110
6.12.3.4 execute_full_query()	110
6.12.3.5 execute_query()	110
6.12.3.6 normalize_triples()	111
6.12.3.7 test_connection()	111

6.12.4 Member Data Documentation	112
6.12.4.1 llm	112
6.12.4.2 model_name	112
6.12.4.3 system_prompt	112
6.12.4.4 temperature	112
6.13 Log Class Reference	112
6.13.1 Detailed Description	114
6.13.2 Member Function Documentation	114
6.13.2.1 fail()	114
6.13.2.2 fail_legacy()	115
6.13.2.3 success()	115
6.13.2.4 success_legacy()	115
6.13.2.5 warn()	116
6.13.3 Member Data Documentation	116
6.13.3.1 bad_addr	116
6.13.3.2 bad_path	116
6.13.3.3 bad_val	116
6.13.3.4 BRIGHT	116
6.13.3.5 conn_abc	116
6.13.3.6 create_db	117
6.13.3.7 db_conn_abc	117
6.13.3.8 db_exists	117
6.13.3.9 doc_db	117
6.13.3.10 drop_db	117
6.13.3.11 drop_gr	117
6.13.3.12 FAILURE_COLOR	117
6.13.3.13 FULL_DF	117
6.13.3.14 get_df	117
6.13.3.15 get_unique	118
6.13.3.16 good_val	118
6.13.3.17 gr_db	118
6.13.3.18 gr_rag	118
6.13.3.19 GREEN	118
6.13.3.20 kg	118
6.13.3.21 msg_bad_addr	118
6.13.3.22 msg_bad_coll	118
6.13.3.23 msg_bad_exec_f	118
6.13.3.24 msg_bad_exec_q	118
6.13.3.25 msg_bad_graph	119
6.13.3.26 msg_bad_path	119
6.13.3.27 msg_bad_table	119
6.13.3.28 msg_bad_triples	119

6.13.3.29 MSG_COLOR	119
6.13.3.30 msg_compare	119
6.13.3.31 msg_db_connect	119
6.13.3.32 msg_db_current	119
6.13.3.33 msg_db_exists	119
6.13.3.34 msg_db_not_found	120
6.13.3.35 msg_fail_manage_db	120
6.13.3.36 msg_fail_manage_gr	120
6.13.3.37 msg_fail_parse	120
6.13.3.38 msg_good_coll	120
6.13.3.39 msg_good_exec_f	120
6.13.3.40 msg_good_exec_q	120
6.13.3.41 msg_good_exec_qr	120
6.13.3.42 msg_good_graph	121
6.13.3.43 msg_good_path	121
6.13.3.44 msg_good_table	121
6.13.3.45 msg_multiple_query	121
6.13.3.46 msg_result	121
6.13.3.47 msg_success_managed_db	121
6.13.3.48 msg_success_managed_gr	121
6.13.3.49 msg_swap_db	122
6.13.3.50 msg_swap_kg	122
6.13.3.51 msg_unknown_error	122
6.13.3.52 pytest_db	122
6.13.3.53 RED	122
6.13.3.54 rel_db	122
6.13.3.55 run_f	122
6.13.3.56 run_q	122
6.13.3.57 SUCCESS_COLOR	122
6.13.3.58 swap_db	123
6.13.3.59 swap_kg	123
6.13.3.60 test_basic	123
6.13.3.61 test_conn	123
6.13.3.62 test_df	123
6.13.3.63 test_info	123
6.13.3.64 test_tmp_db	123
6.13.3.65 USE_COLORS	123
6.13.3.66 WARNING_COLOR	123
6.13.3.67 WHITE	124
6.13.3.68 YELLOW	124
6.14 Metrics Class Reference	124
6.14.1 Detailed Description	125

6.14.2 Constructor & Destructor Documentation	125
6.14.2.1 <code>__init__()</code>	125
6.14.3 Member Function Documentation	125
6.14.3.1 <code>compute_basic_metrics()</code>	125
6.14.3.2 <code>create_summary_payload()</code>	125
6.14.3.3 <code>generate_default_metrics()</code>	126
6.14.3.4 <code>generate_example_metrics()</code>	127
6.14.3.5 <code>post_basic_metrics()</code>	127
6.14.3.6 <code>post_basic_output()</code>	128
6.14.3.7 <code>post_payload()</code>	128
6.14.4 Member Data Documentation	128
6.14.4.1 <code>HOST</code>	128
6.14.4.2 <code>PORT</code>	128
6.14.4.3 <code>timeout_seconds</code>	128
6.14.4.4 <code>url</code>	129
6.15 MetricsController Class Reference	129
6.15.1 Constructor & Destructor Documentation	130
6.15.1.1 <code>MetricsController()</code>	130
6.15.2 Member Function Documentation	130
6.15.2.1 <code>GetAll()</code>	130
6.15.2.2 <code>GetIndex()</code>	130
6.15.2.3 <code>Post()</code>	130
6.15.3 Member Data Documentation	130
6.15.3.1 <code>_hubContext</code>	130
6.15.3.2 <code>_logger</code>	130
6.15.3.3 <code>Summaries</code>	131
6.16 MetricsHub Class Reference	131
6.16.1 Constructor & Destructor Documentation	132
6.16.1.1 <code>MetricsHub()</code>	132
6.16.2 Member Function Documentation	132
6.16.2.1 <code>OnConnectedAsync()</code>	132
6.16.2.2 <code>OnDisconnectedAsync()</code>	132
6.16.3 Member Data Documentation	132
6.16.3.1 <code>_logger</code>	132
6.17 mysqlConnector Class Reference	132
6.17.1 Detailed Description	136
6.17.2 Constructor & Destructor Documentation	136
6.17.2.1 <code>__init__()</code>	136
6.17.3 Member Data Documentation	136
6.17.3.1 <code>specific_queries</code>	136
6.18 ParagraphStreamTEI Class Reference	136
6.18.1 Detailed Description	138

6.18.2 Constructor & Destructor Documentation	138
6.18.2.1 <code>__init__()</code>	138
6.18.3 Member Function Documentation	139
6.18.3.1 <code>pre_compute_segments()</code>	139
6.18.3.2 <code>stream_segments()</code>	139
6.18.4 Member Data Documentation	139
6.18.4.1 <code>allowed_chapters</code>	139
6.18.4.2 <code>book_id</code>	139
6.18.4.3 <code>chunks</code>	139
6.18.4.4 <code>encoding</code>	140
6.18.4.5 <code>end_inclusive</code>	140
6.18.4.6 <code>lines</code>	140
6.18.4.7 <code>root</code>	140
6.18.4.8 <code>start_inclusive</code>	140
6.18.4.9 <code>story_id</code>	140
6.18.4.10 <code>tei_path</code>	140
6.18.4.11 <code>xml_namespace</code> [1/2]	140
6.18.4.12 <code>xml_namespace</code> [2/2]	140
6.19 postgresConnector Class Reference	141
6.19.1 Detailed Description	144
6.19.2 Constructor & Destructor Documentation	144
6.19.2.1 <code>__init__()</code>	144
6.19.3 Member Data Documentation	144
6.19.3.1 <code>specific_queries</code>	144
6.20 PRF1Metric Class Reference	144
6.20.1 Property Documentation	145
6.20.1.1 <code>F1Score</code>	145
6.20.1.2 <code>Name</code>	145
6.20.1.3 <code>Precision</code>	145
6.20.1.4 <code>Recall</code>	145
6.21 QAItem Class Reference	145
6.21.1 Property Documentation	145
6.21.1.1 <code>Accuracy</code>	145
6.21.1.2 <code>GeneratedAnswer</code>	145
6.21.1.3 <code>GoldAnswer</code>	146
6.21.1.4 <code>IsCorrect</code>	146
6.21.1.5 <code>Question</code>	146
6.22 QAMetric Class Reference	146
6.22.1 Property Documentation	146
6.22.1.1 <code>AverageAccuracy</code>	146
6.22.1.2 <code>QAItems</code>	146
6.23 RelationalConnector Class Reference	147

6.23.1 Detailed Description	149
6.23.2 Constructor & Destructor Documentation	150
6.23.2.1 <code>__init__()</code>	150
6.23.3 Member Function Documentation	150
6.23.3.1 <code>_parsable_to_df()</code>	150
6.23.3.2 <code>_returns_data()</code>	150
6.23.3.3 <code>_split_combined()</code>	151
6.23.3.4 <code>change_database()</code>	151
6.23.3.5 <code>check_connection()</code>	152
6.23.3.6 <code>create_database()</code>	152
6.23.3.7 <code>database_exists()</code>	153
6.23.3.8 <code>drop_database()</code>	153
6.23.3.9 <code>execute_query()</code>	153
6.23.3.10 <code>from_env()</code>	154
6.23.3.11 <code>get_dataframe()</code>	154
6.23.3.12 <code>test_connection()</code>	155
6.23.4 Member Data Documentation	155
6.23.4.1 <code>connection_string</code>	155
6.23.4.2 <code>database_name</code>	155
6.23.4.3 <code>db_type</code>	155
6.23.4.4 <code>verbose</code>	156
6.24 RelationExtractor Class Reference	156
6.24.1 Constructor & Destructor Documentation	156
6.24.1.1 <code>__init__()</code>	156
6.24.2 Member Function Documentation	156
6.24.2.1 <code>extract()</code>	156
6.24.3 Member Data Documentation	156
6.24.3.1 <code>max_tokens</code>	156
6.24.3.2 <code>model</code>	157
6.24.3.3 <code>tokenizer</code>	157
6.24.3.4 <code>tuple_delim</code>	157
6.25 ScalarMetric Class Reference	157
6.25.1 Property Documentation	157
6.25.1.1 <code>Name</code>	157
6.25.1.2 <code>Value</code>	157
6.26 Session Class Reference	157
6.26.1 Detailed Description	158
6.26.2 Constructor & Destructor Documentation	158
6.26.2.1 <code>__init__()</code>	158
6.26.3 Member Function Documentation	159
6.26.3.1 <code>__new__()</code>	159
6.26.3.2 <code>reset()</code>	159

6.26.3.3 test_database_connections()	159
6.26.4 Member Data Documentation	159
6.26.4.1 _instance	159
6.26.4.2 docs_db	159
6.26.4.3 graph_db	159
6.26.4.4 main_graph	160
6.26.4.5 relational_db	160
6.26.4.6 verbose	160
6.27 Story Class Reference	160
6.27.1 Constructor & Destructor Documentation	161
6.27.1.1 __init__()	161
6.27.2 Member Function Documentation	161
6.27.2.1 _make_single()	161
6.27.2.2 _merge_chunks()	161
6.27.2.3 pre_split_chunks()	161
6.27.2.4 stream_chunks()	161
6.27.3 Member Data Documentation	162
6.27.3.1 reader	162
6.28 StoryStreamAdapter Class Reference	162
6.28.1 Member Function Documentation	163
6.28.1.1 stream_paragraphs()	163
6.28.1.2 stream_segments()	163
6.28.1.3 stream_sentences()	163
6.29 SummaryData Class Reference	164
6.29.1 Property Documentation	164
6.29.1.1 BookID	164
6.29.1.2 BookTitle	164
6.29.1.3 GoldSummaryText	164
6.29.1.4 Metrics	164
6.29.1.5 QAResults	164
6.29.1.6 SummaryText	164
6.30 SummaryMetrics Class Reference	165
6.30.1 Member Function Documentation	165
6.30.1.1 GetDefault()	165
6.30.2 Property Documentation	165
6.30.2.1 PRF1Metrics	165
6.30.2.2 QA	165
6.30.2.3 ScalarMetrics	165
7 File Documentation	167
7.1 /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py File Reference	167
7.2 /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py File Reference	168

7.3 /home/runner/work/dsci-capstone/dsci-capstone/components/corpus.py File Reference	168
7.4 /home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py File Reference	169
7.5 /home/runner/work/dsci-capstone/dsci-capstone/components/fact_storage.py File Reference	169
7.6 /home/runner/work/dsci-capstone/dsci-capstone/components/metrics.py File Reference	170
7.7 /home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py File Reference	170
7.8 /home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py File Reference	171
7.9 /home/runner/work/dsci-capstone/dsci-capstone/components/__init__.py File Reference	171
7.10 /home/runner/work/dsci-capstone/dsci-capstone/src/__init__.py File Reference	171
7.11 /home/runner/work/dsci-capstone/dsci-capstone/tests/__init__.py File Reference	171
7.12 /home/runner/work/dsci-capstone/dsci-capstone/src/flask.py File Reference	172
7.13 /home/runner/work/dsci-capstone/dsci-capstone/src/main.py File Reference	173
7.14 /home/runner/work/dsci-capstone/dsci-capstone/src/setup.py File Reference	174
7.15 /home/runner/work/dsci-capstone/dsci-capstone/src/util.py File Reference	175
7.16 /home/runner/work/dsci-capstone/dsci-capstone/tests/conftest.py File Reference	175
7.17 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_components.py File Reference	175
7.18 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/_Imports.razor File Reference	177
7.19 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/App.razor File Reference	177
7.20 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/Main↔ Layout.razor File Reference	177
7.21 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/Nav↔ Menu.razor File Reference	177
7.22 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Error.razor File Reference	177
7.23 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Graph.razor File Reference	177
7.24 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Home.razor File Reference	177
7.25 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Metrics.razor File Reference	177
7.26 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Routes.razor File Reference	177
7.27 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/MetricsController.cs File Reference	177
7.28 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Hubs/MetricsHub.cs File Ref- erence	178
7.29 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/PRF1Metric.cs File Reference	178
7.30 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAItem.cs File Refer- ence	178
7.31 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAMetric.cs File Ref- erence	179
7.32 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs File Reference	179

7.33	/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryData.cs	File Reference	179
7.34	/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs	File Reference	179
Index			181

Chapter 1

Namespace Index

1.1 Package List

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

BlazorApp	9
BlazorApp.Controllers	9
BlazorApp.Hubs	9
BlazorApp.Models	9
components	10
components.book_conversion	10
components.connectors	11
components.corpus	11
components.document_storage	13
components.fact_storage	17
components.metrics	18
components.semantic_web	22
components.text_processing	22
src	23
src.flask	
Generic Flask worker microservice for distributed task processing	23
src.main	29
src.setup	40
src.util	41
tests	42
tests.conftest	42
tests.test_components	43

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Book	51
Chunk	54
ControllerBase	
MetricsController	129
EPUBToTEI	80
Hub	
MetricsHub	131
KnowledgeGraph	96
Log	112
Metrics	124
PRF1Metric	144
QALtem	145
QAMetric	146
RelationExtractor	156
RuntimeError	
Log.Failure	83
ScalarMetric	157
Session	157
Story	160
SummaryData	164
SummaryMetrics	165
ABC	
BookFactory	52
StoryStreamAdapter	162
BookStream	53
ParagraphStreamTEI	136
Connector	58
DatabaseConnector	61
RelationalConnector	147
mysqlConnector	132
postgresConnector	141
DocumentConnector	71
GraphConnector	85
LLMConnector	107

Chapter 3

Class Index

3.1 Class List

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

Book	51
BookFactory	52
BookStream	53
Chunk	
Lightweight container for a span of story text	54
Connector	
Abstract base class for external connectors	58
DatabaseConnector	
Abstract base class for database engine connectors	61
DocumentConnector	
Connector for MongoDB (document database)	71
EPUBToTEI	
Converts EPUB files to XML format (TEI specification)	80
Log.Failure	83
GraphConnector	
Connector for Neo4j (graph database)	85
KnowledgeGraph	
Manages a single graph within Neo4j	96
LLMConnector	
Connector for prompting and returning LLM output (raw text/JSON) via LangChain	107
Log	
Standardizes console output	112
Metrics	
Utility class for computing and posting evaluation metrics	124
MetricsController	129
MetricsHub	131
mysqlConnector	
A relational database connector configured for MySQL	132
ParagraphStreamTEI	
Streams paragraphs from a TEI file as Chunk objects	136
postgresConnector	
A relational database connector configured for PostgreSQL	141
PRF1Metric	144
QALtem	145
QAMetric	146

RelationalConnector	
Connector for relational databases (MySQL, PostgreSQL)	147
RelationExtractor	156
ScalarMetric	157
Session	
Stores active database connections and configuration settings	157
Story	160
StoryStreamAdapter	162
SummaryData	164
SummaryMetrics	165

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/dsci-capstone/dsci-capstone/components/___init___py	171
/home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py	167
/home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py	168
/home/runner/work/dsci-capstone/dsci-capstone/components/corpus.py	168
/home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py	169
/home/runner/work/dsci-capstone/dsci-capstone/components/fact_storage.py	169
/home/runner/work/dsci-capstone/dsci-capstone/components/metrics.py	170
/home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py	170
/home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py	171
/home/runner/work/dsci-capstone/dsci-capstone/src/___init___py	171
/home/runner/work/dsci-capstone/dsci-capstone/src/flask.py	172
/home/runner/work/dsci-capstone/dsci-capstone/src/main.py	173
/home/runner/work/dsci-capstone/dsci-capstone/src/setup.py	174
/home/runner/work/dsci-capstone/dsci-capstone/src/util.py	175
/home/runner/work/dsci-capstone/dsci-capstone/tests/___init___py	171
/home/runner/work/dsci-capstone/dsci-capstone/tests/conftest.py	175
/home/runner/work/dsci-capstone/dsci-capstone/tests/test_components.py	175
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/_Imports.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/App.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Routes.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/MainLayout.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/NavMenu.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Error.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Graph.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Home.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Metrics.razor	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/MetricsController.cs	177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Hubs/MetricsHub.cs	178
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/PRF1Metric.cs	178
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAItem.cs	178
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAMetric.cs	179
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs	179
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryData.cs	179
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs	179

Chapter 5

Namespace Documentation

5.1 BlazorApp Namespace Reference

Namespaces

- namespace [Controllers](#)
- namespace [Hubs](#)
- namespace [Models](#)

5.2 BlazorApp.Controllers Namespace Reference

Classes

- class [MetricsController](#)

5.3 BlazorApp.Hubs Namespace Reference

Classes

- class [MetricsHub](#)

5.4 BlazorApp.Models Namespace Reference

Classes

- class [PRF1Metric](#)
- class [QAItem](#)
- class [QAMetric](#)
- class [ScalarMetric](#)
- class [SummaryData](#)
- class [SummaryMetrics](#)

5.5 components Namespace Reference

Namespaces

- namespace [book_conversion](#)
- namespace [connectors](#)
- namespace [corpus](#)
- namespace [document_storage](#)
- namespace [fact_storage](#)
- namespace [metrics](#)
- namespace [semantic_web](#)
- namespace [text_processing](#)

5.6 components.book_conversion Namespace Reference

Classes

- class [Book](#)
- class [BookFactory](#)
- class [BookStream](#)
- class [Chunk](#)
Lightweight container for a span of story text.
- class [EPUBToTEI](#)
Converts EPUB files to XML format (TEI specification).
- class [ParagraphStreamTEI](#)
Streams paragraphs from a TEI file as Chunk objects.
- class [Story](#)
- class [StoryStreamAdapter](#)

Variables

- [nlp](#) = `spacy.blank("en")`
- [sentencizer](#) = `nlp.add_pipe("sentencizer")`

5.6.1 Variable Documentation

5.6.1.1 nlp

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

5.6.1.2 sentencizer

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

5.7 components.connectors Namespace Reference

Classes

- class [Connector](#)
Abstract base class for external connectors.
- class [DatabaseConnector](#)
Abstract base class for database engine connectors.
- class [mysqlConnector](#)
A relational database connector configured for MySQL.
- class [postgresConnector](#)
A relational database connector configured for PostgreSQL.
- class [RelationalConnector](#)
Connector for relational databases (MySQL, PostgreSQL).

5.8 components.corpus Namespace Reference

Functions

- [load_booksum](#) ()
- [to_df_booksum](#) (ds)
- [load_narrativeqa](#) ()
- [to_df_nqa](#) (ds)
- [normalize_title](#) (t)
- [merge_dataframes](#) (df1, df2, suffix1, suffix2, key_columns)
- [fuzzy_merge_titles](#) (df1, df2, suffix1, suffix2, key="title", threshold=90, scorer=fuzz.token_sort_ratio)
Perform a two-way fuzzy merge between two DataFrames on a text column (e.g., book titles).

Variables

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

5.8.1 Function Documentation

5.8.1.1 [fuzzy_merge_titles\(\)](#)

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

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

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

Parameters

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

Returns

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

Note

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

5.8.1.2 load_booksum()

```
load_booksum ( )
```

5.8.1.3 load_narrativeqa()

```
load_narrativeqa ( )
```

5.8.1.4 merge_dataframes()

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

5.8.1.5 normalize_title()

```
normalize_title (
    t )
```

5.8.1.6 to_df_booksum()

```
to_df_booksum (
    ds )
```


5.8.1.7 to_df_nqa()

```
to_df_nqa (
    ds )
```

5.8.2 Variable Documentation

5.8.2.1 df

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

5.8.2.2 df_booksum

```
df_booksum = load_booksum()
```

5.8.2.3 df_nqa

```
df_nqa = load_narrativeqa()
```

5.8.2.4 index

```
index
```

5.8.2.5 m

```
m = Metrics()
```

5.9 components.document_storage Namespace Reference

Classes

- class [DocumentConnector](#)
Connector for MongoDB (document database)

Functions

- [MongoHandle mongo_handle](#) (str host, str alias)
Establish a temporary connection to MongoDB.
- [DataFrame _flatten_recursive](#) (DataFrame df)
Explode all list columns and flatten dict columns until only scalars remain.
- [str _sanitize_json](#) (str text)
Remove comments and other non-JSON content from a MongoDB query string.
- [Dict\[str, Any\] _sanitize_document](#) (Dict[str, Any] doc, Dict[str, Set[Type[Any]]] type_registry)
Normalize document fields to consistent types for DataFrame construction.
- [DataFrame _docs_to_df](#) (List[Dict[str, Any]] docs, bool merge_unspecified=True)
Convert raw MongoDB documents to a Pandas DataFrame.
- [str _find_compatible_nested_key](#) (Type[Any] value_type, Dict[str, Set[Type[Any]]] nested_schema, bool merge_unspecified)
Find a nested column compatible with the given primitive type.

Variables

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

5.9.1 Function Documentation

5.9.1.1 `_docs_to_df()`

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

Convert raw MongoDB documents to a Pandas DataFrame.

