

# CHAIN DOCUMENTATION

## TABLE OF CONTENTS

OBJECTIVES OF CHAIN .....	6
SETTING UP CHAIN .....	7
RUNNING FILES IN CHAIN .....	8
FILE STRUCTURE .....	9
TESTING RESULT DOCUMENTS.....	10
BREAKDOWN OF TESTING DOCUMENTS.....	10
MATCH_STRUC.JAVA.....	12
BASIC FUNCTIONALITY .....	12
DESIGN .....	12
PRIVATE VARIABLES/FIELDS .....	12
METHODS .....	14
ASSOCIATED TESTING DOCUMENTS .....	16
RUNNING THIS FILE .....	16
CALL_SPSM.JAVA .....	17
BASIC FUNCTIONALITY .....	17
DESIGN .....	17
PRIVATE VARIABLES/FIELDS .....	17
METHODS .....	18
ASSOCIATED TESTING DOCUMENTS .....	19
RUNNING THIS FILE .....	19
BEST_MATCH_RESULTS.JAVA .....	20
BASIC FUNCTIONALITY .....	20
DESIGN .....	20
PRIVATE VARIABLES/FIELDS .....	21
METHODS .....	21
ASSOCIATED TESTING DOCUMENTS .....	21
RUNNING THIS FILE .....	22
REPAIR_SCHEMA.JAVA.....	23
BASIC FUNCTIONALITY .....	23
DESIGN .....	23
PRIVATE VARIABLES/FIELDS .....	23
METHODS .....	24
ASSOCIATED TESTING DOCUMENTS .....	24
RUNNING THIS FILE .....	24
NODE.JAVA .....	25
BASIC FUNCTIONALITY .....	25
DESIGN .....	25

PRIVATE VARIABLES/FIELDS .....	25
METHODS .....	25
ASSOCIATED TESTING DOCUMENTS .....	27
RUNNING THIS FILE .....	27
<b>ONTOLOGY_STRUC.JAVA.....</b>	<b>28</b>
BASIC FUNCTIONALITY .....	28
DESIGN .....	28
PRIVATE VARIABLES/FIELDS .....	28
METHODS .....	28
ASSOCIATED TESTING DOCUMENTS .....	29
RUNNING THIS FILE .....	29
<b>CREATE_QUERY.JAVA.....</b>	<b>30</b>
BASIC FUNCTIONALITY .....	30
DESIGN .....	30
PRIVATE VARIABLES/FIELDS .....	30
METHODS .....	31
ASSOCIATED TESTING DOCUMENTS .....	32
RUNNING THIS FILE .....	32
<b>RUN_QUERY.JAVA.....</b>	<b>33</b>
BASIC FUNCTIONALITY .....	33
DESIGN .....	33
PRIVATE VARIABLES/FIELDS .....	33
METHODS .....	34
ASSOCIATED TESTING DOCUMENTS .....	34
RUNNING THIS FILE .....	34
<b>SCHEMA_FROM_QUERY.JAVA.....</b>	<b>35</b>
BASIC FUNCTIONALITY .....	35
DESIGN .....	35
PRIVATE VARIABLES/FIELDS .....	35
METHODS .....	36
ASSOCIATED TESTING DOCUMENTS .....	36
RUNNING THIS FILE .....	37
<b>SPSM_TEST_CASES.JAVA.....</b>	<b>39</b>
BASIC FUNCTIONALITY .....	39
EXPECTED RESULTS .....	39
TEST 1.1.....	39
TEST 2.1.....	39
TEST 2.2.....	39
TEST 2.3.....	39
TEST 2.4.....	40
TEST 3.1.....	40
TEST 3.2.....	40

TEST 3.3.....	40
TEST 4.1.....	40
TEST 4.2.....	40
TEST 4.3.....	41
TEST 4.4.....	41
TEST 4.5.....	41
TEST 5.1.....	41
TEST 5.2.....	41
TEST 5.3.....	41
TEST 5.4.....	41
TEST 5.5.....	41
TEST 5.6.....	42
TEST 5.7.....	42
TEST 5.8.....	42
TEST 6.1.....	42
TEST 6.2.....	42
TEST 6.3.....	42
TEST 6.4.....	42
TEST 6.5.....	42
TEST 6.6.....	43
TEST 6.7.....	43
<b>MATCH_RESULTS_TEST_CASES.JAVA.....</b>	<b>44</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>44</b>
<b>EXPECTED RESULTS .....</b>	<b>44</b>
TEST 1 – FAIL WITH LIMIT .....	44
TEST 2 – MULTIPLE SUCCESSES MATCH.....	44
TEST 3 – MULTIPLE SUCCESSES MATCH.....	44
TEST 4 – SUCCESS WITH LARGE LIMIT.....	45
TEST 5 – SINGLE FAIL MATCH.....	45
TEST 6 – EMPTY MATCHES.....	45
TEST 7 – SUCCESS WITH LIMIT .....	45
TEST 8 – MULTIPLE FAIL MATCHES .....	45
TEST 9 – SINGLE SUCCESS MATCH .....	45
<b>SPSM_FILTER_RESULTS_TEST_CASES.JAVA.....</b>	<b>46</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>46</b>
<b>EXPECTED RESULTS .....</b>	<b>46</b>
TEST 1 – SUCCESS SINGLE CALL.....	46
TEST 2 – FAIL WITH LIMIT .....	46
TEST 3 – SUCCESS MULTI CALL.....	46
TEST 4 – FAIL SINGLE CALL .....	47
TEST 5 – SUCCESS WITH LIMIT .....	47
<b>REPAIR_SCHEMA_TEST_CASES.JAVA .....</b>	<b>48</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>48</b>