Handles schema inconsistencies by:

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

Parameters

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

Exceptions

<i>Log.Failure</i>	If parsing query results to JSON fails.
--------------------	---

5.9.1.2 `_find_compatible_nested_key()`

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

Find a nested column compatible with the given primitive type.

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

Parameters

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

Returns

The nested key name to use for wrapping the primitive.

5.9.1.3 _flatten_recursive()

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

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

Recursive Process:

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

Parameters

<i>df</i>	DataFrame with potentially nested structures.
-----------	---

Returns

Fully flattened DataFrame with only scalar values.

5.9.1.4 _sanitize_document()

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

Normalize document fields to consistent types for DataFrame construction.

Converts all field values to lists and tracks type patterns.

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

Parameters

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

Returns

Document with all fields as lists.

5.9.1.5 `_sanitize_json()`

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

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

Removes the following elements:

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

Parameters

<i>text</i>	Raw text that may contain comments.
-------------	-------------------------------------

Returns

Cleaned text suitable for JSON parsing.

5.9.1.6 `mongo_handle()`

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

Establish a temporary connection to MongoDB.

Parameters

<i>host</i>	A valid MongoDB connection string.
<i>alias</i>	A unique name for the usage of this connection.

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

5.9.2 Variable Documentation

5.9.2.1 `MongoHandle`

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

5.10 components.fact_storage Namespace Reference

Classes

- class [GraphConnector](#)
Connector for Neo4j (graph database).

Functions

- 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.

5.10.1 Function Documentation

5.10.1.1 _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

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

Returns

Filtered DataFrame restricted to the active database.

5.10.1.2 _normalize_elements()

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

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

- Accepts the DataFrame output of `components.fact_storage.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

<code>df</code>	DataFrame containing dict-cast nodes and relationships.
-----------------	---

Returns

DataFrame suitable for downstream filtering and analysis.

5.10.1.3 `_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

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

Returns

DataFrame with requested columns.

5.11 `components.metrics` Namespace Reference

Classes

- class `Metrics`
Utility class for computing and posting evaluation metrics.

Functions

- Dict[str, Any] [run_questeval](#) (Dict[str, Any] chunk, *str qeval_task="summarization", bool use_cuda=False, bool use_question_weighter=True)
Run QuestEval metric calculation.
- Dict[str, Any] [run_bookscore](#) (Dict[str, Any] chunk, *str model="gpt-3.5-turbo", int batch_size=10, bool use_v2=True)
Run BoookScore metric for long-form summarization.
- str [chunk_bookscore](#) (str book_text, str book_title='book', int chunk_size=2048)
Chunk a book into BoookScore segments.

5.11.1 Function Documentation

5.11.1.1 [chunk_bookscore\(\)](#)

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

Chunk a book into BoookScore segments.

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

Parameters

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

Returns

Path to chunked pickle file containing BoookScore-ready segments.

Exceptions

<i>RuntimeError</i>	If BoookScore chunking fails.
---------------------	-------------------------------

5.11.1.2 [run_bookscore\(\)](#)

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

Run BoookScore metric for long-form summarization.

LLM-based coherence evaluation using BoookScore. 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

<i>chunk</i>	MongoDB document containing: <ul style="list-style-type: none"> • <i>summary</i>: Generated summary (required) • <i>text</i>: Full or partial book text (required) • <i>book_title</i>: Book title for identification (optional, for pickling)
<i>model</i>	Model name (optional, default 'gpt-4')
<i>batch_size</i>	Sentences per batch for v2 (optional, default 10)
<i>use_v2</i>	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

<i>KeyError</i>	If required fields are missing from chunk.
<i>RuntimeError</i>	If subprocess execution fails.

5.11.1.3 run_questeval()

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

Run QuestEval metric calculation.

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

Parameters

<i>chunk</i>	MongoDB document containing keys: <ul style="list-style-type: none"> • <i>summary</i>: Generated summary (required) • <i>text</i>: Source document text (required) • <i>gold_summary</i>: Reference summary (optional, filters for better questions)
<i>qeval_task</i>	Task performed by QuestEval (optional, default is summarization). Must be one of the following: generation / nlg, qa, dialogue, data2text, translation.
<i>use_cuda</i>	Run transformers with GPU enabled.
<i>use_question_weighter</i>	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

<i>ImportError</i>	If questeval package not installed.
<i>KeyError</i>	If required fields are missing from chunk.

5.12 components.semantic_web Namespace Reference

Classes

- class [KnowledgeGraph](#)
Manages a single graph within Neo4j.

5.13 components.text_processing Namespace Reference

Classes

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

Variables

- [nlp](#) = `spacy.blank("en")`
- [sentencizer](#) = `nlp.add_pipe("sentencizer")`

5.13.1 Variable Documentation

5.13.1.1 nlp

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

5.13.1.2 sentencizer

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

5.14 src Namespace Reference

Namespaces

- namespace [flask](#)
Generic Flask worker microservice for distributed task processing.
- namespace [main](#)
- namespace [setup](#)
- namespace [util](#)

5.15 src.flask Namespace Reference

Generic Flask worker microservice for distributed task processing.

Functions

- [process_task](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] chunk_doc, str [boss_url](#), Callable[[Dict[str, Any]], Dict[str, Any]] task_handler, Any task_kwargs=None)
Perform the assigned task in a background thread.
- str [load_mongo_config](#) (str database)
Load MongoDB configuration from environment variables.
- str [load_boss_config](#) ()
Load boss service callback URL from environment variables.
- Tuple[Callable[[Dict[str, Any]], Dict[str, Any]], Dict[str, Any]] [get_task_info](#) (str task_name)
Dynamically import and return the appropriate task handler function.
- [load_imports](#) (func)
Pre-warm the task by importing requirements.
- None [mark_task_in_progress](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name)
Mark a task as in-progress in MongoDB before processing begins.
- None [save_task_result](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] result)
Save completed task results to MongoDB.
- None [notify_boss](#) (str [boss_url](#), str chunk_id, str task_name, str status)
Send completion notification to boss service.
- Flask [create_app](#) (str task_name, str [boss_url](#))
Create and configure Flask application for task processing.

Variables

- [MongoHandle](#) = Generator["Database[Any]", None, None]
- [parser](#) = argparse.ArgumentParser(description="Flask worker microservice")
- [required](#)
- [True](#)
- [help](#)
- [args](#) = parser.parse_args()
- str [task_queue](#) = Queue()
- [target](#)
- [task_worker](#) ()
Background threading system for non-blocking task handling.
- [daemon](#)
- str [boss_url](#) = [load_boss_config](#)()
- [PORT](#) = int(os.environ["{args.task.upper()}_PORT"])
- Flask [app](#) = [create_app](#)(args.task, [boss_url](#))
- [host](#)
- [port](#)
- [use_reloader](#)

5.15.1 Detailed Description

Generic Flask worker microservice for distributed task processing.

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

5.15.2 Function Documentation

5.15.2.1 `create_app()`

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

Create and configure Flask application for task processing.

Parameters

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

Returns

Configured Flask application instance.

5.15.2.2 `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

<i>task_name</i>	Name of the task type to execute.
------------------	-----------------------------------

Returns

Callable that processes the task data and returns results.

Exceptions

<i>ImportError</i>	If the task module cannot be imported.
<i>AttributeError</i>	If the task function is not found in the module.

5.15.2.3 load_boss_config()

```
str load_boss_config ( )
```

Load boss service callback URL from environment variables.

Returns

Full callback URL for the boss service.

Exceptions

<i>KeyError</i>	If PYTHON_HOST environment variable is missing.
-----------------	---

5.15.2.4 load_imports()

```
load_imports (
    func )
```

Pre-warm the task by importing requirements.

Parameters

<i>func</i>	The function to perform a dummy call on.
-------------	--

5.15.2.5 load_mongo_config()

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

Load MongoDB configuration from environment variables.

Parameters

<i>database</i>	Name of the MongoDB database to connect to.
-----------------	---

Returns

MongoDB connection string.

Exceptions

<i>KeyError</i>	If required environment variables are missing.
-----------------	--

5.15.2.6 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

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

Exceptions

<i>RuntimeError</i>	If task data already exists (preventing overwrites).
---------------------	--

5.15.2.7 notify_boss()

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

Send completion notification to boss service.

Parameters

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

5.15.2.8 process_task()

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

Perform the assigned task in a background thread.

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

Parameters

<i>mongo_db</i>	MongoDB database instance.
<i>collection_name</i>	The name of the target MongoDB collection.
<i>chunk_id</i>	Unique identifier for the chunk within the story.
<i>task_name</i>	Name of the task being executed.
<i>chunk_doc</i>	Document data for the current chunk.
<i>boss_url</i>	Callback URL for the boss service.
<i>task_handler</i>	Function that performs the actual task computation.
<i>task_kwargs</i>	Dict of configuration settings for each task.

Exceptions

<i>Exception</i>	Logs and reports failures to the boss service.
------------------	--

5.15.2.9 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

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

5.15.3 Variable Documentation

5.15.3.1 app

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

5.15.3.2 args

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

5.15.3.3 boss_url

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

5.15.3.4 daemon

```
daemon
```

5.15.3.5 help

```
help
```

5.15.3.6 host

```
host
```

5.15.3.7 MongoHandle

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

5.15.3.8 parser

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

5.15.3.9 PORT

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

5.15.3.10 port

```
port
```

5.15.3.11 required

```
required
```

5.15.3.12 target

```
target
```


5.15.3.13 task_queue

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

5.15.3.14 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

<i>Exception</i>	Logs any runtime errors that occur during task execution.
------------------	---

5.15.3.15 True

```
True
```

5.15.3.16 use_reloader

```
use_reloader
```

5.16 src.main Namespace Reference

Functions

- [convert_single](#) ()
Converts one EPUB file to TEI format.
- [convert_from_csv](#) ()
Converts several EPUB files to TEI format.
- [chunk_single](#) ()
Creates a Story and many Chunks from a TEI file.
- [test_relation_extraction](#) ()
Runs REBEL on a basic example; used for debugging.
- [process_single](#) ()
Uses NLP and LLM to process an existing TEI file.
- [graph_triple_files](#) (session)
Loads JSON into Neo4j to test the Blazor graph page.
- [output_single](#) (session)

- Generates a summary from triples stored in JSON, and posts data to Blazor.*
- `full_pipeline` (`session`, `collection_name`, `epub_path`, `book_chapters`, `start_str`, `end_str`, `book_id`, `story_id`, `book_title`)
- `old_main` (`session`, `collection_name`)
- `pipeline_1` (`epub_path`, `book_chapters`, `start_str`, `end_str`, `book_id`, `story_id`)
- Connects all components to convert an EPUB file to a book summary.*
- `pipeline_2` (`session`, `collection_name`, `chunks`, `book_title`)
- Extracts triples from a random chunk.*
- `pipeline_3` (`session`, `triples`)
- Generates a LLM summary using Neo4j triples.*
- `pipeline_4` (`session`, `collection_name`, `triples_string`, `chunk_id`)
- Generate chunk summary.*
- `pipeline_5a` (`summary`, `book_title`, `book_id`)
- Send book info to Blazor.*
- `pipeline_5b` (`summary`, `book_title`, `book_id`, `chunk`, `gold_summary=""`, `float bookscore=None`, `float questeval=None`)
- Send metrics to Blazor.*
- `Dict[str, str] load_worker_config` (`List[str] task_types`)
- Load worker service URLs from environment variables.*
- `None clear_task_data` (`MongoHandle mongo_db`, `str collection_name`, `str chunk_id`, `str task_name`)
- Clear any existing task data before assigning new task to worker.*
- `bool assign_task_to_worker` (`str worker_url`, `str database_name`, `str collection_name`, `str chunk_id`)
- Assign a task to a worker microservice.*
- `Flask create_app` (`DocumentConnector docs_db`, `str database_name`, `str collection_name`, `Dict[str, str] worker_urls`)
- Create and configure Flask application for boss service.*
- `requests.models.Response post_story_status` (`int boss_port`, `int story_id`, `str task`, `str status`)
- Helpers to interact with the Flask boss thread.*
- `requests.models.Response post_chunk_status` (`int boss_port`, `str chunk_id`, `int story_id`, `str task`, `str status`)
- Send a chunk-level update to the boss Flask app.*
- `requests.models.Response post_process_full_story` (`int boss_port`, `int story_id`, `str task_type`)
- Process all chunks in MongoDB matching the provided story ID.*

Variables

- `str tei = "/datasets/examples/trilogy-wishes-1.tei"`
- Will revisit later - Book classes need refactoring ###.*
- `str chapters`
- `str start = ""`
- `str end = "But I must say no more."`
- `list triple_files`
- `list response_files = ["/datasets/triples/chunk-160_story-1.txt"]`
- `MongoHandle = Generator["Database[Any]", None, None]`
- `session = Session(verbose=False)`
- `DB_NAME = os.environ["DB_NAME"]`
- `BOSS_PORT = int(os.environ["PYTHON_PORT"])`
- `COLLECTION = os.environ["COLLECTION_NAME"]`
- `mongo_db = session.docs_db.get_unmanaged_handle()`
- `collection = getattr(mongo_db, COLLECTION)`
- `list task_types = ["questeval", "bookscore"]`
- `Dict[str, str] worker_urls = load_worker_config(task_types)`
- `Flask app = create_app(session.docs_db, DB_NAME, COLLECTION, worker_urls)`

- `app run_app` = `lambda.run(host="0.0.0.0", port=BOSS_PORT, use_reloader=False)`
- `target`
- `daemon`
- `int story_id` = 1
- `int book_id` = 2
- `str book_title` = "The Phoenix and the Carpet"
- `chunks`
- `triples`
- `chunk`
- `chunk_id` = `chunk.get_chunk_id()`
- `triples_string` = `pipeline_3(session, triples)`
- `summary` = `pipeline_4(session, COLLECTION, triples_string, chunk.get_chunk_id())`
- `requests.models.Response response` = `post_process_full_story(BOSS_PORT, story_id, task_type)`

5.16.1 Function Documentation

5.16.1.1 `assign_task_to_worker()`

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

Assign a task to a worker microservice.

Parameters

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

Returns

True if task was successfully assigned, False otherwise.

5.16.1.2 `chunk_single()`

```
chunk_single ( )
```

Creates a Story and many Chunks from a TEI file.

Requires hard-coded specifications

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

5.16.1.3 clear_task_data()

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

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

Parameters

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

5.16.1.4 convert_from_csv()

```
convert_from_csv ( )
```

Converts several EPUB files to TEI format.

Note

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

5.16.1.5 convert_single()

```
convert_single ( )
```

Converts one EPUB file to TEI format.

5.16.1.6 create_app()

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

Create and configure Flask application for boss service.

Parameters

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

Returns

Configured Flask application instance.

5.16.1.7 full_pipeline()

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

5.16.1.8 graph_triple_files()

```
graph_triple_files (
    session )
```

Loads JSON into Neo4j to test the Blazor graph page.

5.16.1.9 load_worker_config()

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

Load worker service URLs from environment variables.

Parameters

<i>task_types</i>	List of valid task keys to use when searching the .env
-------------------	--

Returns

Dictionary mapping task names to worker URLs.

5.16.1.10 old_main()

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

5.16.1.11 output_single()

```
output_single (
    session )
```

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

5.16.1.12 pipeline_1()

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

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

Data conversions:

- EPUB file
- XML (TEI)

5.16.1.13 pipeline_2()

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

Extracts triples from a random chunk.

- JSON triples (NLP & LLM)

5.16.1.14 pipeline_3()

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

Generates a LLM summary using Neo4j triples.

- Neo4j graph database
- Blazor graph page

5.16.1.15 pipeline_4()

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

Generate chunk summary.

5.16.1.16 pipeline_5a()

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

Send book info to Blazor.

- Post to Blazor metrics page

5.16.1.17 pipeline_5b()

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

Send metrics to Blazor.

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

5.16.1.18 post_chunk_status()

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

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

Parameters

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

Returns

JSON response indicating success or failure.

5.16.1.19 post_process_full_story()

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

Process all chunks in MongoDB matching the provided story ID.

Parameters

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

Returns

JSON response indicating success or failure.

5.16.1.20 post_story_status()

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

Helpers to interact with the Flask boss thread.

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

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

Parameters

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

Returns

JSON response indicating success or failure.

5.16.1.21 process_single()

```
process_single ( )
```

Uses NLP and LLM to process an existing TEI file.

5.16.1.22 test_relation_extraction()

```
test_relation_extraction ( )
```

Runs REBEL on a basic example; used for debugging.

5.16.2 Variable Documentation

5.16.2.1 app

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

5.16.2.2 book_id

```
int book_id = 2
```

5.16.2.3 book_title

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

5.16.2.4 BOSS_PORT

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

5.16.2.5 chapters

```
str chapters
```

Initial value:

```
00001 = """
00002 CHAPTER 1 BEAUTIFUL AS THE DAY\n
00003 CHAPTER 2 GOLDEN GUINEAS\n
00004 CHAPTER 3 BEING WANTED\n
00005 CHAPTER 4 WINGS\n
00006 CHAPTER 5 NO WINGS\n
00007 CHAPTER 6 A CASTLE AND NO DINNER\n
00008 CHAPTER 7 A SIEGE AND BED\n
00009 CHAPTER 8 BIGGER THAN THE BAKER'S BOY\n
00010 CHAPTER 9 GROWN UP\n
00011 CHAPTER 10 SCALPS\n
00012 CHAPTER 11 THE LAST WISH\n
00013 """
```

5.16.2.6 chunk

chunk

5.16.2.7 chunk_id

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

5.16.2.8 chunks

chunks

Initial value:

```
00001 = pipeline_1(  
00002     epub_path="./datasets/examples/trilogy-wishes-2.epub",  
00003     book_chapters=,  
00004     start_str="",  
00005     end_str="end of the Phoenix and the Carpet.",  
00006     book_id=book_id,  
00007     story_id=story_id,  
00008 )
```

5.16.2.9 COLLECTION

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

5.16.2.10 collection

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

5.16.2.11 daemon

daemon

5.16.2.12 DB_NAME

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

5.16.2.13 end

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

5.16.2.14 mongo_db

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

5.16.2.15 MongoHandle

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

5.16.2.16 response

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

5.16.2.17 response_files

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

5.16.2.18 run_app

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

5.16.2.19 session

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

5.16.2.20 start

```
str start = ""
```

5.16.2.21 story_id

```
int story_id = 1
```

5.16.2.22 summary

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

5.16.2.23 target

```
target
```

5.16.2.24 task_types

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

5.16.2.25 `tei`

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

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

5.16.2.26 `triple_files`

```
list triple_files
```

Initial value:

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

5.16.2.27 `triples`

```
triples
```

5.16.2.28 `triples_string`

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

5.16.2.29 `worker_urls`

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

5.17 `src.setup` Namespace Reference

Classes

- class [Session](#)
Stores active database connections and configuration settings.

Variables

- `session` = [Session](#)()

5.17.1 Variable Documentation

5.17.1.1 `session`

```
session = Session()
```

5.18 src.util Namespace Reference

Classes

- class [Log](#)

The Log class standardizes console output.

Functions

- [all_none](#) (*args)
Checks if all provided args are None.
- DataFrame [df_natural_sorted](#) (DataFrame df, List[str] ignored_columns=[], List[str] sort_columns=[])
Sort a DataFrame in natural order using only certain columns.
- bool [check_values](#) (List[Any] results, List[Any] expected, bool verbose, str log_source, bool raise_error)
Safely compare two lists of values.

5.18.1 Function Documentation

5.18.1.1 all_none()

```
all_none (
    * args )
```

Checks if all provided args are None.

5.18.1.2 check_values()

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

Safely compare two lists of values.

Helper for [components.connectors.RelationalConnector.test_connection](#)

Parameters

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

Exceptions

<i>Log.Failure</i>	If any result does not match what was expected.
--------------------	---

5.18.1.3 df_natural_sorted()

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

Sort a DataFrame in natural order using only certain columns.

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

Parameters

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

5.19 tests Namespace Reference**Namespaces**

- namespace [conftest](#)
- namespace [test_components](#)

5.20 tests.conftest Namespace Reference**Functions**

- [pytest_addoption](#) (parser)
- [session](#) (request)

Fixture to create session.

5.20.1 Function Documentation**5.20.1.1 pytest_addoption()**

```
pytest_addoption (
    parser )
```

5.20.1.2 session()

```
session (
    request )
```

Fixture to create session.

5.21 tests.test_components Namespace Reference

Functions

- [RelationalConnector relational_db](#) ([Session session](#))
Fixture to get relational database connection.
- [DocumentConnector docs_db](#) ([Session session](#))
Fixture to get document database connection.
- [GraphConnector graph_db](#) ([Session session](#))
Fixture to get document database connection.
- None [test_db_relational_minimal](#) ([RelationalConnector relational_db](#))
Tests if the RelationalConnector has a valid connection string.
- None [test_db_docs_minimal](#) ([DocumentConnector docs_db](#))
Tests if the DocumentConnector has a valid connection string.
- None [test_db_graph_minimal](#) ([GraphConnector graph_db](#))
Tests if the GraphConnector has a valid connection string.
- None [test_db_relational_comprehensive](#) ([RelationalConnector relational_db](#))
Tests if the GraphConnector is working as intended.
- None [test_db_docs_comprehensive](#) ([DocumentConnector docs_db](#))
Tests if the GraphConnector is working as intended.
- None [test_db_graph_comprehensive](#) ([GraphConnector graph_db](#))
Tests if the GraphConnector is working as intended.
- Generator[None, None, None] [load_examples_relational](#) ([RelationalConnector relational_db](#))
Fixture to create relational tables using engine-specific syntax.
- None [test_sql_example_1](#) ([RelationalConnector relational_db](#), Generator[None, None, None] [load_examples_relational](#))
Run queries contained within test files.
- None [test_sql_example_2](#) ([RelationalConnector relational_db](#), Generator[None, None, None] [load_examples_relational](#))
Run queries contained within test files.
- None [test_mongo_example_1](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_mongo_example_2](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_mongo_example_3](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_cypher_example_1](#) ([GraphConnector graph_db](#))
Run queries contained within test files.
- None [test_cypher_example_2](#) ([GraphConnector graph_db](#))
Test social network graph with relationships and mixed query patterns.
- None [test_cypher_example_3](#) ([GraphConnector graph_db](#))
Test scene and dialogue graphs with proper isolation.
- None [test_cypher_example_4](#) ([GraphConnector graph_db](#))
Test event graph with property mutations and multi-hop traversal.

- None `_test_query_file` (`DatabaseConnector` `db_fixture`, `str` `filename`, `List[str]` `valid_files`)
Run queries from a local file through the database.
- None `test_knowledge_graph_triples` (`GraphConnector` `graph_db`)
Test KnowledgeGraph triple operations using `add_triple` and `get_all_triples`.
- `KnowledgeGraph` `nature_scene_graph` (`GraphConnector` `graph_db`)
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.

5.21.1 Function Documentation

5.21.1.1 `_test_query_file()`

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

Run queries from a local file through the database.

Parameters

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

5.21.1.2 `docs_db()`

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

Fixture to get document database connection.

5.21.1.3 `graph_db()`

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

Fixture to get document database connection.

5.21.1.4 load_examples_relational()

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

Fixture to create relational tables using engine-specific syntax.

5.21.1.5 nature_scene_graph()

```
KnowledgeGraph nature_scene_graph (
    GraphConnector graph_db )
```

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.

5.21.1.6 relational_db()

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

Fixture to get relational database connection.

5.21.1.7 test_cypher_example_1()

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

Run queries contained within test files.

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

5.21.1.8 test_cypher_example_2()

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

Test social network graph with relationships and mixed query patterns.

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

5.21.1.9 test_cypher_example_3()

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

Test scene and dialogue graphs with proper isolation.

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

5.21.1.10 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.

5.21.1.11 test_db_docs_comprehensive()

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

Tests if the GraphConnector is working as intended.

5.21.1.12 test_db_docs_minimal()

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

Tests if the DocumentConnector has a valid connection string.

5.21.1.13 test_db_graph_comprehensive()

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

Tests if the GraphConnector is working as intended.

5.21.1.14 test_db_graph_minimal()

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

Tests if the GraphConnector has a valid connection string.

5.21.1.15 test_db_relational_comprehensive()

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

Tests if the GraphConnector is working as intended.

5.21.1.16 test_db_relational_minimal()

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

Tests if the RelationalConnector has a valid connection string.

5.21.1.17 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

5.21.1.18 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.

5.21.1.19 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.

5.21.1.20 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

5.21.1.21 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.

5.21.1.22 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

5.21.1.23 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.

5.21.1.24 test_knowledge_graph_triples()

```
None test_knowledge_graph_triples (
    GraphConnector graph_db )
```

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

5.21.1.25 test_mongo_example_1()

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

Run queries contained within test files.

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

5.21.1.26 test_mongo_example_2()

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

Run queries contained within test files.

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

5.21.1.27 test_mongo_example_3()

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

Run queries contained within test files.

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

5.21.1.28 test_sql_example_1()

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

Run queries contained within test files.

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

Note

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

5.21.1.29 test_sql_example_2()

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

Run queries contained within test files.

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

Note

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

Chapter 6

Class Documentation

6.1 Book Class Reference

Public Member Functions

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

6.1.1 Constructor & Destructor Documentation

6.1.1.1 `__init__()`

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

6.1.2 Member Function Documentation

6.1.2.1 `stream_chapters()`

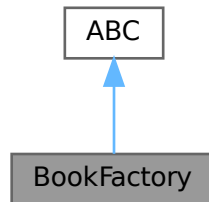
```
Iterator[Tuple[str, Dict[str, Any]]] stream_chapters (
    self )
```

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

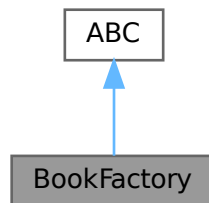
- `/home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py`

6.2 BookFactory Class Reference

Inheritance diagram for BookFactory:



Collaboration diagram for BookFactory:



Public Member Functions

- [Book create_book](#) (self)

6.2.1 Member Function Documentation

6.2.1.1 create_book()

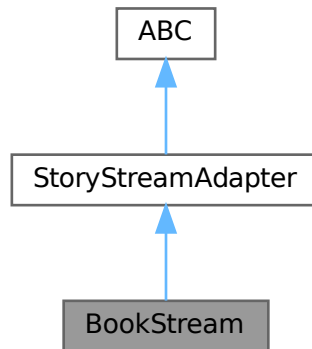
```
Book create_book (  
    self )
```

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

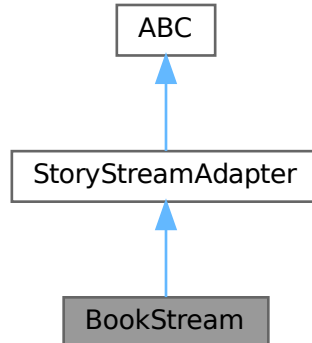
- [/home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py](#)

6.3 BookStream Class Reference

Inheritance diagram for BookStream:



Collaboration diagram for BookStream:



Public Member Functions

- None `__init__` (self, [Book book](#))
- Iterator[[Chunk](#)] `stream_segments` (self)
Yields sanitized parts of a book.

Public Member Functions inherited from [StoryStreamAdapter](#)

- Iterator[[Chunk](#)] `stream_paragraphs` (self)
Concrete helper method to split segments into paragraphs.
- Iterator[str] `stream_sentences` (self)
Concrete helper method to split paragraphs into sentences.

Public Attributes

- [book](#)

6.3.1 Constructor & Destructor Documentation

6.3.1.1 `__init__()`

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

6.3.2 Member Function Documentation

6.3.2.1 `stream_segments()`

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

Yields sanitized parts of a book.

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

Reimplemented from [StoryStreamAdapter](#).

6.3.3 Member Data Documentation

6.3.3.1 `book`

`book`

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

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

6.4 Chunk Class Reference

Lightweight container for a span of story text.

Public Member Functions

- None `__init__` (self, str `text`, int `book_id`, int `chapter_number`, int `line_start`, int `line_end`, int `story_id`, float `story_percent`, float `chapter_percent`, int `max_chunk_length=-1`)
Construct a Chunk.
- int `char_count` (self, bool `prune_newlines=False`)
Computes the character count.
- str `get_chunk_id` (self)
Use story ID, book ID, chapter, and chapter percentage to generate a chunk ID.
- Dict[str, Any] `to_mongo_dict` (self)
Convert Chunk to Mongo document format.
- str `__repr__` (self)

Public Attributes

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

6.4.1 Detailed Description

Lightweight container for a span of story text.

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

6.4.2 Constructor & Destructor Documentation

6.4.2.1 `__init__()`

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

Construct a Chunk.

Parameters

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

Exceptions

<i>ValueError</i>	if text exceeds max_chunk_length when max_chunk_length > 0.
-------------------	---

6.4.3 Member Function Documentation**6.4.3.1 __repr__()**

```
str __repr__ (
    self )
```

6.4.3.2 char_count()

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

Computes the character count.

Parameters

<i>prune_newlines</i>	Whether to remove newlines for the count.
-----------------------	---

Returns

The number of characters in the chunk text.

6.4.3.3 get_chunk_id()

```
str get_chunk_id (
    self )
```

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

Returns

A string uniquely identifying a chunk.

6.4.3.4 to_mongo_dict()

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

Convert Chunk to Mongo document format.

Returns

A dictionary which can be easily loaded into MongoDB.

6.4.4 Member Data Documentation**6.4.4.1 book_id**

```
book_id
```

6.4.4.2 chapter_number

```
chapter_number
```

6.4.4.3 chapter_percent

```
chapter_percent
```

6.4.4.4 line_end

```
line_end
```

6.4.4.5 line_start

```
line_start
```

6.4.4.6 story_id

```
story_id
```

6.4.4.7 story_percent

```
story_percent
```

6.4.4.8 text

text

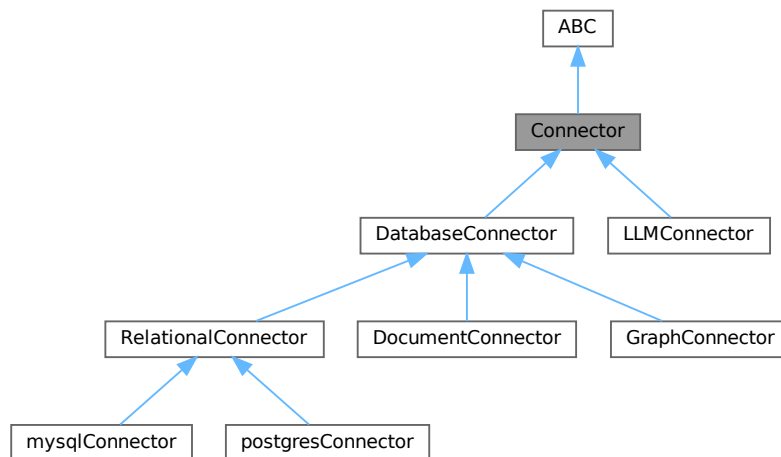
The documentation for this class was generated from the following file:

- [/home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py](#)

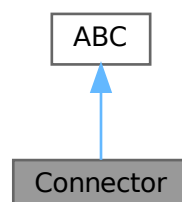
6.5 Connector Class Reference

Abstract base class for external connectors.

Inheritance diagram for Connector:



Collaboration diagram for Connector:



Public Member Functions

- None [configure](#) (self, str DB, str database_name)
Read connection settings from the .env file.
- bool [test_connection](#) (self, bool raise_error=True)
Establish a basic connection to the database, and test full functionality.
- bool [check_connection](#) (self, str log_source, bool raise_error)
Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] [execute_query](#) (self, str query)
Send a single command through the connection.
- List[Optional[DataFrame]] [execute_file](#) (self, str filename)
Run several commands from a file.

6.5.1 Detailed Description

Abstract base class for external connectors.

Note

Credentials are specified in the .env file.

Derived classes should implement:

- **init**
- [components.connectors.Connector.configure](#)
- [components.connectors.Connector.test_connection](#)
- [components.connectors.Connector.execute_query](#)
- [components.connectors.Connector.execute_file](#)

6.5.2 Member Function Documentation

6.5.2.1 check_connection()

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

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

Parameters

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

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If raise_error is True and the connection test fails to complete.
--------------------	---

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), [GraphConnector](#), and [LLMConnector](#).

6.5.2.2 configure()

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

Read connection settings from the .env file.

Parameters

<i>DB</i>	The prefix of fetched credentials.
<i>database_name</i>	The specific service to connect to.

Reimplemented in [LLMConnector](#), and [DatabaseConnector](#).

6.5.2.3 execute_file()

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

Run several commands from a file.

Parameters

<i>filename</i>	The path to a specified query or prompt file (.sql, .txt).
-----------------	--

Returns

Whether the query was performed successfully.

Reimplemented in [DatabaseConnector](#), and [LLMConnector](#).

6.5.2.4 execute_query()

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

Send a single command through the connection.

Parameters

<i>query</i>	A single query to perform on the database.
--------------	--

Returns

The result of the query, or None

Reimplemented in [DatabaseConnector](#), [RelationalConnector](#), [DocumentConnector](#), [LLMConnector](#), and [GraphConnector](#).

6.5.2.5 test_connection()

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

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

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

Parameters

<i>raise_error</i>	Whether to raise an error on connection failure.
--------------------	--

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True and the connection test fails to complete.
--------------------	--

Reimplemented in [LLMConnector](#), [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

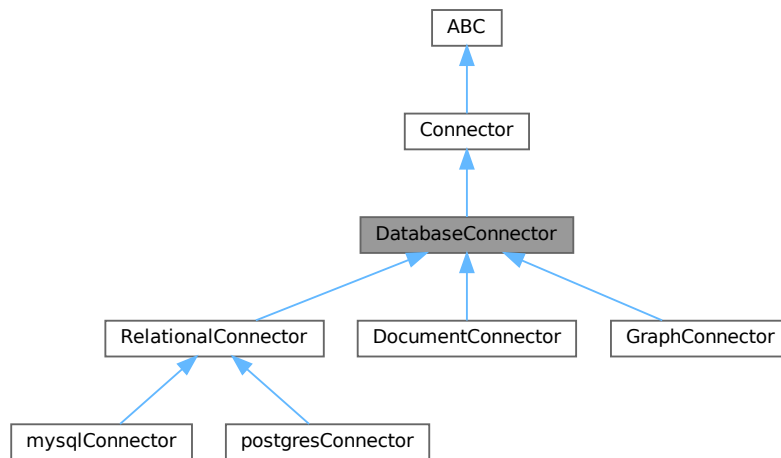
The documentation for this class was generated from the following file:

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

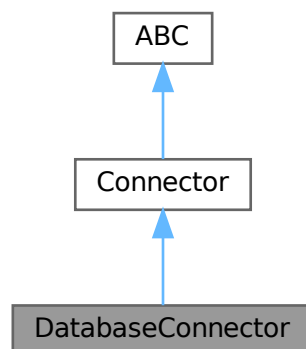
6.6 DatabaseConnector Class Reference

Abstract base class for database engine connectors.

Inheritance diagram for DatabaseConnector:



Collaboration diagram for DatabaseConnector:



Public Member Functions

- None `__init__` (self, bool `verbose`=False)
Initialize the connector.
- None `configure` (self, str DB, str database_name)
Read connection settings from the .env file.
- None `change_database` (self, str new_database)
Update the connection URI to reference a different database in the same engine.
- Generator[None, None, None] `temp_database` (self, str database_name)
Temporarily switch to a pseudo-database, creating and dropping it if needed.
- Optional[DataFrame] `execute_query` (self, str query)

- Send a single command through the connection.*

 - List[Optional[DataFrame]] [execute_combined](#) (self, str multi_query)
- Run several database commands in sequence.*

 - List[Optional[DataFrame]] [execute_file](#) (self, str filename)
- Run several database commands from a file.*

 - Optional[DataFrame] [get_dataframe](#) (self, str name, List[str] columns=[])
 - Automatically generate and run a query for the specified resource.*
- None [create_database](#) (self, str database_name)

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

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

 - Search for an existing database using the provided name.*

Public Member Functions inherited from [Connector](#)

- bool [test_connection](#) (self, bool raise_error=True)
 - Establish a basic connection to the database, and test full functionality.*
- bool [check_connection](#) (self, str log_source, bool raise_error)
 - Minimal connection test to determine if our connection string is valid.*

Public Attributes

- [verbose](#)
 - Whether to print debug messages.*
- [db_type](#)
- [db_engine](#)
- [username](#)
- [password](#)
- [host](#)
- [port](#)
- [connection_string](#)

Protected Member Functions

- bool [_is_single_query](#) (self, str query)
 - Checks if a string contains multiple queries.*
- List[str] [_split_combined](#) (self, str multi_query)
 - Checks if a string contains multiple queries.*
- 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.*

6.6.1 Detailed Description

Abstract base class for database engine connectors.

Derived classes should implement:

- [components.connectors.DatabaseConnector.__init__](#)
- [components.connectors.DatabaseConnector.test_connection](#)
- [components.connectors.DatabaseConnector.execute_query](#)
- [components.connectors.DatabaseConnector._split_combined](#)
- [components.connectors.DatabaseConnector.get_dataframe](#)
- [components.connectors.DatabaseConnector.create_database](#)
- [components.connectors.DatabaseConnector.drop_database](#)
- [components.connectors.DatabaseConnector.change_database](#)
- [components.connectors.DatabaseConnector.database_exists](#)

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

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

Initialize the connector.

Parameters

<i>verbose</i>	Whether to print debug messages.
----------------	----------------------------------

Note

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

Reimplemented in [RelationalConnector](#), [mysqlConnector](#), [postgresConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3 Member Function Documentation

6.6.3.1 _is_single_query()

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

Checks if a string contains multiple queries.

Parameters

<i>query</i>	A single or combined query string.
--------------	------------------------------------

Returns

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

6.6.3.2 `_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

<i>result</i>	The result of a SQL, Cypher, or JSON query.
---------------	---

Returns

Whether the object is parsable to DataFrame.

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.3 `_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

<i>query</i>	A single query string.
--------------	------------------------

Returns

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

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.4 `_split_combined()`

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

Checks if a string contains multiple queries.

Parameters

<i>multi_query</i>	A string containing multiple queries.
--------------------	---------------------------------------

Returns

A list of single-query strings.

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.5 `change_database()`

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

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

Parameters

<i>new_database</i>	The name of the database to connect to.
---------------------	---

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.6 `configure()`

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

Read connection settings from the .env file.

Parameters

<i>DB</i>	The prefix of fetched database credentials.
<i>database_name</i>	The name of the database to connect to.

Reimplemented from [Connector](#).

6.6.3.7 create_database()

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

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

Parameters

<i>database_name</i>	The name of the new database to create.
----------------------	---

Exceptions

<i>Log.Failure</i>	If the database already exists.
--------------------	---------------------------------

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.8 database_exists()

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

Search for an existing database using the provided name.

Parameters

<i>database_name</i>	The name of a database to search for.
----------------------	---------------------------------------

Returns

Whether the database is visible to this connector.

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.9 drop_database()

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

Delete all data stored in a particular database.

Parameters

<i>database_name</i>	The name of an existing database.
----------------------	-----------------------------------

Exceptions

<i>Log.Failure</i>	If the database does not exist.
--------------------	---------------------------------

Reimplemented in [DocumentConnector](#), [GraphConnector](#), and [RelationalConnector](#).

6.6.3.10 execute_combined()

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

Run several database commands in sequence.

Parameters

<i>multi_query</i>	A string containing multiple queries.
--------------------	---------------------------------------

Returns

A list of query results converted to DataFrames.

6.6.3.11 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

<i>filename</i>	The path to a specified query file (.sql, .cql, .json).
-----------------	---

Returns

Whether the query was performed successfully.

Exceptions

<i>Log.Failure</i>	If any query in the file fails to execute.
--------------------	--

Reimplemented from [Connector](#).

6.6.3.12 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

<i>query</i>	A single query to perform on the database.
--------------	--

Returns

DataFrame containing the result of the query, or None

Exceptions

<i>Log.Failure</i>	If the query fails to execute.
--------------------	--------------------------------

Reimplemented from [Connector](#).

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.13 get_dataframe()

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

Automatically generate and run a query for the specified resource.

Parameters

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

Returns

DataFrame containing the requested data, or None

Reimplemented in [RelationalConnector](#), [DocumentConnector](#), and [GraphConnector](#).

6.6.3.14 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

<i>database_name</i>	The name of the pseudo-database to use temporarily.
----------------------	---

6.6.4 Member Data Documentation

6.6.4.1 connection_string

connection_string

6.6.4.2 db_engine

db_engine

6.6.4.3 db_type

db_type

6.6.4.4 host

host

6.6.4.5 password

password

6.6.4.6 port

port

6.6.4.7 username

username

6.6.4.8 verbose

verbose

Whether to print debug messages.

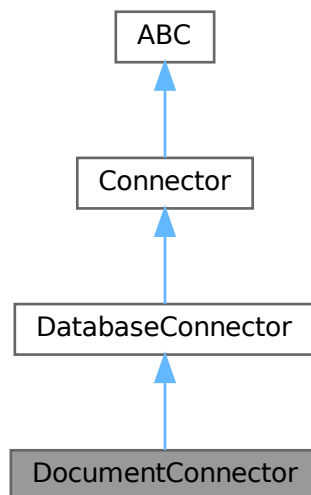
The documentation for this class was generated from the following file:

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

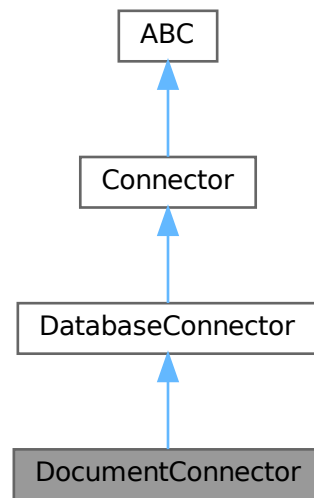
6.7 DocumentConnector Class Reference

Connector for MongoDB (document database)

Inheritance diagram for DocumentConnector:



Collaboration diagram for DocumentConnector:



Public Member Functions

- None `__init__` (self, bool `verbose=False`)
Creates a new MongoDB connector.
- None `change_database` (self, str `new_database`)
Update the connection URI to reference a different database in the same engine.
- bool `test_connection` (self, bool `raise_error=True`)
Establish a basic connection to the MongoDB database, and test full functionality.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- `get_unmanaged_handle` (self)
Expose the low-level PyMongo handle for external use.
- Optional[DataFrame] `execute_query` (self, str `query`)
Send a single MongoDB command using PyMongo.
- Optional[DataFrame] `get_dataframe` (self, str `name`, List[str] `columns=[]`)
Automatically generate and run a query for the specified collection.
- None `create_database` (self, str `database_name`)
Use the current database connection to create a sibling database in this engine.
- None `drop_database` (self, str `database_name`)
Delete all data stored in a particular database.
- bool `database_exists` (self, str `database_name`)
Search for an existing database using the provided name.
- None `delete_dummy` (self)
Delete the initial dummy collection from the database.

Public Member Functions inherited from DatabaseConnector

- None [configure](#) (self, str DB, str database_name)
Read connection settings from the .env file.
- Generator[None, None, None] [temp_database](#) (self, str database_name)
Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] [execute_combined](#) (self, str multi_query)
Run several database commands in sequence.
- List[Optional[DataFrame]] [execute_file](#) (self, str filename)
Run several database commands from a file.

Public Attributes

- [database_name](#)
- [verbose](#)
- [connection_string](#)

Public Attributes inherited from DatabaseConnector

- [verbose](#)
Whether to print debug messages.
- [db_type](#)
- [db_engine](#)
- [username](#)
- [password](#)
- [host](#)
- [port](#)
- [connection_string](#)

Protected Member Functions

- list[str] [_split_combined](#) (self, str multi_query)
Divides a string into non-divisible MongoDB commands by splitting on semicolons at depth 0.
- 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](#)

6.7.1 Detailed Description

Connector for MongoDB (document database)

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

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `__init__()`

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

Creates a new MongoDB connector.

Parameters

<i>verbose</i>	Whether to print debug messages.
----------------	----------------------------------

Reimplemented from [DatabaseConnector](#).

6.7.3 Member Function Documentation

6.7.3.1 `_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

<i>result</i>	The result of a JSON query.
---------------	-----------------------------

Returns

Whether the object is parsable to DataFrame.

Reimplemented from [DatabaseConnector](#).

6.7.3.2 _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

<i>query</i>	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](#).

6.7.3.3 _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

<i>multi_query</i>	A string containing multiple queries with possible comments.
--------------------	--

Returns

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

Reimplemented from [DatabaseConnector](#).

6.7.3.4 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

<i>new_database</i>	The name of the database to connect to.
---------------------	---

Reimplemented from [DatabaseConnector](#).

6.7.3.5 check_connection()

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

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

Connect to MongoDB using `MongoEngine.connect()`

Parameters

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

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <code>raise_error</code> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.7.3.6 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

<i>database_name</i>	The name of the new database to create.
----------------------	---

Exceptions

<i>Log.Failure</i>	If we fail to create the requested database for any reason.
--------------------	---

Reimplemented from [DatabaseConnector](#).

6.7.3.7 database_exists()

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

Search for an existing database using the provided name.

Parameters

<i>database_name</i>	The name of a database to search for.
----------------------	---------------------------------------

Returns

Whether the database is visible to this connector.

Reimplemented from [DatabaseConnector](#).

6.7.3.8 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.

6.7.3.9 drop_database()

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

Delete all data stored in a particular database.

Parameters

<i>database_name</i>	The name of an existing database.
----------------------	-----------------------------------

Exceptions

<i>Log.Failure</i>	If we fail to drop the target database for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.7.3.10 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

<i>Log.Failure</i>	If the query fails to execute.
--------------------	--------------------------------

Reimplemented from [DatabaseConnector](#).

6.7.3.11 get_dataframe()

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

Automatically generate and run a query for the specified collection.

Parameters

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

Returns

DataFrame containing the requested data, or None

Exceptions

<i>Log.Failure</i>	If we fail to create the requested DataFrame for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.7.3.12 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.

6.7.3.13 test_connection()

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

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

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

Parameters

<i>raise_error</i>	Whether to raise an error on connection failure.
--------------------	--

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.7.4 Member Data Documentation

6.7.4.1 `_auth_suffix`

`_auth_suffix` [protected]

6.7.4.2 `connection_string`

`connection_string`

6.7.4.3 `database_name`

`database_name`

6.7.4.4 `verbose`

`verbose`

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

- [/home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py](#)

6.8 EPUBToTEI Class Reference

Converts EPUB files to XML format (TEI specification).

Public Member Functions

- None `__init__` (self, str [epub_path](#), bool save_pandoc=False, bool save_tei=True)
Initialize the converter.
- None `convert_to_tei` (self)
Uses Pandoc to draft a TEI string from EPUB.
- None `clean_tei` (self)
Wrap root if missing, sanitize ids, and save cleaned TEI.

Public Attributes

- [epub_path](#)
- [pandoc_xml_path](#)
- [raw_tei_content](#)
- [clean_tei_content](#)
- [tei_path](#)

Static Public Attributes

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

Protected Member Functions

- str `_sanitize_ids` (self, str content)
Sanitize XML IDs in the TEI content to ensure they are valid and consistent.
- str `_prune_bad_tags` (self, str content)
Replace all `lb` tags with newline characters in TEI.

6.8.1 Detailed Description

Converts EPUB files to XML format (TEI specification).

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

6.8.2 Constructor & Destructor Documentation

6.8.2.1 `__init__()`

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

Initialize the converter.

Parameters

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

6.8.3 Member Function Documentation

6.8.3.1 `_prune_bad_tags()`

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

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

6.8.3.2 `_sanitize_ids()`

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

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

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

Parameters

<code>content</code>	The raw TEI XML string possibly containing invalid <code>xml:id</code> attributes.
----------------------	--

Returns

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

6.8.3.3 `clean_tei()`

```
None clean_tei (
    self )
```

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

6.8.3.4 `convert_to_tei()`

```
None convert_to_tei (
    self )
```

Uses Pandoc to draft a TEI string from EPUB.

6.8.4 Member Data Documentation

6.8.4.1 `clean_tei_content`

```
clean_tei_content
```

6.8.4.2 `encoding`

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

6.8.4.3 `epub_path`

```
epub_path
```

6.8.4.4 pandoc_xml_path

```
pandoc_xml_path
```

6.8.4.5 raw_tei_content

```
raw_tei_content
```

6.8.4.6 tei_path

```
tei_path
```

6.8.4.7 xml_namespace

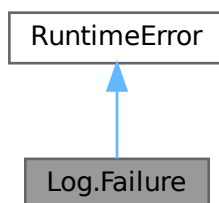
```
dict xml_namespace = {"tei": "http://www.tei-c.org/ns/1.0"} [static]
```

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

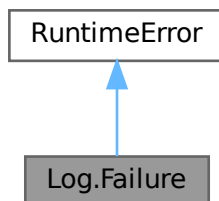
- [/home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py](#)

6.9 Log.Failure Class Reference

Inheritance diagram for Log.Failure:



Collaboration diagram for Log.Failure:



Public Member Functions

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

Public Attributes

- `prefix`
- `msg`

6.9.1 Constructor & Destructor Documentation

6.9.1.1 `__init__()`

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

6.9.2 Member Function Documentation

6.9.2.1 `__str__()`

```
__str__ (
    self )
```

6.9.3 Member Data Documentation

6.9.3.1 `msg`

`msg`

6.9.3.2 `prefix`

`prefix`

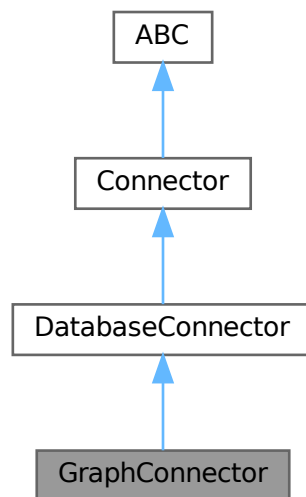
The documentation for this class was generated from the following file:

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

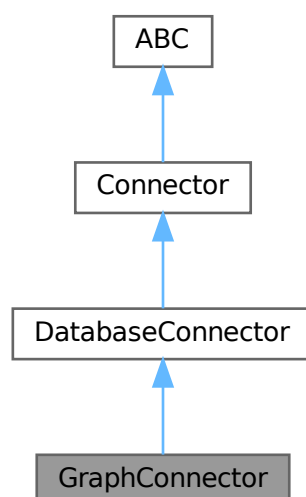
6.10 GraphConnector Class Reference

Connector for Neo4j (graph database).

Inheritance diagram for GraphConnector:



Collaboration diagram for GraphConnector:



Public Member Functions

- None `__init__` (self, bool `verbose`=False)
Creates a new Neo4j connector.
- None `change_database` (self, str `new_database`)
Update the connection URI to reference a different database in the same engine.
- Generator[None, None, None] `temp_graph` (self, str `graph_name`)
Temporarily inspect the specified graph, then swap back when finished.
- bool `test_connection` (self, bool `raise_error`=True)
Establish a basic connection to the Neo4j database, and test full functionality.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] `execute_query` (self, str `query`, bool `_filter_results`=True)
Send a single Cypher query to Neo4j.
- Optional[DataFrame] `get_dataframe` (self, str `name`, List[str] `columns`=[])
Automatically generate and run a query for the specified Knowledge Graph collection.
- List[str] `get_unique` (self, str `key`)
Retrieve all unique values for a specified node property.
- None `create_database` (self, str `database_name`)
Create a fresh pseudo-database if it does not already exist.
- None `drop_database` (self, str `database_name`)
Delete all nodes stored under a particular database name.
- None `drop_graph` (self, str `graph_name`)
Delete all nodes stored under a particular graph name.
- bool `database_exists` (self, str `database_name`)
Search for an existing database using the provided name.
- None `delete_dummy` (self)
Delete the initial dummy node from the database.
- 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`
TODO: remove once error handling is fixed check_values will raise, so this never became an issue until now.
- `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](#)

6.10.1 Detailed Description

Connector for Neo4j (graph database).

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

6.10.2 Constructor & Destructor Documentation

6.10.2.1 __init__()

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

Creates a new Neo4j connector.

Parameters

<i>verbose</i>	Whether to print success and failure messages.
----------------	--

Reimplemented from [DatabaseConnector](#).

6.10.3 Member Function Documentation

6.10.3.1 `_execute_retag_db()`

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

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

6.10.3.2 `_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

<i>results</i>	Original list of untagged tuples from <code>db.cypher_query()</code> .
----------------	--

Returns

Latest version of results fetched from the database.

6.10.3.3 `_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

<i>result</i>	The result of a Cypher query.
---------------	-------------------------------

Returns

Whether the object is parsable to DataFrame.

Reimplemented from [DatabaseConnector](#).

6.10.3.4 _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

<i>query</i>	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](#).

6.10.3.5 _split_combined()

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

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

Parameters

<i>multi_query</i>	A string containing multiple queries.
--------------------	---------------------------------------

Returns

A list of single-query strings.

Reimplemented from [DatabaseConnector](#).

6.10.3.6 `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

<i>new_database</i>	The name of the database to connect to.
---------------------	---

Reimplemented from [DatabaseConnector](#).

6.10.3.7 `check_connection()`

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

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

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

Parameters

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

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <code>raise_error</code> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.10.3.8 `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 [components.connectors.DatabaseConnector.change_database](#) is called.

Parameters

<i>database_name</i>	A database ID specifying the pseudo-database.
----------------------	---

Exceptions

<i>Log.Failure</i>	If we fail to create the requested database for any reason.
--------------------	---

Reimplemented from [DatabaseConnector](#).

6.10.3.9 database_exists()

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

Search for an existing database using the provided name.

Parameters

<i>database_name</i>	The name of a database to search for.
----------------------	---------------------------------------

Returns

Whether the database is visible to this connector.

Reimplemented from [DatabaseConnector](#).

6.10.3.10 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.

6.10.3.11 drop_database()

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

Delete all nodes stored under a particular database name.

Parameters

<i>database_name</i>	A database ID specifying the pseudo-database.
----------------------	---

Exceptions

<i>Log.Failure</i>	If we fail to drop the target database for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.10.3.12 drop_graph()

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

Delete all nodes stored under a particular graph name.

Parameters

<i>graph_name</i>	The name of a graph in the current database.
-------------------	--

Exceptions

<i>Log.Failure</i>	If we fail to drop the target graph for any reason.
--------------------	---

6.10.3.13 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

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

Returns

DataFrame containing the result of the query, or None

Exceptions

<i>Log.Failure</i>	If the query fails to execute.
--------------------	--------------------------------

Reimplemented from [DatabaseConnector](#).

6.10.3.14 get_dataframe()

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

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

- Fetches all 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

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

Returns

DataFrame containing the requested data, or None.

Exceptions

<i>Log.Failure</i>	If we fail to create the requested DataFrame for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.10.3.15 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

<i>key</i>	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

<i>Log.Failure</i>	If the query fails to execute.
--------------------	--------------------------------

6.10.3.16 IS_DUMMY_()

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

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

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

Returns

A string containing Cypher code.

6.10.3.17 NOT_DUMMY_()

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

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

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

Returns

A string containing Cypher code.

6.10.3.18 SAME_DB_KG_()

```
str SAME_DB_KG_ (
    self )
```

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

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

Returns

A string containing Cypher code.

6.10.3.19 temp_graph()

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

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

Parameters

<i>graph_name</i>	The name of a graph in the current database.
-------------------	--

6.10.3.20 test_connection()

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

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

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

Parameters

<i>raise_error</i>	Whether to raise an error on connection failure.
--------------------	--

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.10.4 Member Data Documentation

6.10.4.1 __graph_name

```
__graph_name [protected]
```

6.10.4.2 connection_string

```
connection_string
```

6.10.4.3 database_name

database_name

6.10.4.4 verbose

verbose

TODO: remove once error handling is fixed check_values will raise, so this never became an issue until now.

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

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

6.11 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.
- 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.
- 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.
- DataFrame `get_all_triples` (self)
Return all triples in the specified graph as a pandas DataFrame.
- DataFrame `get_subgraph_by_nodes` (self, List[str] node_ids)
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_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_triple_string` (self, Optional[DataFrame] triples_df=None, str format="natural")

- 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=10)

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.

Public Attributes

- `graph_name`
The name of this graph.
- `database`
Reference to a pre-configured graph database wrapper.
- `verbose`
Whether to print debug messages.

6.11.1 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.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 `__init__()`

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

6.11.3 Member Function Documentation

6.11.3.1 `add_triple()`

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

Add a semantic triple to the graph using raw Cypher.

Parameters

<i>subject</i>	A string representing the entity performing an action.
<i>relation</i>	A string describing the action.
<i>object</i> ↔	A string representing the entity being acted upon.
—	

Note

LLM output should be pre-normalized using [components.text_processing.LLMConnector.normalize_triples](#).

Exceptions

<i>Log.Failure</i>	If the triple cannot be added to our graph database.
--------------------	--

6.11.3.2 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

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

Exceptions

<i>Log.Failure</i>	If GDS is unavailable or any query fails.
--------------------	---

6.11.3.3 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

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

Returns

DataFrame with added columns: *name_columns*.

Exceptions

<i>Log.Failure</i>	If mapping fails or required IDs are missing.
--------------------	---

6.11.3.4 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

<i>Log.Failure</i>	If the query fails to retrieve or process the DataFrame.
--------------------	--

6.11.3.5 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

<i>community_id</i>	The identifier of the community to retrieve.
---------------------	--

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

<i>Log.Failure</i>	If the query fails to retrieve the requested DataFrame or community detection has not been run.
--------------------	---

6.11.3.6 get_edge_counts()

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

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

Parameters

<i>top_n</i>	Number of top nodes to return (by edge count). Default is 10.
--------------	---

Returns

DataFrame with columns: node_id, edge_count

Exceptions

<i>Log.Failure</i>	If the query fails to retrieve the requested DataFrame.
--------------------	---

6.11.3.7 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

<i>node↔ _id</i>	The element ID of the central node.
<i>depth</i>	Number of hops to traverse (default: 1).

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

<i>Log.Failure</i>	If the query fails to retrieve the requested DataFrame.
--------------------	---

6.11.3.8 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

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

Returns

Natural language description of the node.

Exceptions

<i>Log.Failure</i>	If the node does not exist in the graph.
--------------------	--

6.11.3.9 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

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

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

<i>Log.Failure</i>	If the query fails to retrieve the requested DataFrame.
--------------------	---

6.11.3.10 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

6.11.3.11 get_subgraph_by_nodes()

```
DataFrame get_subgraph_by_nodes (
    self,
    List[str] node_ids )
```

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

Parameters

<i>node_ids</i>	List of node element IDs to filter by.
-----------------	--

Returns

DataFrame with columns: subject_id, relation_id, object_id

Exceptions

<i>Log.Failure</i>	If the query fails to retrieve the requested DataFrame.
--------------------	---

6.11.3.12 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.

6.11.3.13 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

<i>Log.Failure</i>	If the elements DataFrame cannot be loaded or pivoting fails.
--------------------	---

6.11.3.14 print_nodes()

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

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

6.11.3.15 print_triples()

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

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

6.11.3.16 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

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

Returns

Formatted string with contextualized triple groups.

6.11.3.17 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

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

Returns

Natural language narrative describing the graph.

Exceptions

<i>ValueError</i>	If strategy is not recognized.
-------------------	--------------------------------

6.11.3.18 to_triple_string()

```
str to_triple_string (
    self,
    Optional[DataFrame] triples_df = None,
    str format = "natural" )
```

Convert triples to string representation in various formats.

Supports multiple output formats for LLM consumption:

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

Parameters

<i>triples↔ _df</i>	DataFrame with subject/relation/object columns. If None, uses all triples from this graph.
<i>format</i>	Output format: "natural", "triple", or "json" (default: "natural").

Returns

String representation of triples in the specified format.

Exceptions

<i>ValueError</i>	If format is not recognized.
-------------------	------------------------------

6.11.3.19 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

<i>df_ids</i>	DataFrame with added columns: subject_id, relation_id, object_id.
<i>drop_ids</i>	Whether to remove columns from results: subject_id, relation_id, object_id.
<i>df_lookup</i>	Optional DataFrame fetched from components.fact_storage.GraphConnector.get_dataframe with required columns: element_id, element_type, name, and rel_type.

Returns

DataFrame with added columns: subject, relation, object.

Exceptions

<i>Log.Failure</i>	If mapping fails or required IDs are missing.
--------------------	---

6.11.4 Member Data Documentation**6.11.4.1 database**

database

Reference to a pre-configured graph database wrapper.

6.11.4.2 graph_name

graph_name

The name of this graph.

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

6.11.4.3 verbose

verbose

Whether to print debug messages.

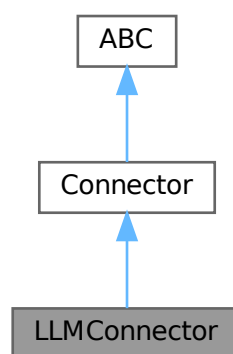
The documentation for this class was generated from the following file:

- [/home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py](#)

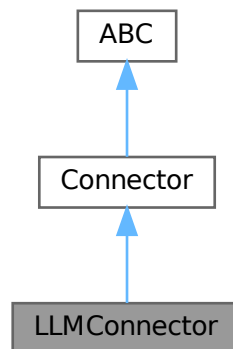
6.12 LLMConnector Class Reference

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

Inheritance diagram for LLMConnector:



Collaboration diagram for LLMConnector:



Public Member Functions

- `__init__` (self, float `temperature`=0, str `system_prompt`="You are a helpful assistant.")
Initialize the connector.
- `configure` (self)
Initialize the LangChain LLM using environment credentials.
- `test_connection` (self)
Send a trivial prompt to verify LLM connectivity.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- str `execute_full_query` (self, str `system_prompt`, str `human_prompt`)
Send a single prompt to the LLM with separate system and human instructions.
- str `execute_query` (self, str `query`)
Send a single prompt through the connection and return raw LLM output.
- str `execute_file` (self, str `filename`)
Run a single prompt from a file.

Static Public Member Functions

- List[Tuple[str, str, str]] `normalize_triples` (Any data)
Normalize flexible LLM output into a list of clean (subject, relation, object) triples.

Public Attributes

- `model_name`
- `temperature`
- `system_prompt`
- `llm`

6.12.1 Detailed Description

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

Note

The method [components.text_processing.LLMConnector.execute_query](#) simplifies the prompt process.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 `__init__()`

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

Initialize the connector.

Note

Model name is specified in the .env file.

6.12.3 Member Function Documentation

6.12.3.1 `check_connection()`

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

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

Parameters

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

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.12.3.2 configure()

```
configure (
    self )
```

Initialize the LangChain LLM using environment credentials.

Reads:

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

Reimplemented from [Connector](#).

6.12.3.3 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

<i>filename</i>	Path to the prompt file (.txt)
-----------------	--------------------------------

Returns

Raw LLM response as a string.

Reimplemented from [Connector](#).

6.12.3.4 execute_full_query()

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

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

6.12.3.5 execute_query()

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

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

Parameters

<i>query</i>	A single string prompt to send to the LLM.
--------------	--

Returns

Raw LLM response as a string.

Reimplemented from [Connector](#).

6.12.3.6 normalize_triples()

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

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

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

Parameters

<i>data</i>	Raw LLM output to normalize.
-------------	------------------------------

Returns

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

Exceptions

<i>ValueError</i>	If input format cannot be parsed.
-------------------	-----------------------------------

6.12.3.7 test_connection()

```
test_connection (  
    self )
```

Send a trivial prompt to verify LLM connectivity.

Returns

Whether the prompt executed successfully.

Reimplemented from [Connector](#).

6.12.4 Member Data Documentation

6.12.4.1 llm

llm

6.12.4.2 model_name

model_name

6.12.4.3 system_prompt

system_prompt

6.12.4.4 temperature

temperature

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

- [/home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py](#)

6.13 Log Class Reference

The Log class standardizes console output.

Classes

- class [Failure](#)

Static Public Member Functions

- None [success](#) (str prefix="PASS", str msg="", bool verbose=True)
A success message begins with a green prefix.
- None [warn](#) (str prefix="PASS", str msg="", bool verbose=True)
A warning message begins with a yellow prefix.
- None [fail](#) (str prefix="ERROR", str msg="", bool raise_error=True, Optional[Exception] other_error=None)
A failure message begins with a red prefix.
- None [success_legacy](#) (str msg="")
A legacy success message begins with a Green Plus.
- None [fail_legacy](#) (str msg="")
A legacy failure message begins with a Red X.

Static Public Attributes

- bool `USE_COLORS` = True
Enable ANSI colors in output.
- str `GREEN` = "\033[32m"
ANSI code for green text.
- str `RED` = "\033[31m"
ANSI code for red text.
- str `YELLOW` = "\033[33m"
ANSI code for yellow text.
- str `BRIGHT` = "\033[93m"
ANSI code for bright yellow / cream.
- str `WHITE` = "\033[0m"
ANSI code to reset color.
- str `SUCCESS_COLOR` = `GREEN`
ANSI color applied to the prefix of success messages.
- str `WARNING_COLOR` = `YELLOW`
ANSI color applied to the prefix of ignored fail messages.
- str `FAILURE_COLOR` = `RED`
ANSI color applied to the prefix of critical fail messages.
- str `MSG_COLOR` = `BRIGHT`
ANSI color applied to the body of every Log message.
- bool `FULL_DF` = False
When printing the results of a query.
- str `conn_abc` = "BASE CONNECTOR: "
- str `db_conn_abc` = "CONNECTOR: "
- str `rel_db` = "REL DB: "
- str `gr_db` = "GRAPH DB: "
- str `doc_db` = "DOCS DB: "
- str `bad_addr` = "BAD ADDRESS: "
- f `msg_bad_addr` = lambda connection_string: "Failed to connect on {connection_string}"
- str `bad_path` = "FILE NOT FOUND: "
- f `msg_bad_path` = lambda file_path: "Failed to open file '{file_path}'"
- f `msg_good_path` = lambda file_path: "Reading contents of file '{file_path}'"
- f `msg_good_exec_f` = lambda file_path: "Finished executing queries from '{file_path}'"
- f `msg_bad_exec_f` = lambda file_path: "Error occurred while executing queries from '{file_path}'"
- f `msg_db_connect` = lambda database_name: "Successfully connected to database: {database_name}"
- str `good_val` = "VALID RESULT: "
- str `bad_val` = "INCORRECT RESULT: "
- f `msg_compare` = lambda observed, expected: "Expected {expected}, got {observed}"
- tuple `msg_result`
- tuple `msg_good_table`
- tuple `msg_good_coll`
- tuple `msg_good_graph`
- f `msg_bad_table` = lambda name: "Table '{name}' not found"
- f `msg_bad_coll` = lambda name: "Collection '{name}' not found"
- f `msg_bad_graph` = lambda name: "Graph '{name}' not found"
- str `test_conn` = "CONNECTION TEST: "
- str `test_basic` = "BASIC: "
- str `test_info` = "DB INFO: "
- str `test_df` = "GET DF: "
- str `test_tmp_db` = "CREATE DB: "
- str `msg_unknown_error` = "An unhandled error occurred."

- str `get_df` = "GET_DF: "
- str `create_db` = "CREATE_DB: "
- str `drop_db` = "DROP_DB: "
- str `run_q` = "QUERY: "
- str `run_f` = "FILE EXEC: "
- str `drop_gr` = "DROP_GRAPH: "
- f `msg_success_managed_db` = lambda managed, database_name"Successfully {managed} database '{database_name}'"
- tuple `msg_fail_manage_db`
- f `msg_success_managed_gr` = lambda managed, database_name"Successfully {managed} graph '{database_name}'"
- f `msg_fail_manage_gr` = lambda manage, database_name, connection_string"Failed to {manage} graph '{database_name}' on connection {connection_string}"
- f `msg_fail_parse` = lambda alias, bad_value, expected_type"Could not convert {alias} with value {bad_value} to type {expected_type}"
- tuple `msg_multiple_query`
- f `msg_good_exec_q` = lambda query"Executed successfully:\n'{query}'"
- f `msg_good_exec_qr` = lambda query, results"Executed successfully:\n'{query}'\n{Log.msg_result(results)}"
- f `msg_bad_exec_q` = lambda query"Failed to execute query:\n'{query}'"
- str `kg` = "KG: "
- str `pytest_db` = "PYTEST (DB): "
- str `db_exists` = "DB_EXIST: "
- f `msg_db_exists` = lambda database_name"Database '{database_name}' already exists."
- f `msg_db_not_found` = lambda database_name, connection_string"Could not find database '{database_name}' using connection '{connection_string}'"
- f `msg_db_current` = lambda database_name"Cannot drop database '{database_name}' while connected to it!"
- str `swap_db` = "SWAP_DB: "
- str `swap_kg` = "SWAP_GRAPH: "
- f `msg_swap_db` = lambda old_db, new_db"Switched from database '{old_db}' to database '{new_db}'"
- f `msg_swap_kg` = lambda old_kg, new_kg"Switched from graph '{old_kg}' to graph '{new_kg}'"
- str `get_unique` = "UNIQUE: "
- str `gr_rag` = "RAG: "
- f `msg_bad_triples` = lambda graph_name"No triples found for graph {graph_name}"

6.13.1 Detailed Description

The Log class standardizes console output.

6.13.2 Member Function Documentation

6.13.2.1 fail()

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

A failure message begins with a red prefix.

Parameters

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

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True
--------------------	-------------------------------

6.13.2.2 fail_legacy()

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

A legacy failure message begins with a Red X.

Parameters

<i>msg</i>	The message to print.
------------	-----------------------

6.13.2.3 success()

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

A success message begins with a green prefix.

Parameters

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

6.13.2.4 success_legacy()

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

A legacy success message begins with a Green Plus.

Parameters

<i>msg</i>	The message to print.
------------	-----------------------

6.13.2.5 warn()

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

A warning message begins with a yellow prefix.

Parameters

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

6.13.3 Member Data Documentation**6.13.3.1 bad_addr**

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

6.13.3.2 bad_path

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

6.13.3.3 bad_val

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

6.13.3.4 BRIGHT

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

ANSI code for bright yellow / cream.

6.13.3.5 conn_abc

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


6.13.3.6 create_db

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

6.13.3.7 db_conn_abc

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

6.13.3.8 db_exists

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

6.13.3.9 doc_db

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

6.13.3.10 drop_db

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

6.13.3.11 drop_gr

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

6.13.3.12 FAILURE_COLOR

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

ANSI color applied to the prefix of critical fail messages.

6.13.3.13 FULL_DF

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

When printing the results of a query.

6.13.3.14 get_df

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

6.13.3.15 get_unique

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

6.13.3.16 good_val

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

6.13.3.17 gr_db

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

6.13.3.18 gr_rag

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

6.13.3.19 GREEN

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

ANSI code for green text.

6.13.3.20 kg

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

6.13.3.21 msg_bad_addr

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

6.13.3.22 msg_bad_coll

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

6.13.3.23 msg_bad_exec_f

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

6.13.3.24 msg_bad_exec_q

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

6.13.3.25 msg_bad_graph

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

6.13.3.26 msg_bad_path

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

6.13.3.27 msg_bad_table

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

6.13.3.28 msg_bad_triples

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

6.13.3.29 MSG_COLOR

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

ANSI color applied to the body of every Log message.

6.13.3.30 msg_compare

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

6.13.3.31 msg_db_connect

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

6.13.3.32 msg_db_current

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

6.13.3.33 msg_db_exists

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

6.13.3.34 msg_db_not_found

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

6.13.3.35 msg_fail_manage_db

```
tuple msg_fail_manage_db [static]
```

Initial value:

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

6.13.3.36 msg_fail_manage_gr

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

6.13.3.37 msg_fail_parse

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

6.13.3.38 msg_good_coll

```
tuple msg_good_coll [static]
```

Initial value:

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

6.13.3.39 msg_good_exec_f

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

6.13.3.40 msg_good_exec_q

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

6.13.3.41 msg_good_exec_qr

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

6.13.3.42 msg_good_graph

```
tuple msg_good_graph [static]
```

Initial value:

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

6.13.3.43 msg_good_path

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

6.13.3.44 msg_good_table

```
tuple msg_good_table [static]
```

Initial value:

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

6.13.3.45 msg_multiple_query

```
tuple msg_multiple_query [static]
```

Initial value:

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

6.13.3.46 msg_result

```
tuple msg_result [static]
```

Initial value:

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

6.13.3.47 msg_success_managed_db

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

6.13.3.48 msg_success_managed_gr

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

6.13.3.49 msg_swap_db

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

6.13.3.50 msg_swap_kg

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

6.13.3.51 msg_unknown_error

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

6.13.3.52 pytest_db

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

6.13.3.53 RED

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

ANSI code for red text.

6.13.3.54 rel_db

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

6.13.3.55 run_f

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

6.13.3.56 run_q

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

6.13.3.57 SUCCESS_COLOR

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

ANSI color applied to the prefix of success messages.

6.13.3.58 swap_db

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

6.13.3.59 swap_kg

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

6.13.3.60 test_basic

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

6.13.3.61 test_conn

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

6.13.3.62 test_df

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

6.13.3.63 test_info

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

6.13.3.64 test_tmp_db

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

6.13.3.65 USE_COLORS

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

Enable ANSI colors in output.

6.13.3.66 WARNING_COLOR

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

ANSI color applied to the prefix of ignored fail messages.

6.13.3.67 WHITE

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

ANSI code to reset color.

6.13.3.68 YELLOW

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

ANSI code for yellow text.

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

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

6.14 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 data 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 boook_score=0.0, float questeval_score=0.0, str qa_question1="UNKNOWN", str qa_gold1="UNKNOWN", str qa_generated1="UNKNOWN", bool qa_correct1=False, float qa_accuracy1=0.0, str qa_question2="UNKNOWN", str qa_gold2="UNKNOWN", str qa_generated2="UNKNOWN", bool qa_correct2=False, float qa_accuracy2=0.0)
Generate the metrics sub-payload with customizable default values.
- Dict[str, Any] `generate_example_metrics` ()
Create a placeholder payload with dummy values.

Public Attributes

- [HOST](#)
- [PORT](#)
- [url](#)

Static Public Attributes

- `int timeout_seconds = 900`

6.14.1 Detailed Description

Utility class for computing and posting evaluation metrics.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 `__init__()`

```
None __init__ (
    self )
```

6.14.3 Member Function Documentation

6.14.3.1 `compute_basic_metrics()`

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

Compute ROUGE and BERTScore.

Parameters

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

Returns

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

6.14.3.2 `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

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

Returns

A dictionary with C#-style key names.

6.14.3.3 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 boook_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

<i>rouge1_f1</i>	The ROUGE-1 evaluation metric.
<i>rouge2_f1</i>	The ROUGE-2 evaluation metric.
<i>rougeL_f1</i>	The ROUGE-L evaluation metric.
<i>rougeLsum_f1</i>	The ROUGE-Lsum evaluation metric.
<i>bert_precision</i>	The BERTScore precision score.
<i>bert_recall</i>	The BERTScore recall score.
<i>bert_f1</i>	The BERTScore F1 score.

Parameters

<i>boook_score</i>	The BoookScore evaluation metric.
<i>questeval_score</i>	The QuestEval evaluation metric.
<i>qa_question1</i>	A question about the book.
<i>qa_gold1</i>	The correct answer to the question.
<i>qa_generated1</i>	A generated answer to judge.
<i>qa_correct1</i>	Whether our answer is correct.
<i>qa_accuracy1</i>	The accuracy score for this QA sample.
<i>qa_question2</i>	A question about the book.
<i>qa_gold2</i>	The correct answer to the question.
<i>qa_generated2</i>	A generated answer to judge.
<i>qa_correct2</i>	Whether our answer is correct.
<i>qa_accuracy2</i>	The accuracy score for this QA sample.

Returns

Dictionary containing various nested evaluation metrics.

6.14.3.4 generate_example_metrics()

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

Create a placeholder payload with dummy values.

Returns

Full payload with nested metrics.

6.14.3.5 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

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

6.14.3.6 post_basic_output()

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

POST dummy data to Blazor.

Parameters

<i>book_id</i>	Unique identifier for one book.
<i>book_title</i>	String containing the title of a book.
<i>summary</i>	String containing a book summary.

6.14.3.7 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

<i>payload</i>	JSON dictionary containing data for a single book.
----------------	--

Returns

Whether the POST operation was successful.

6.14.4 Member Data Documentation

6.14.4.1 HOST

HOST

6.14.4.2 PORT

PORT

6.14.4.3 timeout_seconds

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

6.14.4.4 url

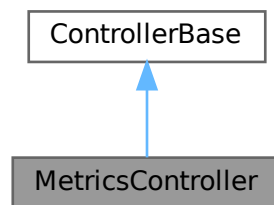
url

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

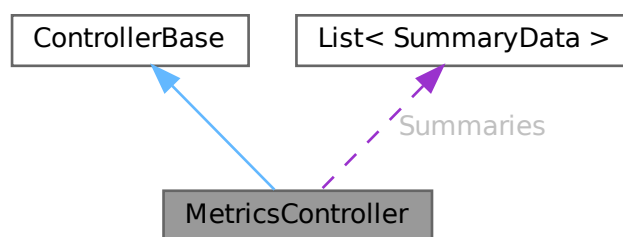
- [/home/runner/work/dsci-capstone/dsci-capstone/components/metrics.py](#)

6.15 MetricsController Class Reference

Inheritance diagram for MetricsController:



Collaboration diagram for MetricsController:



Public Member Functions

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

Private Attributes

- readonly ILogger< [MetricsController](#) > [_logger](#)
- readonly IHubContext< [MetricsHub](#) > [_hubContext](#)

Static Private Attributes

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

6.15.1 Constructor & Destructor Documentation

6.15.1.1 MetricsController()

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

6.15.2 Member Function Documentation

6.15.2.1 GetAll()

```
IActionResult GetAll ( )
```

6.15.2.2 GetIndex()

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

6.15.2.3 Post()

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

6.15.3 Member Data Documentation

6.15.3.1 _hubContext

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

6.15.3.2 _logger

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

6.15.3.3 Summaries

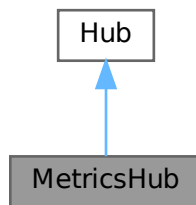
```
readonly List<SummaryData> Summaries = new() [static], [private]
```

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

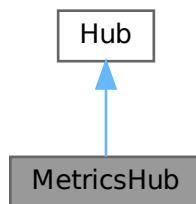
- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/[MetricsController.cs](#)

6.16 MetricsHub Class Reference

Inheritance diagram for MetricsHub:



Collaboration diagram for MetricsHub:



Public Member Functions

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

Private Attributes

- readonly? ILogger< [MetricsHub](#) > [_logger](#)

6.16.1 Constructor & Destructor Documentation

6.16.1.1 MetricsHub()

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

6.16.2 Member Function Documentation

6.16.2.1 OnConnectedAsync()

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

6.16.2.2 OnDisconnectedAsync()

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

6.16.3 Member Data Documentation

6.16.3.1 _logger

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

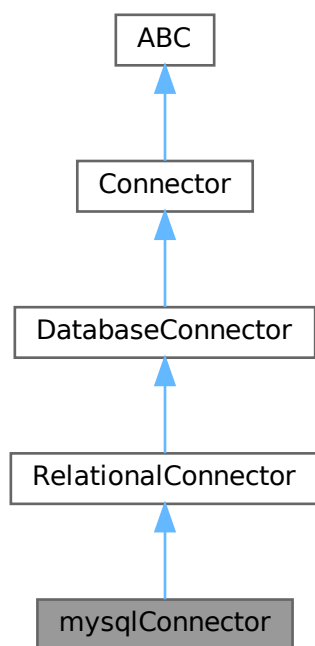
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Hubs/[MetricsHub.cs](#)

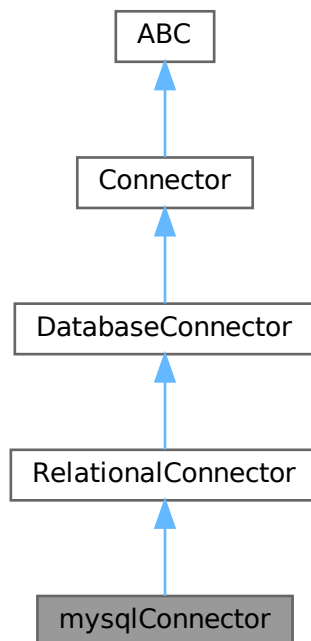
6.17 mysqlConnector Class Reference

A relational database connector configured for MySQL.

Inheritance diagram for mysqlConnector:



Collaboration diagram for mysqlConnector:



Public Member Functions

- None `__init__` (self, bool `verbose=False`)
Configures the relational connector.

Public Member Functions inherited from `RelationalConnector`

- "RelationalConnector" `from_env` (cls, bool `verbose=False`)
Decides what type of relational connector to create using the .env file.
- None `change_database` (self, str `new_database`)
Update the connection URI to reference a different database in the same engine.
- bool `test_connection` (self, bool `raise_error=True`)
Establish a basic connection to the database, and test full functionality.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] `execute_query` (self, str `query`)
Send a single command to the database connection.
- Optional[DataFrame] `get_dataframe` (self, str `name`, List[str] `columns=[]`)
Automatically generate and run a query for the specified table using SQLAlchemy.
- None `create_database` (self, str `database_name`)
Use the current database connection to create a sibling database in this engine.
- None `drop_database` (self, str `database_name=""`)
Delete all data stored in a particular database.
- bool `database_exists` (self, str `database_name`)
Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None [configure](#) (self, str DB, str database_name)
Read connection settings from the .env file.
- Generator[None, None, None] [temp_database](#) (self, str database_name)
Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] [execute_combined](#) (self, str multi_query)
Run several database commands in sequence.
- List[Optional[DataFrame]] [execute_file](#) (self, str filename)
Run several database commands from a file.

Static Public Attributes

- dict [specific_queries](#)

Additional Inherited Members

Public Attributes inherited from RelationalConnector

- [database_name](#)
- [verbose](#)
- [connection_string](#)
- [db_type](#)

Public Attributes inherited from DatabaseConnector

- [verbose](#)
Whether to print debug messages.
- [db_type](#)
- [db_engine](#)
- [username](#)
- [password](#)
- [host](#)
- [port](#)
- [connection_string](#)

Protected Member Functions inherited from RelationalConnector

- List[str] [_split_combined](#) (self, str multi_query)
Divides a string into non-divisible SQL queries using `sqlparse`.
- 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 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.

6.17.1 Detailed Description

A relational database connector configured for MySQL.

Note

Should be hidden from the user using a factory method.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 `__init__()`

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

Configures the relational connector.

Parameters

<i>verbose</i>	Whether to print success and failure messages.
----------------	--

Reimplemented from [RelationalConnector](#).

6.17.3 Member Data Documentation

6.17.3.1 `specific_queries`

```
dict specific_queries [static]
```

Initial value:

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

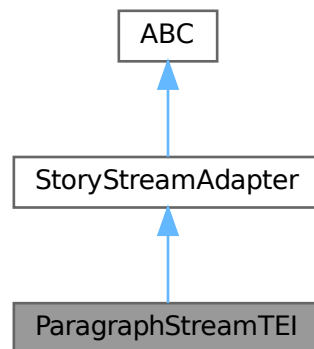
The documentation for this class was generated from the following file:

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

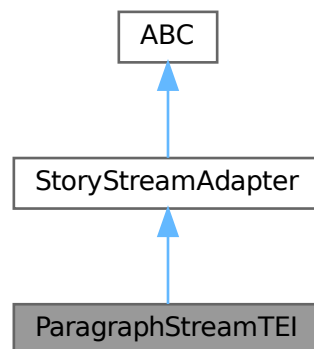
6.18 ParagraphStreamTEI Class Reference

Streams paragraphs from a TEI file as Chunk objects.

Inheritance diagram for ParagraphStreamTEI:



Collaboration diagram for ParagraphStreamTEI:



Public Member Functions

- None `__init__` (self, str `tei_path`, int `book_id`, int `story_id`, list[str] `allowed_chapters`=None, str `start_inclusive`="", str `end_inclusive`="")
Create a ParagraphStreamTEI object.
- Iterator[Chunk] `stream_segments` (self)
Yields sanitized parts of a book.
- List[Chunk] `pre_compute_segments` (self)
Splits the target book into paragraphs.

Public Member Functions inherited from [StoryStreamAdapter](#)

- [Iterator\[Chunk\]](#) [stream_paragraphs](#) (self)
Concrete helper method to split segments into paragraphs.
- [Iterator\[str\]](#) [stream_sentences](#) (self)
Concrete helper method to split paragraphs into sentences.

Public Attributes

- [tei_path](#)
- [book_id](#)
- [story_id](#)
- [allowed_chapters](#)
- [start_inclusive](#)
- [end_inclusive](#)
- [lines](#)
- [root](#)
- [chunks](#)
- [xml_namespace](#)

Static Public Attributes

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

6.18.1 Detailed Description

Streams paragraphs from a TEI file as Chunk objects.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 `__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

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

6.18.3 Member Function Documentation

6.18.3.1 pre_compute_segments()

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

Splits the target book into paragraphs.

Yields Chunk objects for each paragraph (

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

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

6.18.3.2 stream_segments()

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

Yields sanitized parts of a book.

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

Reimplemented from [StoryStreamAdapter](#).

6.18.4 Member Data Documentation

6.18.4.1 allowed_chapters

```
allowed_chapters
```

6.18.4.2 book_id

```
book_id
```

6.18.4.3 chunks

```
chunks
```

6.18.4.4 encoding

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

6.18.4.5 end_inclusive

```
end_inclusive
```

6.18.4.6 lines

```
lines
```

6.18.4.7 root

```
root
```

6.18.4.8 start_inclusive

```
start_inclusive
```

6.18.4.9 story_id

```
story_id
```

6.18.4.10 tei_path

```
tei_path
```

6.18.4.11 xml_namespace [1/2]

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

6.18.4.12 xml_namespace [2/2]

```
xml_namespace
```

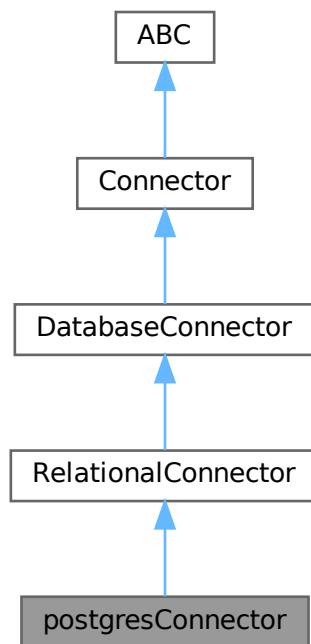
The documentation for this class was generated from the following file:

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

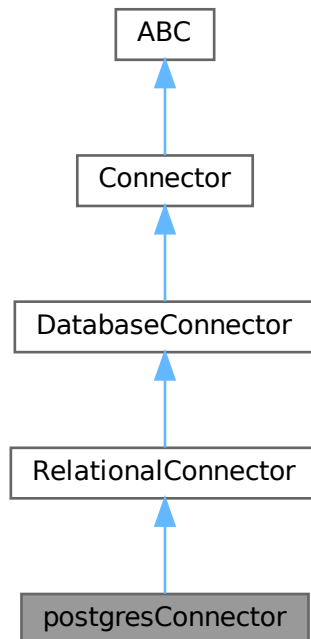
6.19 postgresConnector Class Reference

A relational database connector configured for PostgreSQL.

Inheritance diagram for postgresConnector:



Collaboration diagram for postgresConnector:



Public Member Functions

- None `__init__` (self, bool `verbose=False`)
Configures the relational connector.

Public Member Functions inherited from `RelationalConnector`

- "RelationalConnector" `from_env` (cls, bool `verbose=False`)
Decides what type of relational connector to create using the .env file.
- None `change_database` (self, str `new_database`)
Update the connection URI to reference a different database in the same engine.
- bool `test_connection` (self, bool `raise_error=True`)
Establish a basic connection to the database, and test full functionality.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] `execute_query` (self, str `query`)
Send a single command to the database connection.
- Optional[DataFrame] `get_dataframe` (self, str `name`, List[str] `columns=[]`)
Automatically generate and run a query for the specified table using SQLAlchemy.
- None `create_database` (self, str `database_name`)
Use the current database connection to create a sibling database in this engine.
- None `drop_database` (self, str `database_name=""`)
Delete all data stored in a particular database.
- bool `database_exists` (self, str `database_name`)
Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None [configure](#) (self, str DB, str database_name)
Read connection settings from the .env file.
- Generator[None, None, None] [temp_database](#) (self, str database_name)
Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] [execute_combined](#) (self, str multi_query)
Run several database commands in sequence.
- List[Optional[DataFrame]] [execute_file](#) (self, str filename)
Run several database commands from a file.

Static Public Attributes

- dict [specific_queries](#)

Additional Inherited Members

Public Attributes inherited from RelationalConnector

- [database_name](#)
- [verbose](#)
- [connection_string](#)
- [db_type](#)

Public Attributes inherited from DatabaseConnector

- [verbose](#)
Whether to print debug messages.
- [db_type](#)
- [db_engine](#)
- [username](#)
- [password](#)
- [host](#)
- [port](#)
- [connection_string](#)

Protected Member Functions inherited from RelationalConnector

- List[str] [_split_combined](#) (self, str multi_query)
Divides a string into non-divisible SQL queries using `sqlparse`.
- 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 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.

6.19.1 Detailed Description

A relational database connector configured for PostgreSQL.

Note

Should be hidden from the user using a factory method.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 `__init__()`

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

Configures the relational connector.

Parameters

<i>verbose</i>	Whether to print success and failure messages.
----------------	--

Reimplemented from [RelationalConnector](#).

6.19.3 Member Data Documentation

6.19.3.1 `specific_queries`

```
dict specific_queries [static]
```

Initial value:

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

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

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

6.20 PRF1Metric Class Reference

Properties

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

6.20.1 Property Documentation

6.20.1.1 F1Score

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

6.20.1.2 Name

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

6.20.1.3 Precision

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

6.20.1.4 Recall

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

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

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/[PRF1Metric.cs](#)

6.21 QALtem Class Reference

Properties

- string [Question](#) [get, set]
- string [GoldAnswer](#) [get, set]
- string [GeneratedAnswer](#) [get, set]
- bool? [IsCorrect](#) [get, set]
- double? [Accuracy](#) [get, set]

6.21.1 Property Documentation

6.21.1.1 Accuracy

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

6.21.1.2 GeneratedAnswer

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

6.21.1.3 GoldAnswer

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

6.21.1.4 IsCorrect

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

6.21.1.5 Question

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

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

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/[QAItem.cs](#)

6.22 QAMetric Class Reference

Properties

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

6.22.1 Property Documentation

6.22.1.1 AverageAccuracy

```
double AverageAccuracy [get]
```

6.22.1.2 QAItems

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

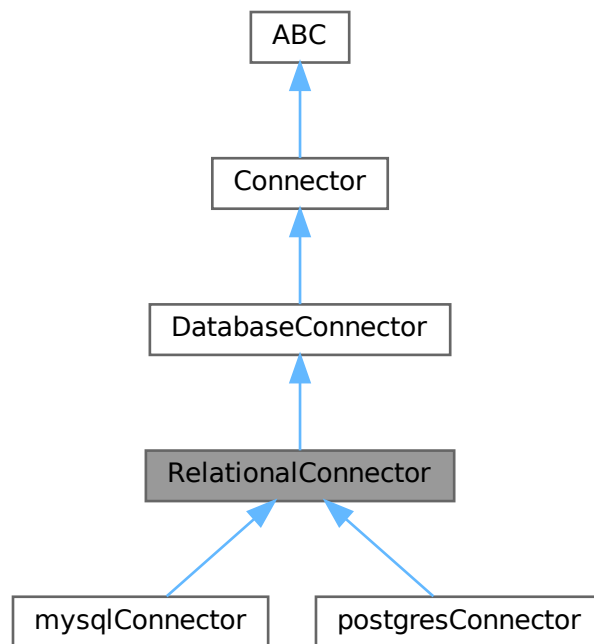
The documentation for this class was generated from the following file:

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/[QAMetric.cs](#)

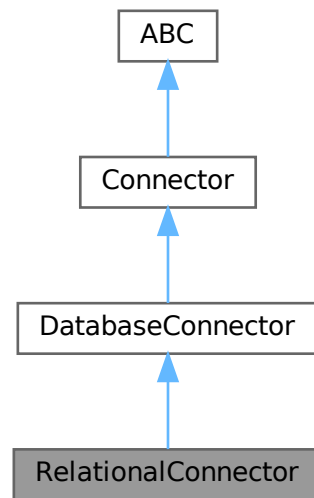
6.23 RelationalConnector Class Reference

Connector for relational databases (MySQL, PostgreSQL).

Inheritance diagram for RelationalConnector:



Collaboration diagram for RelationalConnector:



Public Member Functions

- None `__init__` (self, bool `verbose`, List[str] `specific_queries`)
Creates a new database connector.
- "RelationalConnector" `from_env` (cls, bool `verbose=False`)
Decides what type of relational connector to create using the .env file.
- None `change_database` (self, str `new_database`)
Update the connection URI to reference a different database in the same engine.
- bool `test_connection` (self, bool `raise_error=True`)
Establish a basic connection to the database, and test full functionality.
- bool `check_connection` (self, str `log_source`, bool `raise_error`)
Minimal connection test to determine if our connection string is valid.
- Optional[DataFrame] `execute_query` (self, str `query`)
Send a single command to the database connection.
- Optional[DataFrame] `get_dataframe` (self, str `name`, List[str] `columns=[]`)
Automatically generate and run a query for the specified table using SQLAlchemy.
- None `create_database` (self, str `database_name`)
Use the current database connection to create a sibling database in this engine.
- None `drop_database` (self, str `database_name=""`)
Delete all data stored in a particular database.
- bool `database_exists` (self, str `database_name`)
Search for an existing database using the provided name.

Public Member Functions inherited from DatabaseConnector

- None [configure](#) (self, str DB, str database_name)
Read connection settings from the .env file.
- Generator[None, None, None] [temp_database](#) (self, str database_name)
Temporarily switch to a pseudo-database, creating and dropping it if needed.
- List[Optional[DataFrame]] [execute_combined](#) (self, str multi_query)
Run several database commands in sequence.
- List[Optional[DataFrame]] [execute_file](#) (self, str filename)
Run several database commands from a file.

Public Attributes

- [database_name](#)
- [verbose](#)
- [connection_string](#)
- [db_type](#)

Public Attributes inherited from DatabaseConnector

- [verbose](#)
Whether to print debug messages.
- [db_type](#)
- [db_engine](#)
- [username](#)
- [password](#)
- [host](#)
- [port](#)
- [connection_string](#)

Protected Member Functions

- List[str] [_split_combined](#) (self, str multi_query)
Divides a string into non-divisible SQL queries using `sqlparse`.
- 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 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.

6.23.1 Detailed Description

Connector for relational databases (MySQL, PostgreSQL).

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

6.23.2 Constructor & Destructor Documentation

6.23.2.1 `__init__()`

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

Creates a new database connector.

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

Parameters

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

Reimplemented from [DatabaseConnector](#).

Reimplemented in [mysqlConnector](#), and [postgresConnector](#).

6.23.3 Member Function Documentation

6.23.3.1 `_parsable_to_df()`

```
bool _parsable_to_df (
    self,
    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()`.
- DDL/DML (CREATE, INSERT, UPDATE, etc.) produce no rows and only rowcount/status.

Parameters

<i>result</i>	The result of a SQL, Cypher, or JSON query.
---------------	---

Returns

Whether the object is parsable to DataFrame.

Reimplemented from [DatabaseConnector](#).

6.23.3.2 `_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

<i>query</i>	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](#).

6.23.3.3 _split_combined()

```

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

```

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

Parameters

<i>multi_query</i>	A string containing multiple queries.
--------------------	---------------------------------------

Returns

A list of single-query strings.

Reimplemented from [DatabaseConnector](#).

6.23.3.4 change_database()

```

None change_database (
    self,
    str new_database )

```

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

Parameters

<i>new_database</i>	The name of the database to connect to.
---------------------	---

Reimplemented from [DatabaseConnector](#).

6.23.3.5 check_connection()

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

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

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

Parameters

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

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If raise_error is True and the connection test fails to complete.
--------------------	---

Reimplemented from [Connector](#).

6.23.3.6 create_database()

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

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

Parameters

<i>database_name</i>	The name of the new database to create.
----------------------	---

Exceptions

<i>Log.Failure</i>	If we fail to create the requested database for any reason.
--------------------	---

Reimplemented from [DatabaseConnector](#).

6.23.3.7 database_exists()

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

Search for an existing database using the provided name.

Parameters

<i>database_name</i>	The name of a database to search for.
----------------------	---------------------------------------

Returns

Whether the database is visible to this connector.

Reimplemented from [DatabaseConnector](#).

6.23.3.8 drop_database()

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

Delete all data stored in a particular database.

Parameters

<i>database_name</i>	The name of an existing database.
----------------------	-----------------------------------

Exceptions

<i>Log.Failure</i>	If we fail to drop the target database for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.23.3.9 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

<i>query</i>	A single query to perform on the database.
--------------	--

Returns

DataFrame containing the result of the query, or None

Exceptions

<i>Log.Failure</i>	If the query fails to execute.
--------------------	--------------------------------

Reimplemented from [DatabaseConnector](#).

6.23.3.10 from_env()

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

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

Parameters

<i>verbose</i>	Whether to print success and failure messages.
----------------	--

Exceptions

<i>Log.Failure</i>	If the .env file contains an invalid DB_ENGINE value.
--------------------	---

6.23.3.11 get_dataframe()

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

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

Parameters

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

Returns

Sorted DataFrame containing the requested data, or None

Exceptions

<i>Log.Failure</i>	If we fail to create the requested DataFrame for any reason.
--------------------	--

Reimplemented from [DatabaseConnector](#).

6.23.3.12 test_connection()

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

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

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

Parameters

<i>raise_error</i>	Whether to raise an error on connection failure.
--------------------	--

Returns

Whether the connection test was successful.

Exceptions

<i>Log.Failure</i>	If <i>raise_error</i> is True and the connection test fails to complete.
--------------------	--

Reimplemented from [Connector](#).

6.23.4 Member Data Documentation**6.23.4.1 connection_string**

`connection_string`

6.23.4.2 database_name

`database_name`

6.23.4.3 db_type

`db_type`

6.23.4.4 verbose

verbose

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

- [/home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py](#)

6.24 RelationExtractor Class Reference

Public Member Functions

- [__init__](#) (self, model_name="Babelscape/rebel-large", [max_tokens](#)=1024)
- [extract](#) (self, str text, bool parse_tuples=False)

Public Attributes

- [tokenizer](#)
- [model](#)
- [max_tokens](#)
- [tuple_delim](#)

6.24.1 Constructor & Destructor Documentation

6.24.1.1 __init__()

```
__init__ (
    self,
    model_name = "Babelscape/rebel-large",
    max_tokens = 1024 )
```

6.24.2 Member Function Documentation

6.24.2.1 extract()

```
extract (
    self,
    str text,
    bool parse_tuples = False )
```

6.24.3 Member Data Documentation

6.24.3.1 max_tokens

max_tokens

6.24.3.2 model

```
model
```

6.24.3.3 tokenizer

```
tokenizer
```

6.24.3.4 tuple_delim

```
tuple_delim
```

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

- [/home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py](#)

6.25 ScalarMetric Class Reference

Properties

- string [Name](#) [get, set]
- double [Value](#) [get, set]

6.25.1 Property Documentation

6.25.1.1 Name

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

6.25.1.2 Value

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

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

- [/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs](#)

6.26 Session Class Reference

Stores active database connections and configuration settings.

Public Member Functions

- `__new__` (cls, *args, **kwargs)
Creates a new session at first access, otherwise uses the existing session.
- `__init__` (self, `verbose=False`)
Initializes the session using the .env file.
- `test_database_connections` (self)
Configure the databases and verify they are working correctly.
- `reset` (self)
Deletes all created databases and tables.

Public Attributes

- `verbose`
Initializes the session using the .env file.
- `relational_db`
Stores RDF-compliant semantic triples.
- `docs_db`
Stores input text, pre-processed chunks, JSON intermediates, and final output.
- `graph_db`
Stores entities (nodes) and relations (edges).
- `main_graph`
Main storage for initial pipeline.

Static Protected Attributes

- `__instance` = None
Creates a new session at first access, otherwise uses the existing session.

6.26.1 Detailed Description

Stores active database connections and configuration settings.

- This class implements Singleton design, so only one session can be created.
- However, the session config can still be updated using the normal constructor.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 `__init__()`

```
__init__ (
    self,
    verbose = False )
```

Initializes the session using the .env file.

- The relational database connector is created using a Factory Method, choosing mysql or postgres based on the .env file.
- The document database connector is created normally since mongo is the only supported option.
- The graph database connector is created normally since neo4j is the only supported option.

6.26.3 Member Function Documentation

6.26.3.1 `__new__()`

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

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

6.26.3.2 `reset()`

```
reset (
    self )
```

Deletes all created databases and tables.

6.26.3.3 `test_database_connections()`

```
test_database_connections (
    self )
```

Configure the databases and verify they are working correctly.

6.26.4 Member Data Documentation

6.26.4.1 `_instance`

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

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

6.26.4.2 `docs_db`

```
docs_db
```

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

6.26.4.3 `graph_db`

```
graph_db
```

Stores entities (nodes) and relations (edges).

6.26.4.4 main_graph

`main_graph`

Main storage for initial pipeline.

6.26.4.5 relational_db

`relational_db`

Stores RDF-compliant semantic triples.

6.26.4.6 verbose

`verbose`

Initializes the session using the `.env` file.

- The relational database connector is created using a Factory Method, choosing mysql or postgres based on the `.env` file.
 - The document database connector is created normally since mongo is the only supported option.
 - The graph database connector is created normally since neo4j is the only supported option.

Enables or disables the components from printing debug info.

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

- `/home/runner/work/dsci-capstone/dsci-capstone/src/setup.py`

6.27 Story Class Reference

Public Member Functions

- None `__init__` (self, [StoryStreamAdapter](#) reader)
- Iterator[[Chunk](#)] `stream_chunks` (self)
- None `pre_split_chunks` (self, int max_chunk_length)
Splits paragraphs into chunks.

Public Attributes

- [reader](#)

Protected Member Functions

- None `_merge_chunks` (self, List[[Chunk](#)] segs, int max_len)
- [Chunk](#) `_make_single` (self, [Chunk](#) seg, str text, int max_len, Optional[[Chunk](#)] start=None)

6.27.1 Constructor & Destructor Documentation

6.27.1.1 `__init__()`

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

6.27.2 Member Function Documentation

6.27.2.1 `_make_single()`

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

6.27.2.2 `_merge_chunks()`

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

6.27.2.3 `pre_split_chunks()`

```
None pre_split_chunks (
    self,
    int max_chunk_length )
```

Splits paragraphs into chunks.

- Populates `self.chunks` with `Chunk` objects that obey `max_chunk_length`.
- Combines adjacent paragraphs when possible.
- Falls back to splitting by sentences if one paragraph is too long.

6.27.2.4 `stream_chunks()`

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

6.27.3 Member Data Documentation

6.27.3.1 reader

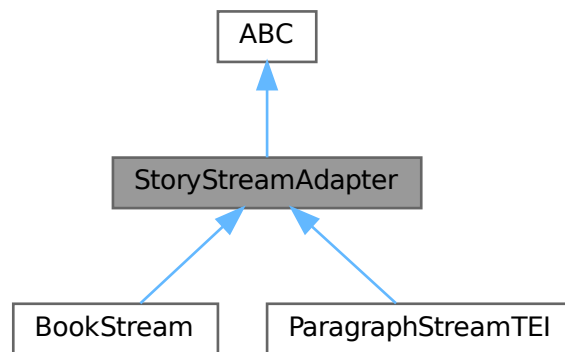
reader

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

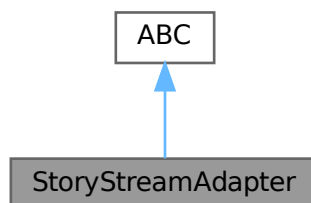
- /home/runner/work/dsci-capstone/dsci-capstone/components/book_conversion.py

6.28 StoryStreamAdapter Class Reference

Inheritance diagram for StoryStreamAdapter:



Collaboration diagram for StoryStreamAdapter:



Public Member Functions

- `Iterator[Chunk] stream_segments (self)`
Yields sanitized parts of a book.
- `Iterator[Chunk] stream_paragraphs (self)`
Concrete helper method to split segments into paragraphs.
- `Iterator[str] stream_sentences (self)`
Concrete helper method to split paragraphs into sentences.

6.28.1 Member Function Documentation

6.28.1.1 stream_paragraphs()

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

Concrete helper method to split segments into paragraphs.

The Chunk class is repurposed here so we pass location info. Depending on the `Story.pre_split_chunks` implementation, this might be unnecessary.

6.28.1.2 stream_segments()

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

Yields sanitized parts of a book.

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

Reimplemented in [ParagraphStreamTEI](#), and [BookStream](#).

6.28.1.3 stream_sentences()

```
Iterator[str] stream_sentences (  
    self )
```

Concrete helper method to split paragraphs into sentences.

Mostly for debugging.

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

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

6.29 SummaryData Class Reference

Properties

- string [BookID](#) [get, set]
- string [BookTitle](#) [get, set]
- string [SummaryText](#) [get, set]
- string [GoldSummaryText](#) [get, set]
- [SummaryMetrics Metrics](#) = new() [get, set]
- List<[QAMetric](#)> [QAResults](#) = new() [get, set]

6.29.1 Property Documentation

6.29.1.1 BookID

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

6.29.1.2 BookTitle

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

6.29.1.3 GoldSummaryText

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

6.29.1.4 Metrics

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

6.29.1.5 QAResults

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

6.29.1.6 SummaryText

```
string SummaryText [get], [set]
```

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

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/[SummaryData.cs](#)

6.30 SummaryMetrics Class Reference

Static Public Member Functions

- static [SummaryMetrics](#) [GetDefault](#) ()

Properties

- List< [PRF1Metric](#) > [PRF1Metrics](#) = new() [get, set]
- [QAMetric](#) [QA](#) = new() [get, set]
- List< [ScalarMetric](#) > [ScalarMetrics](#) = new() [get, set]

6.30.1 Member Function Documentation

6.30.1.1 GetDefault()

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

6.30.2 Property Documentation

6.30.2.1 PRF1Metrics

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

6.30.2.2 QA

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

6.30.2.3 ScalarMetrics

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

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

- /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/[SummaryMetrics.cs](#)

Chapter 7

File Documentation

7.1 `/home/runner/work/dsci-capstone/dsci-capstone/components/book↔ _conversion.py` File Reference

Classes

- class `Chunk`
Lightweight container for a span of story text.
- class `StoryStreamAdapter`
- class `Story`
- class `ParagraphStreamTEI`
Streams paragraphs from a TEI file as Chunk objects.
- class `Book`
- class `BookStream`
- class `BookFactory`
- class `EPUBToTEI`
Converts EPUB files to XML format (TEI specification).

Namespaces

- namespace `components`
- namespace `components.book_conversion`

Variables

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

7.2 /home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py File Reference

Classes

- class [Connector](#)
Abstract base class for external connectors.
- class [DatabaseConnector](#)
Abstract base class for database engine connectors.
- class [RelationalConnector](#)
Connector for relational databases (MySQL, PostgreSQL).
- class [mysqlConnector](#)
A relational database connector configured for MySQL.
- class [postgresConnector](#)
A relational database connector configured for PostgreSQL.

Namespaces

- namespace [components](#)
- namespace [components.connectors](#)

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

Namespaces

- namespace [components](#)
- namespace [components.corpus](#)

Functions

- [load_booksum](#) ()
- [to_df_booksum](#) (ds)
- [load_narrativeqa](#) ()
- [to_df_nqa](#) (ds)
- [normalize_title](#) (t)
- [merge_dataframes](#) (df1, df2, suffix1, suffix2, key_columns)
- [fuzzy_merge_titles](#) (df1, df2, suffix1, suffix2, key="title", threshold=90, scorer=fuzz.token_sort_ratio)
Perform a two-way fuzzy merge between two DataFrames on a text column (e.g., book titles).

Variables

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

7.4 /home/runner/work/dsci-capstone/dsci-capstone/components/document_storage.py File Reference

Classes

- class [DocumentConnector](#)
Connector for MongoDB (document database)

Namespaces

- namespace [components](#)
- namespace [components.document_storage](#)

Functions

- [MongoHandle mongo_handle](#) (str host, str alias)
Establish a temporary connection to MongoDB.
- [DataFrame _flatten_recursive](#) (DataFrame df)
Explode all list columns and flatten dict columns until only scalars remain.
- [str _sanitize_json](#) (str text)
Remove comments and other non-JSON content from a MongoDB query string.
- [Dict\[str, Any\] _sanitize_document](#) (Dict[str, Any] doc, Dict[str, Set[Type[Any]]] type_registry)
Normalize document fields to consistent types for DataFrame construction.
- [DataFrame _docs_to_df](#) (List[Dict[str, Any]] docs, bool merge_unspecified=True)
Convert raw MongoDB documents to a Pandas DataFrame.
- [str _find_compatible_nested_key](#) (Type[Any] value_type, Dict[str, Set[Type[Any]]] nested_schema, bool merge_unspecified)
Find a nested column compatible with the given primitive type.

Variables

- [MongoHandle](#) = Generator["Database[Any]", None, None]

7.5 /home/runner/work/dsci-capstone/dsci-capstone/components/fact_storage.py File Reference ↩

Classes

- class [GraphConnector](#)
Connector for Neo4j (graph database).

Namespaces

- namespace [components](#)
- namespace [components.fact_storage](#)

Functions

- 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.

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

Classes

- class [Metrics](#)
Utility class for computing and posting evaluation metrics.

Namespaces

- namespace [components](#)
- namespace [components.metrics](#)

Functions

- Dict[str, Any] [run_questeval](#) (Dict[str, Any] chunk, *str qeval_task="summarization", bool use_cuda=False, bool use_question_weighter=True)
Run QuestEval metric calculation.
- Dict[str, Any] [run_bookscore](#) (Dict[str, Any] chunk, *str model="gpt-3.5-turbo", int batch_size=10, bool use_v2=True)
Run BoookScore metric for long-form summarization.
- str [chunk_bookscore](#) (str book_text, str book_title='book', int chunk_size=2048)
Chunk a book into BoookScore segments.

7.7 /home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py File Reference

Classes

- class [KnowledgeGraph](#)
Manages a single graph within Neo4j.

Namespaces

- namespace [components](#)
- namespace [components.semantic_web](#)

7.8 /home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py File Reference

Classes

- class [RelationExtractor](#)
- class [LLMConnector](#)

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

Namespaces

- namespace [components](#)
- namespace [components.text_processing](#)

Variables

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

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

Namespaces

- namespace [components](#)

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

Namespaces

- namespace [src](#)

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

Namespaces

- namespace [tests](#)

7.12 /home/runner/work/dsci-capstone/dsci-capstone/src/flask.py File Reference

Namespaces

- namespace [src](#)
- namespace [src.flask](#)
Generic Flask worker microservice for distributed task processing.

Functions

- [process_task](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] chunk_doc, str [boss_url](#), Callable[[Dict[str, Any]], Dict[str, Any]] task_handler, Any task_kwargs=None)
Perform the assigned task in a background thread.
- str [load_mongo_config](#) (str database)
Load MongoDB configuration from environment variables.
- str [load_boss_config](#) ()
Load boss service callback URL from environment variables.
- Tuple[Callable[[Dict[str, Any]], Dict[str, Any]], Dict[str, Any]] [get_task_info](#) (str task_name)
Dynamically import and return the appropriate task handler function.
- [load_imports](#) (func)
Pre-warm the task by importing requirements.
- None [mark_task_in_progress](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name)
Mark a task as in-progress in MongoDB before processing begins.
- None [save_task_result](#) ([MongoHandle](#) mongo_db, str collection_name, str chunk_id, str task_name, Dict[str, Any] result)
Save completed task results to MongoDB.
- None [notify_boss](#) (str [boss_url](#), str chunk_id, str task_name, str status)
Send completion notification to boss service.
- Flask [create_app](#) (str task_name, str [boss_url](#))
Create and configure Flask application for task processing.

Variables

- [MongoHandle](#) = Generator["Database[Any]", None, None]
- [parser](#) = argparse.ArgumentParser(description="Flask worker microservice")
- [required](#)
- [True](#)
- [help](#)
- [args](#) = parser.parse_args()
- str [task_queue](#) = Queue()
- [target](#)
- [task_worker](#) ()
Background threading system for non-blocking task handling.
- [daemon](#)
- str [boss_url](#) = [load_boss_config](#)()
- [PORT](#) = int(os.environ[f"{args.task.upper()}_PORT"])
- Flask [app](#) = [create_app](#)(args.task, [boss_url](#))
- [host](#)
- [port](#)
- [use_reloader](#)

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

Namespaces

- namespace [src](#)
- namespace [src.main](#)

Functions

- [convert_single](#) ()
Converts one EPUB file to TEI format.
- [convert_from_csv](#) ()
Converts several EPUB files to TEI format.
- [chunk_single](#) ()
Creates a Story and many Chunks from a TEI file.
- [test_relation_extraction](#) ()
Runs REBEL on a basic example; used for debugging.
- [process_single](#) ()
Uses NLP and LLM to process an existing TEI file.
- [graph_triple_files](#) (session)
Loads JSON into Neo4j to test the Blazor graph page.
- [output_single](#) (session)
Generates a summary from triples stored in JSON, and posts data to Blazor.
- [full_pipeline](#) (session, collection_name, epub_path, book_chapters, start_str, end_str, book_id, story_id, book_title)
- [old_main](#) (session, collection_name)
- [pipeline_1](#) (epub_path, book_chapters, start_str, end_str, book_id, story_id)
Connects all components to convert an EPUB file to a book summary.
- [pipeline_2](#) (session, collection_name, chunks, book_title)
Extracts triples from a random chunk.
- [pipeline_3](#) (session, triples)
Generates a LLM summary using Neo4j triples.
- [pipeline_4](#) (session, collection_name, triples_string, chunk_id)
Generate chunk summary.
- [pipeline_5a](#) (summary, book_title, book_id)
Send book info to Blazor.
- [pipeline_5b](#) (summary, book_title, book_id, chunk, gold_summary="", float bookscore=None, float questeval=None)
Send metrics to Blazor.
- Dict[str, str] [load_worker_config](#) (List[str] task_types)
Load worker service URLs from environment variables.
- None [clear_task_data](#) (MongoHandle mongo_db, str collection_name, str chunk_id, str task_name)
Clear any existing task data before assigning new task to worker.
- bool [assign_task_to_worker](#) (str worker_url, str database_name, str collection_name, str chunk_id)
Assign a task to a worker microservice.
- Flask [create_app](#) (DocumentConnector docs_db, str database_name, str collection_name, Dict[str, str] worker_urls)
Create and configure Flask application for boss service.
- requests.models.Response [post_story_status](#) (int boss_port, int story_id, str task, str status)
Helpers to interact with the Flask boss thread.
- requests.models.Response [post_chunk_status](#) (int boss_port, str chunk_id, int story_id, str task, str status)
Send a chunk-level update to the boss Flask app.
- requests.models.Response [post_process_full_story](#) (int boss_port, int story_id, str task_type)
Process all chunks in MongoDB matching the provided story ID.

Variables

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

7.14 /home/runner/work/dsci-capstone/dsci-capstone/src/setup.py File Reference

Classes

- class `Session`
Stores active database connections and configuration settings.

Namespaces

- namespace `src`
- namespace `src.setup`

Variables

- `session` = `Session()`

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

Classes

- class [Log](#)
The Log class standardizes console output.
- class [Log.Failure](#)

Namespaces

- namespace [src](#)
- namespace [src.util](#)

Functions

- [all_none](#) (*args)
Checks if all provided args are None.
- [DataFrame df_natural_sorted](#) (DataFrame df, List[str] ignored_columns=[], List[str] sort_columns=[])
Sort a DataFrame in natural order using only certain columns.
- bool [check_values](#) (List[Any] results, List[Any] expected, bool verbose, str log_source, bool raise_error)
Safely compare two lists of values.

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

Namespaces

- namespace [tests](#)
- namespace [tests.conftest](#)

Functions

- [pytest_addoption](#) (parser)
- [session](#) (request)
Fixture to create session.

7.17 /home/runner/work/dsci-capstone/dsci-capstone/tests/test_↵ components.py File Reference

Namespaces

- namespace [tests](#)
- namespace [tests.test_components](#)

Functions

- [RelationalConnector relational_db](#) ([Session session](#))
Fixture to get relational database connection.
- [DocumentConnector docs_db](#) ([Session session](#))
Fixture to get document database connection.
- [GraphConnector graph_db](#) ([Session session](#))
Fixture to get document database connection.
- None [test_db_relational_minimal](#) ([RelationalConnector relational_db](#))
Tests if the RelationalConnector has a valid connection string.
- None [test_db_docs_minimal](#) ([DocumentConnector docs_db](#))
Tests if the DocumentConnector has a valid connection string.
- None [test_db_graph_minimal](#) ([GraphConnector graph_db](#))
Tests if the GraphConnector has a valid connection string.
- None [test_db_relational_comprehensive](#) ([RelationalConnector relational_db](#))
Tests if the GraphConnector is working as intended.
- None [test_db_docs_comprehensive](#) ([DocumentConnector docs_db](#))
Tests if the GraphConnector is working as intended.
- None [test_db_graph_comprehensive](#) ([GraphConnector graph_db](#))
Tests if the GraphConnector is working as intended.
- Generator[None, None, None] [load_examples_relational](#) ([RelationalConnector relational_db](#))
Fixture to create relational tables using engine-specific syntax.
- None [test_sql_example_1](#) ([RelationalConnector relational_db](#), Generator[None, None, None] [load_examples_relational](#))
Run queries contained within test files.
- None [test_sql_example_2](#) ([RelationalConnector relational_db](#), Generator[None, None, None] [load_examples_relational](#))
Run queries contained within test files.
- None [test_mongo_example_1](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_mongo_example_2](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_mongo_example_3](#) ([DocumentConnector docs_db](#))
Run queries contained within test files.
- None [test_cypher_example_1](#) ([GraphConnector graph_db](#))
Run queries contained within test files.
- None [test_cypher_example_2](#) ([GraphConnector graph_db](#))
Test social network graph with relationships and mixed query patterns.
- None [test_cypher_example_3](#) ([GraphConnector graph_db](#))
Test scene and dialogue graphs with proper isolation.
- None [test_cypher_example_4](#) ([GraphConnector graph_db](#))
Test event graph with property mutations and multi-hop traversal.
- None [_test_query_file](#) ([DatabaseConnector db_fixture](#), str filename, List[str] valid_files)
Run queries from a local file through the database.
- None [test_knowledge_graph_triples](#) ([GraphConnector graph_db](#))
Test KnowledgeGraph triple operations using add_triple and get_all_triples.
- [KnowledgeGraph nature_scene_graph](#) ([GraphConnector graph_db](#))
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.

7.18 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/_Imports.razor File Reference↵

7.19 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/App.razor File Reference↵

7.20 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/MainLayout.razor File Reference↵

7.21 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/NavMenu.razor File Reference↵

7.22 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Error.razor File Reference↵

7.23 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Graph.razor File Reference↵

7.24 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Home.razor File Reference↵

7.25 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Metrics.razor File Reference↵

7.26 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Routes.razor File Reference↵

7.27 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Controllers/MetricsController.cs File Reference↵

Classes

- class [MetricsController](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Controllers](#)

7.28 [/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Hubs/MetricsHub.cs](#) File Reference

Classes

- class [MetricsHub](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Hubs](#)

7.29 [/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/PRF1Metric.cs](#) File Reference

Classes

- class [PRF1Metric](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

7.30 [/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAItem.cs](#) File Reference

Classes

- class [QAItem](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

7.31 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/QAMetric.cs File Reference ↩

Classes

- class [QAMetric](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

7.32 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/ScalarMetric.cs File Reference ↩

Classes

- class [ScalarMetric](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

7.33 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryData.cs File Reference ↩

Classes

- class [SummaryData](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

7.34 /home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Models/SummaryMetrics.cs File Reference ↩

Classes

- class [SummaryMetrics](#)

Namespaces

- namespace [BlazorApp](#)
- namespace [BlazorApp.Models](#)

Index

/home/runner/work/dsci-capstone/dsci-capstone/components/__init__.py, 171
/home/runner/work/dsci-capstone/dsci-capstone/components/converters.py, 167
/home/runner/work/dsci-capstone/dsci-capstone/components/connectors.py, 168
/home/runner/work/dsci-capstone/dsci-capstone/components/corpusapp/BlazorApp/Components/Routes.razor, 168
/home/runner/work/dsci-capstone/dsci-capstone/components/dataloader.py, 169
/home/runner/work/dsci-capstone/dsci-capstone/components/fact_store.py, 169
/home/runner/work/dsci-capstone/dsci-capstone/components/metricsapp/BlazorApp/Controllers/MetricsController.cs, 170
/home/runner/work/dsci-capstone/dsci-capstone/components/semantic_web.py, 170
/home/runner/work/dsci-capstone/dsci-capstone/components/text_processing.py, 171
/home/runner/work/dsci-capstone/dsci-capstone/src/__init__.py, 171
/home/runner/work/dsci-capstone/dsci-capstone/src/flask.py, 172
/home/runner/work/dsci-capstone/dsci-capstone/src/main.py, 173
/home/runner/work/dsci-capstone/dsci-capstone/src/setup.py, 174
/home/runner/work/dsci-capstone/dsci-capstone/src/util.py, 175
/home/runner/work/dsci-capstone/dsci-capstone/tests/__init__.py, 171
/home/runner/work/dsci-capstone/dsci-capstone/tests/conftest.py, 175
/home/runner/work/dsci-capstone/dsci-capstone/tests/test_components.py, 175
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/App.razor, 177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/MainLayout.razor, 177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Layout/NavMenu.razor, 177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Error.razor, 177
/home/runner/work/dsci-capstone/dsci-capstone/web-app/BlazorApp/Components/Pages/Graph.razor, 177
/home/runner/work/dsci-capstone/dsci-capstone/web-
__new__

Book, 51
BookStream, 54
Comments, 55
DatabaseConnector, 64
DocumentConnector, 74
EPUBToTEI, 81
GraphConnector, 87
KnowledgeGraph, 97
LLMConnector, 109
Log.Failure, 84
Metrics, 125
mysqlConnector, 136
ParagraphStreamTEI, 138
postgresConnector, 144
RelationalConnector, 150
RelationExtractor, 156
Session, 158
Story, 161

- Session, 159
- __repr__
 - Chunk, 56
- __str__
 - Log.Failure, 84
- _auth_suffix
 - DocumentConnector, 80
- _docs_to_df
 - components.document_storage, 14
- _execute_retag_db
 - GraphConnector, 88
- _fetch_latest
 - GraphConnector, 88
- _filter_to_db
 - components.fact_storage, 17
- _find_compatible_nested_key
 - components.document_storage, 14
- _flatten_recursive
 - components.document_storage, 15
- _graph_name
 - GraphConnector, 95
- _hubContext
 - MetricsController, 130
- _instance
 - Session, 159
- _is_single_query
 - DatabaseConnector, 64
- _logger
 - MetricsController, 130
 - MetricsHub, 132
- _make_single
 - Story, 161
- _merge_chunks
 - Story, 161
- _normalize_elements
 - components.fact_storage, 17
- _parsable_to_df
 - DatabaseConnector, 65
 - DocumentConnector, 74
 - GraphConnector, 88
 - RelationalConnector, 150
- _prune_bad_tags
 - EPUBToTEI, 81
- _returns_data
 - DatabaseConnector, 65
 - DocumentConnector, 75
 - GraphConnector, 89
 - RelationalConnector, 150
- _sanitize_document
 - components.document_storage, 15
- _sanitize_ids
 - EPUBToTEI, 81
- _sanitize_json
 - components.document_storage, 15
- _split_combined
 - DatabaseConnector, 65
 - DocumentConnector, 75
 - GraphConnector, 89
- RelationalConnector, 151
- _test_query_file
 - tests.test_components, 44
- _tuples_to_df
 - components.fact_storage, 18
- Accuracy
 - QALtem, 145
- add_triple
 - KnowledgeGraph, 97
- all_none
 - src.util, 41
- allowed_chapters
 - ParagraphStreamTEI, 139
- app
 - src.flask, 27
 - src.main, 37
- args
 - src.flask, 27
- assign_task_to_worker
 - src.main, 31
- AverageAccuracy
 - QAMetric, 146
- bad_addr
 - Log, 116
- bad_path
 - Log, 116
- bad_val
 - Log, 116
- BlazorApp, 9
- BlazorApp.Controllers, 9
- BlazorApp.Hubs, 9
- BlazorApp.Models, 9
- Book, 51
 - __init__, 51
 - stream_chapters, 51
- book
 - BookStream, 54
- book_id
 - Chunk, 57
 - ParagraphStreamTEI, 139
 - src.main, 37
- book_title
 - src.main, 37
- BookFactory, 52
 - create_book, 52
- BookID
 - SummaryData, 164
- BookStream, 53
 - __init__, 54
 - book, 54
 - stream_segments, 54
- BookTitle
 - SummaryData, 164
- BOSS_PORT
 - src.main, 37
- boss_url
 - src.flask, 27

- BRIGHT
 - Log, 116
- change_database
 - DatabaseConnector, 66
 - DocumentConnector, 75
 - GraphConnector, 89
 - RelationalConnector, 151
- chapter_number
 - Chunk, 57
- chapter_percent
 - Chunk, 57
- chapters
 - src.main, 37
- char_count
 - Chunk, 56
- check_connection
 - Connector, 59
 - DocumentConnector, 76
 - GraphConnector, 90
 - LLMConnector, 109
 - RelationalConnector, 152
- check_values
 - src.util, 41
- Chunk, 54
 - __init__, 55
 - __repr__, 56
 - book_id, 57
 - chapter_number, 57
 - chapter_percent, 57
 - char_count, 56
 - get_chunk_id, 56
 - line_end, 57
 - line_start, 57
 - story_id, 57
 - story_percent, 57
 - text, 57
 - to_mongo_dict, 57
- chunk
 - src.main, 37
- chunk_bookscore
 - components.metrics, 19
- chunk_id
 - src.main, 38
- chunk_single
 - src.main, 31
- chunks
 - ParagraphStreamTEI, 139
 - src.main, 38
- clean_tei
 - EPUBToTEI, 82
- clean_tei_content
 - EPUBToTEI, 82
- clear_task_data
 - src.main, 31
- COLLECTION
 - src.main, 38
- collection
 - src.main, 38
- components, 10
- components.book_conversion, 10
 - nlp, 10
 - sentencizer, 10
- components.connectors, 11
- components.corpus, 11
 - df, 13
 - df_booksum, 13
 - df_nqa, 13
 - fuzzy_merge_titles, 11
 - index, 13
 - load_booksum, 12
 - load_narrativeqa, 12
 - m, 13
 - merge_dataframes, 12
 - normalize_title, 12
 - to_df_booksum, 12
 - to_df_nqa, 12
- components.document_storage, 13
 - _docs_to_df, 14
 - _find_compatible_nested_key, 14
 - _flatten_recursive, 15
 - _sanitize_document, 15
 - _sanitize_json, 15
 - mongo_handle, 16
 - MongoHandle, 16
- components.fact_storage, 17
 - _filter_to_db, 17
 - _normalize_elements, 17
 - _tuples_to_df, 18
- components.metrics, 18
 - chunk_bookscore, 19
 - run_bookscore, 19
 - run_questeval, 21
- components.semantic_web, 22
- components.text_processing, 22
 - nlp, 22
 - sentencizer, 22
- compute_basic_metrics
 - Metrics, 125
- configure
 - Connector, 60
 - DatabaseConnector, 66
 - LLMConnector, 109
- conn_abc
 - Log, 116
- connection_string
 - DatabaseConnector, 70
 - DocumentConnector, 80
 - GraphConnector, 95
 - RelationalConnector, 155
- Connector, 58
 - check_connection, 59
 - configure, 60
 - execute_file, 60
 - execute_query, 60
 - test_connection, 61
- convert_from_csv

- src.main, 32
- convert_single
 - src.main, 32
- convert_to_tei
 - EPUBToTEI, 82
- create_app
 - src.flask, 24
 - src.main, 32
- create_book
 - BookFactory, 52
- create_database
 - DatabaseConnector, 66
 - DocumentConnector, 76
 - GraphConnector, 90
 - RelationalConnector, 152
- create_db
 - Log, 116
- create_summary_payload
 - Metrics, 125
- daemon
 - src.flask, 28
 - src.main, 38
- database
 - KnowledgeGraph, 106
- database_exists
 - DatabaseConnector, 67
 - DocumentConnector, 77
 - GraphConnector, 91
 - RelationalConnector, 152
- database_name
 - DocumentConnector, 80
 - GraphConnector, 95
 - RelationalConnector, 155
- DatabaseConnector, 61
 - __init__, 64
 - _is_single_query, 64
 - _parsable_to_df, 65
 - _returns_data, 65
 - _split_combined, 65
 - change_database, 66
 - configure, 66
 - connection_string, 70
 - create_database, 66
 - database_exists, 67
 - db_engine, 70
 - db_type, 70
 - drop_database, 67
 - execute_combined, 68
 - execute_file, 68
 - execute_query, 68
 - get_dataframe, 69
 - host, 70
 - password, 70
 - port, 70
 - temp_database, 69
 - username, 70
 - verbose, 71
- db_conn_abc
 - Log, 117
- db_engine
 - DatabaseConnector, 70
- db_exists
 - Log, 117
- DB_NAME
 - src.main, 38
- db_type
 - DatabaseConnector, 70
 - RelationalConnector, 155
- delete_dummy
 - DocumentConnector, 77
 - GraphConnector, 91
- detect_community_clusters
 - KnowledgeGraph, 98
- df
 - components.corpus, 13
- df_booksum
 - components.corpus, 13
- df_natural_sorted
 - src.util, 42
- df_nqa
 - components.corpus, 13
- doc_db
 - Log, 117
- docs_db
 - Session, 159
 - tests.test_components, 44
- DocumentConnector, 71
 - __init__, 74
 - _auth_suffix, 80
 - _parsable_to_df, 74
 - _returns_data, 75
 - _split_combined, 75
 - change_database, 75
 - check_connection, 76
 - connection_string, 80
 - create_database, 76
 - database_exists, 77
 - database_name, 80
 - delete_dummy, 77
 - drop_database, 77
 - execute_query, 78
 - get_dataframe, 78
 - get_unmanaged_handle, 79
 - test_connection, 79
 - verbose, 80
- drop_database
 - DatabaseConnector, 67
 - DocumentConnector, 77
 - GraphConnector, 91
 - RelationalConnector, 153
- drop_db
 - Log, 117
- drop_gr
 - Log, 117
- drop_graph
 - GraphConnector, 92

- encoding
 - EPUBToTEI, 82
 - ParagraphStreamTEI, 139
- end
 - src.main, 38
- end_inclusive
 - ParagraphStreamTEI, 140
- epub_path
 - EPUBToTEI, 82
- EPUBToTEI, 80
 - __init__, 81
 - _prune_bad_tags, 81
 - _sanitize_ids, 81
 - clean_tei, 82
 - clean_tei_content, 82
 - convert_to_tei, 82
 - encoding, 82
 - epub_path, 82
 - pandoc_xml_path, 82
 - raw_tei_content, 83
 - tei_path, 83
 - xml_namespace, 83
- execute_combined
 - DatabaseConnector, 68
- execute_file
 - Connector, 60
 - DatabaseConnector, 68
 - LLMConnector, 110
- execute_full_query
 - LLMConnector, 110
- execute_query
 - Connector, 60
 - DatabaseConnector, 68
 - DocumentConnector, 78
 - GraphConnector, 92
 - LLMConnector, 110
 - RelationalConnector, 153
- extract
 - RelationExtractor, 156
- F1Score
 - PRF1Metric, 145
- fail
 - Log, 114
- fail_legacy
 - Log, 115
- FAILURE_COLOR
 - Log, 117
- find_element_names
 - KnowledgeGraph, 98
- from_env
 - RelationalConnector, 154
- FULL_DF
 - Log, 117
- full_pipeline
 - src.main, 33
- fuzzy_merge_titles
 - components.corpus, 11
- generate_default_metrics
 - Metrics, 126
- generate_example_metrics
 - Metrics, 127
- GeneratedAnswer
 - QAItem, 145
- get_all_triples
 - KnowledgeGraph, 99
- get_chunk_id
 - Chunk, 56
- get_community_subgraph
 - KnowledgeGraph, 100
- get_dataframe
 - DatabaseConnector, 69
 - DocumentConnector, 78
 - GraphConnector, 93
 - RelationalConnector, 154
- get_df
 - Log, 117
- get_edge_counts
 - KnowledgeGraph, 100
- get_neighborhood
 - KnowledgeGraph, 101
- get_node_context
 - KnowledgeGraph, 101
- get_random_walk_sample
 - KnowledgeGraph, 102
- get_relation_summary
 - KnowledgeGraph, 102
- get_subgraph_by_nodes
 - KnowledgeGraph, 102
- get_summary_stats
 - KnowledgeGraph, 103
- get_task_info
 - src.flask, 24
- get_triple_properties
 - KnowledgeGraph, 103
- get_unique
 - GraphConnector, 93
 - Log, 117
- get_unmanaged_handle
 - DocumentConnector, 79
- GetAll
 - MetricsController, 130
- GetDefault
 - SummaryMetrics, 165
- GetIndex
 - MetricsController, 130
- GoldAnswer
 - QAItem, 145
- GoldSummaryText
 - SummaryData, 164
- good_val
 - Log, 118
- gr_db
 - Log, 118
- gr_rag
 - Log, 118

- graph_db
 - Session, 159
 - tests.test_components, 44
- graph_name
 - KnowledgeGraph, 107
- graph_triple_files
 - src.main, 33
- GraphConnector, 85
 - __init__, 87
 - _execute_retag_db, 88
 - _fetch_latest, 88
 - _graph_name, 95
 - _parsable_to_df, 88
 - _returns_data, 89
 - _split_combined, 89
 - change_database, 89
 - check_connection, 90
 - connection_string, 95
 - create_database, 90
 - database_exists, 91
 - database_name, 95
 - delete_dummy, 91
 - drop_database, 91
 - drop_graph, 92
 - execute_query, 92
 - get_dataframe, 93
 - get_unique, 93
 - IS_DUMMY_, 94
 - NOT_DUMMY_, 94
 - SAME_DB_KG_, 94
 - temp_graph, 94
 - test_connection, 95
 - verbose, 96
- GREEN
 - Log, 118
- help
 - src.flask, 28
- HOST
 - Metrics, 128
- host
 - DatabaseConnector, 70
 - src.flask, 28
- index
 - components.corpus, 13
- IS_DUMMY_
 - GraphConnector, 94
- IsCorrect
 - QAItem, 146
- kg
 - Log, 118
- KnowledgeGraph, 96
 - __init__, 97
 - add_triple, 97
 - database, 106
 - detect_community_clusters, 98
 - find_element_names, 98
 - get_all_triples, 99
 - get_community_subgraph, 100
 - get_edge_counts, 100
 - get_neighborhood, 101
 - get_node_context, 101
 - get_random_walk_sample, 102
 - get_relation_summary, 102
 - get_subgraph_by_nodes, 102
 - get_summary_stats, 103
 - get_triple_properties, 103
 - graph_name, 107
 - print_nodes, 104
 - print_triples, 104
 - to_contextualized_string, 104
 - to_narrative, 105
 - to_triple_string, 105
 - triples_to_names, 106
 - verbose, 107
- line_end
 - Chunk, 57
- line_start
 - Chunk, 57
- lines
 - ParagraphStreamTEI, 140
- llm
 - LLMConnector, 112
- LLMConnector, 107
 - __init__, 109
 - check_connection, 109
 - configure, 109
 - execute_file, 110
 - execute_full_query, 110
 - execute_query, 110
 - llm, 112
 - model_name, 112
 - normalize_triples, 111
 - system_prompt, 112
 - temperature, 112
 - test_connection, 111
- load_booksum
 - components.corpus, 12
- load_boss_config
 - src.flask, 24
- load_examples_relational
 - tests.test_components, 44
- load_imports
 - src.flask, 25
- load_mongo_config
 - src.flask, 25
- load_narrativeqa
 - components.corpus, 12
- load_worker_config
 - src.main, 33
- Log, 112
 - bad_addr, 116
 - bad_path, 116
 - bad_val, 116
 - BRIGHT, 116

- conn_abc, 116
- create_db, 116
- db_conn_abc, 117
- db_exists, 117
- doc_db, 117
- drop_db, 117
- drop_gr, 117
- fail, 114
- fail_legacy, 115
- FAILURE_COLOR, 117
- FULL_DF, 117
- get_df, 117
- get_unique, 117
- good_val, 118
- gr_db, 118
- gr_rag, 118
- GREEN, 118
- kg, 118
- msg_bad_addr, 118
- msg_bad_coll, 118
- msg_bad_exec_f, 118
- msg_bad_exec_q, 118
- msg_bad_graph, 118
- msg_bad_path, 119
- msg_bad_table, 119
- msg_bad triples, 119
- MSG_COLOR, 119
- msg_compare, 119
- msg_db_connect, 119
- msg_db_current, 119
- msg_db_exists, 119
- msg_db_not_found, 119
- msg_fail_manage_db, 120
- msg_fail_manage_gr, 120
- msg_fail_parse, 120
- msg_good_coll, 120
- msg_good_exec_f, 120
- msg_good_exec_q, 120
- msg_good_exec_qr, 120
- msg_good_graph, 120
- msg_good_path, 121
- msg_good_table, 121
- msg_multiple_query, 121
- msg_result, 121
- msg_success_managed_db, 121
- msg_success_managed_gr, 121
- msg_swap_db, 121
- msg_swap_kg, 122
- msg_unknown_error, 122
- pytest_db, 122
- RED, 122
- rel_db, 122
- run_f, 122
- run_q, 122
- success, 115
- SUCCESS_COLOR, 122
- success_legacy, 115
- swap_db, 122
- swap_kg, 123
- test_basic, 123
- test_conn, 123
- test_df, 123
- test_info, 123
- test_tmp_db, 123
- USE_COLORS, 123
- warn, 116
- WARNING_COLOR, 123
- WHITE, 123
- YELLOW, 124
- Log.Failure, 83
 - __init__, 84
 - __str__, 84
 - msg, 84
 - prefix, 84
- m
 - components.corpus, 13
- main_graph
 - Session, 159
- mark_task_in_progress
 - src.flask, 25
- max_tokens
 - RelationExtractor, 156
- merge_dataframes
 - components.corpus, 12
- Metrics, 124
 - __init__, 125
 - compute_basic_metrics, 125
 - create_summary_payload, 125
 - generate_default_metrics, 126
 - generate_example_metrics, 127
 - HOST, 128
 - PORT, 128
 - post_basic_metrics, 127
 - post_basic_output, 128
 - post_payload, 128
 - SummaryData, 164
 - timeout_seconds, 128
 - url, 128
- MetricsController, 129
 - _hubContext, 130
 - _logger, 130
 - GetAll, 130
 - GetIndex, 130
 - MetricsController, 130
 - Post, 130
 - Summaries, 130
- MetricsHub, 131
 - _logger, 132
 - MetricsHub, 132
 - OnConnectedAsync, 132
 - OnDisconnectedAsync, 132
- model
 - RelationExtractor, 156
- model_name
 - LLMConnector, 112
- mongo_db

- src.main, 38
- mongo_handle
 - components.document_storage, 16
- MongoHandle
 - components.document_storage, 16
 - src.flask, 28
 - src.main, 38
- msg
 - Log.Failure, 84
- msg_bad_addr
 - Log, 118
- msg_bad_coll
 - Log, 118
- msg_bad_exec_f
 - Log, 118
- msg_bad_exec_q
 - Log, 118
- msg_bad_graph
 - Log, 118
- msg_bad_path
 - Log, 119
- msg_bad_table
 - Log, 119
- msg_bad_triples
 - Log, 119
- MSG_COLOR
 - Log, 119
- msg_compare
 - Log, 119
- msg_db_connect
 - Log, 119
- msg_db_current
 - Log, 119
- msg_db_exists
 - Log, 119
- msg_db_not_found
 - Log, 119
- msg_fail_manage_db
 - Log, 120
- msg_fail_manage_gr
 - Log, 120
- msg_fail_parse
 - Log, 120
- msg_good_coll
 - Log, 120
- msg_good_exec_f
 - Log, 120
- msg_good_exec_q
 - Log, 120
- msg_good_exec_qr
 - Log, 120
- msg_good_graph
 - Log, 120
- msg_good_path
 - Log, 121
- msg_good_table
 - Log, 121
- msg_multiple_query
 - Log, 121
- msg_result
 - Log, 121
- msg_success_managed_db
 - Log, 121
- msg_success_managed_gr
 - Log, 121
- msg_swap_db
 - Log, 121
- msg_swap_kg
 - Log, 122
- msg_unknown_error
 - Log, 122
- mysqlConnector, 132
 - __init__, 136
 - specific_queries, 136
- Name
 - PRF1Metric, 145
 - ScalarMetric, 157
- nature_scene_graph
 - tests.test_components, 45
- nlp
 - components.book_conversion, 10
 - components.text_processing, 22
- normalize_title
 - components.corpus, 12
- normalize_triples
 - LLMConnector, 111
- NOT_DUMMY_
 - GraphConnector, 94
- notify_boss
 - src.flask, 26
- old_main
 - src.main, 33
- OnConnectedAsync
 - MetricsHub, 132
- OnDisconnectedAsync
 - MetricsHub, 132
- output_single
 - src.main, 33
- pandoc_xml_path
 - EPUBToTEI, 82
- ParagraphStreamTEI, 136
 - __init__, 138
 - allowed_chapters, 139
 - book_id, 139
 - chunks, 139
 - encoding, 139
 - end_inclusive, 140
 - lines, 140
 - pre_compute_segments, 139
 - root, 140
 - start_inclusive, 140
 - story_id, 140
 - stream_segments, 139
 - tei_path, 140

- xml_namespace, 140
- parser
 - src.flask, 28
- password
 - DatabaseConnector, 70
- pipeline_1
 - src.main, 33
- pipeline_2
 - src.main, 34
- pipeline_3
 - src.main, 34
- pipeline_4
 - src.main, 34
- pipeline_5a
 - src.main, 34
- pipeline_5b
 - src.main, 35
- PORT
 - Metrics, 128
 - src.flask, 28
- port
 - DatabaseConnector, 70
 - src.flask, 28
- Post
 - MetricsController, 130
- post_basic_metrics
 - Metrics, 127
- post_basic_output
 - Metrics, 128
- post_chunk_status
 - src.main, 35
- post_payload
 - Metrics, 128
- post_process_full_story
 - src.main, 36
- post_story_status
 - src.main, 36
- postgresConnector, 141
 - __init__, 144
 - specific_queries, 144
- pre_compute_segments
 - ParagraphStreamTEI, 139
- pre_split_chunks
 - Story, 161
- Precision
 - PRF1Metric, 145
- prefix
 - Log.Failure, 84
- PRF1Metric, 144
 - F1Score, 145
 - Name, 145
 - Precision, 145
 - Recall, 145
- PRF1Metrics
 - SummaryMetrics, 165
- print_nodes
 - KnowledgeGraph, 104
- print_triples
 - KnowledgeGraph, 104
- process_single
 - src.main, 36
- process_task
 - src.flask, 26
- pytest_adoption
 - tests.conftest, 42
- pytest_db
 - Log, 122
- QA
 - SummaryMetrics, 165
- QALtem, 145
 - Accuracy, 145
 - GeneratedAnswer, 145
 - GoldAnswer, 145
 - IsCorrect, 146
 - Question, 146
- QALtems
 - QAMetric, 146
- QAMetric, 146
 - AverageAccuracy, 146
 - QALtems, 146
- QAResults
 - SummaryData, 164
- Question
 - QALtem, 146
- raw_tei_content
 - EPUBToTEI, 83
- reader
 - Story, 162
- Recall
 - PRF1Metric, 145
- RED
 - Log, 122
- rel_db
 - Log, 122
- relational_db
 - Session, 160
 - tests.test_components, 45
- RelationalConnector, 147
 - __init__, 150
 - _parsable_to_df, 150
 - _returns_data, 150
 - _split_combined, 151
 - change_database, 151
 - check_connection, 152
 - connection_string, 155
 - create_database, 152
 - database_exists, 152
 - database_name, 155
 - db_type, 155
 - drop_database, 153
 - execute_query, 153
 - from_env, 154
 - get_dataframe, 154
 - test_connection, 155
 - verbose, 155

- RelationExtractor, 156
 - __init__, 156
 - extract, 156
 - max_tokens, 156
 - model, 156
 - tokenizer, 157
 - tuple_delim, 157
- required
 - src.flask, 28
- reset
 - Session, 159
- response
 - src.main, 39
- response_files
 - src.main, 39
- root
 - ParagraphStreamTEI, 140
- run_app
 - src.main, 39
- run_bookscore
 - components.metrics, 19
- run_f
 - Log, 122
- run_q
 - Log, 122
- run_questeval
 - components.metrics, 21
- SAME_DB_KG_
 - GraphConnector, 94
- save_task_result
 - src.flask, 27
- ScalarMetric, 157
 - Name, 157
 - Value, 157
- ScalarMetrics
 - SummaryMetrics, 165
- sentencizer
 - components.book_conversion, 10
 - components.text_processing, 22
- Session, 157
 - __init__, 158
 - __new__, 159
 - _instance, 159
 - docs_db, 159
 - graph_db, 159
 - main_graph, 159
 - relational_db, 160
 - reset, 159
 - test_database_connections, 159
 - verbose, 160
- session
 - src.main, 39
 - src.setup, 40
 - tests.conftest, 42
- specific_queries
 - mysqlConnector, 136
 - postgresConnector, 144
- src, 23
 - src.flask, 23
 - app, 27
 - args, 27
 - boss_url, 27
 - create_app, 24
 - daemon, 28
 - get_task_info, 24
 - help, 28
 - host, 28
 - load_boss_config, 24
 - load_imports, 25
 - load_mongo_config, 25
 - mark_task_in_progress, 25
 - MongoHandle, 28
 - notify_boss, 26
 - parser, 28
 - PORT, 28
 - port, 28
 - process_task, 26
 - required, 28
 - save_task_result, 27
 - target, 28
 - task_queue, 28
 - task_worker, 29
 - True, 29
 - use_reloader, 29
 - src.main, 29
 - app, 37
 - assign_task_to_worker, 31
 - book_id, 37
 - book_title, 37
 - BOSS_PORT, 37
 - chapters, 37
 - chunk, 37
 - chunk_id, 38
 - chunk_single, 31
 - chunks, 38
 - clear_task_data, 31
 - COLLECTION, 38
 - collection, 38
 - convert_from_csv, 32
 - convert_single, 32
 - create_app, 32
 - daemon, 38
 - DB_NAME, 38
 - end, 38
 - full_pipeline, 33
 - graph_triple_files, 33
 - load_worker_config, 33
 - mongo_db, 38
 - MongoHandle, 38
 - old_main, 33
 - output_single, 33
 - pipeline_1, 33
 - pipeline_2, 34
 - pipeline_3, 34
 - pipeline_4, 34
 - pipeline_5a, 34

- pipeline_5b, 35
- post_chunk_status, 35
- post_process_full_story, 36
- post_story_status, 36
- process_single, 36
- response, 39
- response_files, 39
- run_app, 39
- session, 39
- start, 39
- story_id, 39
- summary, 39
- target, 39
- task_types, 39
- tei, 39
- test_relation_extraction, 37
- triple_files, 40
- triples, 40
- triples_string, 40
- worker_urls, 40
- src.setup, 40
 - session, 40
- src.util, 41
 - all_none, 41
 - check_values, 41
 - df_natural_sorted, 42
- start
 - src.main, 39
- start_inclusive
 - ParagraphStreamTEI, 140
- Story, 160
 - __init__, 161
 - _make_single, 161
 - _merge_chunks, 161
 - pre_split_chunks, 161
 - reader, 162
 - stream_chunks, 161
- story_id
 - Chunk, 57
 - ParagraphStreamTEI, 140
 - src.main, 39
- story_percent
 - Chunk, 57
- StoryStreamAdapter, 162
 - stream_paragraphs, 163
 - stream_segments, 163
 - stream_sentences, 163
- stream_chapters
 - Book, 51
- stream_chunks
 - Story, 161
- stream_paragraphs
 - StoryStreamAdapter, 163
- stream_segments
 - BookStream, 54
 - ParagraphStreamTEI, 139
 - StoryStreamAdapter, 163
- stream_sentences
 - StoryStreamAdapter, 163
- success
 - Log, 115
- SUCCESS_COLOR
 - Log, 122
- success_legacy
 - Log, 115
- Summaries
 - MetricsController, 130
- summary
 - src.main, 39
- SummaryData, 164
 - BookID, 164
 - BookTitle, 164
 - GoldSummaryText, 164
 - Metrics, 164
 - QAResults, 164
 - SummaryText, 164
- SummaryMetrics, 165
 - GetDefault, 165
 - PRF1Metrics, 165
 - QA, 165
 - ScalarMetrics, 165
- SummaryText
 - SummaryData, 164
- swap_db
 - Log, 122
- swap_kg
 - Log, 123
- system_prompt
 - LLMConnector, 112
- target
 - src.flask, 28
 - src.main, 39
- task_queue
 - src.flask, 28
- task_types
 - src.main, 39
- task_worker
 - src.flask, 29
- tei
 - src.main, 39
- tei_path
 - EPUBToTEI, 83
 - ParagraphStreamTEI, 140
- temp_database
 - DatabaseConnector, 69
- temp_graph
 - GraphConnector, 94
- temperature
 - LLMConnector, 112
- test_basic
 - Log, 123
- test_conn
 - Log, 123
- test_connection
 - Connector, 61
 - DocumentConnector, 79

- GraphConnector, 95
- LLMConnector, 111
- RelationalConnector, 155
- test_cypher_example_1
 - tests.test_components, 45
- test_cypher_example_2
 - tests.test_components, 45
- test_cypher_example_3
 - tests.test_components, 45
- test_cypher_example_4
 - tests.test_components, 46
- test_database_connections
 - Session, 159
- test_db_docs_comprehensive
 - tests.test_components, 46
- test_db_docs_minimal
 - tests.test_components, 46
- test_db_graph_comprehensive
 - tests.test_components, 46
- test_db_graph_minimal
 - tests.test_components, 46
- test_db_relational_comprehensive
 - tests.test_components, 46
- test_db_relational_minimal
 - tests.test_components, 47
- test_detect_community_clusters_comprehensive
 - tests.test_components, 47
- test_detect_community_clusters_minimal
 - tests.test_components, 47
- test_df
 - Log, 123
- test_get_neighborhood
 - tests.test_components, 47
- test_get_neighborhood_comprehensive
 - tests.test_components, 47
- test_get_random_walk_sample
 - tests.test_components, 48
- test_get_random_walk_sample_comprehensive
 - tests.test_components, 48
- test_get_subgraph_by_nodes
 - tests.test_components, 48
- test_info
 - Log, 123
- test_knowledge_graph_triples
 - tests.test_components, 48
- test_mongo_example_1
 - tests.test_components, 49
- test_mongo_example_2
 - tests.test_components, 49
- test_mongo_example_3
 - tests.test_components, 49
- test_relation_extraction
 - src.main, 37
- test_sql_example_1
 - tests.test_components, 49
- test_sql_example_2
 - tests.test_components, 50
- test_tmp_db
 - Log, 123
- tests, 42
- tests.conftest, 42
 - pytest_addoption, 42
 - session, 42
- tests.test_components, 43
 - _test_query_file, 44
 - docs_db, 44
 - graph_db, 44
 - load_examples_relational, 44
 - nature_scene_graph, 45
 - relational_db, 45
 - test_cypher_example_1, 45
 - test_cypher_example_2, 45
 - test_cypher_example_3, 45
 - test_cypher_example_4, 46
 - test_db_docs_comprehensive, 46
 - test_db_docs_minimal, 46
 - test_db_graph_comprehensive, 46
 - test_db_graph_minimal, 46
 - test_db_relational_comprehensive, 46
 - test_db_relational_minimal, 47
 - test_detect_community_clusters_comprehensive, 47
 - test_detect_community_clusters_minimal, 47
 - test_get_neighborhood, 47
 - test_get_neighborhood_comprehensive, 47
 - test_get_random_walk_sample, 48
 - test_get_random_walk_sample_comprehensive, 48
 - test_get_subgraph_by_nodes, 48
 - test_knowledge_graph_triples, 48
 - test_mongo_example_1, 49
 - test_mongo_example_2, 49
 - test_mongo_example_3, 49
 - test_sql_example_1, 49
 - test_sql_example_2, 50
- text
 - Chunk, 57
- timeout_seconds
 - Metrics, 128
- to_contextualized_string
 - KnowledgeGraph, 104
- to_df_booksum
 - components.corpus, 12
- to_df_nqa
 - components.corpus, 12
- to_mongo_dict
 - Chunk, 57
- to_narrative
 - KnowledgeGraph, 105
- to_triple_string
 - KnowledgeGraph, 105
- tokenizer
 - RelationExtractor, 157
- triple_files
 - src.main, 40
- triples

- src.main, [40](#)
- triples_string
 - src.main, [40](#)
- triples_to_names
 - KnowledgeGraph, [106](#)
- True
 - src.flask, [29](#)
- tuple_delim
 - RelationExtractor, [157](#)
- url
 - Metrics, [128](#)
- USE_COLORS
 - Log, [123](#)
- use_reloader
 - src.flask, [29](#)
- username
 - DatabaseConnector, [70](#)
- Value
 - ScalarMetric, [157](#)
- verbose
 - DatabaseConnector, [71](#)
 - DocumentConnector, [80](#)
 - GraphConnector, [96](#)
 - KnowledgeGraph, [107](#)
 - RelationalConnector, [155](#)
 - Session, [160](#)
- warn
 - Log, [116](#)
- WARNING_COLOR
 - Log, [123](#)
- WHITE
 - Log, [123](#)
- worker_urls
 - src.main, [40](#)
- xml_namespace
 - EPUBToTEI, [83](#)
 - ParagraphStreamTEI, [140](#)
- YELLOW
 - Log, [124](#)