<b>EXPECTED RESULTS .....</b>	<b>48</b>
TEST 1.1.....	48
TEST 5.6.....	48
TEST 5.7.....	48
<b>CREATE_QUERY_TEST_CASES.JAVA .....</b>	<b>49</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>49</b>
<b>EXPECTED RESULTS .....</b>	<b>49</b>
TEST 1.1 – SEPA QUERY.....	49
TEST 1.2 – SEPA QUERY.....	49
TEST 1.3 – SEPA QUERY.....	50
TEST 1.4 – SEPA QUERY.....	50
TEST 1.5 – SEPA QUERY.....	50
TEST 1.6 – SEPA QUERY.....	50
TEST 1.7 – SEPA QUERY.....	50
TEST 1.8 – SEPA QUERY.....	51
TEST 1.9 – SEPA QUERY.....	51
TEST 2.1 – DBPEDIA QUERY .....	51
TEST 2.2 – DBPEDIA QUERY .....	51
TEST 2.3 – DBPEDIA QUERY .....	52
TEST 2.4 – DBPEDIA QUERY .....	52
TEST 2.5 – DBPEDIA QUERY .....	52
TEST 2.6 – DBPEDIA QUERY .....	52
TEST 2.7 – DBPEDIA QUERY .....	52
TEST 2.8 – DBPEDIA QUERY .....	53
TEST 2.9 – DBPEDIA QUERY .....	53
TEST 2.10 – DBPEDIA QUERY .....	53
TEST 2.11 – DBPEDIA QUERY .....	53
TEST 2.12 – DBPEDIA QUERY .....	53
TEST 2.13 – DBPEDIA QUERY .....	54
<b>RUN_QUERY_TEST_CASES.JAVA .....</b>	<b>55</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>55</b>
<b>EXPECTED RESULTS .....</b>	<b>55</b>
TEST 1.1 – SEPA QUERY.....	55
TEST 1.2 – SEPA QUERY.....	55
TEST 1.3 – SEPA QUERY.....	55
TEST 1.4 – SEPA QUERY.....	55
TEST 1.5 – SEPA QUERY.....	56
TEST 1.6 – SEPA QUERY.....	56
TEST 1.7 – SEPA QUERY.....	56
TEST 1.8 – SEPA QUERY.....	56
TEST 1.9 – SEPA QUERY.....	56
TEST 1.10 – SEPA QUERY.....	56
TEST 2.1 – DBPEDIA QUERY .....	56
TEST 2.2 – DBPEDIA QUERY .....	56

TEST 2.3 – DBPEDIA QUERY .....	56
TEST 2.4 – DBPEDIA QUERY .....	57
TEST 2.5 – DBPEDIA QUERY .....	57
TEST 2.6 – DBPEDIA QUERY .....	57
TEST 2.7 – DBPEDIA QUERY .....	57
TEST 2.8 – DBPEDIA QUERY .....	57
TEST 2.9 – DBPEDIA QUERY .....	57
TEST 2.10 – DBPEDIA QUERY .....	57
TEST 2.11 – DBPEDIA QUERY .....	57
TEST 2.12 – DBPEDIA QUERY .....	57
TEST 2.13 – DBPEDIA QUERY .....	57
TEST 2.14 – DBPEDIA QUERY .....	58
TEST 2.15 – DBPEDIA QUERY .....	58
<b>SCHEMA_FROM_QUERY_TEST_CASES.JAVA .....</b>	<b>59</b>
<b>BASIC FUNCTIONALITY .....</b>	<b>59</b>
<b>EXPECTED RESULTS .....</b>	<b>59</b>
TEST 1.1.....	59
TEST 1.2.....	59
TEST 1.3.....	59
TEST 1.4.....	59
TEST 1.5.....	59
TEST 1.6.....	59
TEST 1.7.....	59
TEST 1.8.....	59
TEST 1.9.....	60
TEST 1.10.....	60
TEST 2.1.....	60
TEST 2.2.....	60
TEST 2.3.....	60
TEST 2.4.....	60
TEST 2.5.....	60
TEST 2.6.....	60
TEST 2.7.....	60
TEST 2.8.....	61
TEST 2.9.....	61
TEST 2.10.....	61
TEST 2.11.....	61
TEST 2.12.....	61
TEST 2.13.....	61

## OBJECTIVES OF CHAIN

The CHAIn (Combining Heterogeneous Agencies' Information) system dynamically re-writes queries to databases when mismatches led to query failure.

In general, queries to databases only succeed if they are correctly writing according to the schema of that database, which is only possible if the schema of the database is known in advance. During an emergency response (and, indeed, many other situations), this is plausible in some situations, but emergencies are characterised by their unpredictability and such foresight is not always possible. If one wishes to broadcast a query to all relevant agencies, or to a new agency, or to a previous collaborator who has updated their data sources, for example, a query may fail even if there is relevant data in the data source.

CHAIn sits locally within an agency. When a query to the data sources of that agency fails, CHAIn will use various matching techniques to determine on-the-fly whether there is mismatch between the schema of the query and the schema of the database and, if so, whether there is anything in the database that matches, or approximately matches the query. If so, CHAIn rewrites the query according to the schema of the database and sends back an appropriate response (or responses), together with a score indicating how good the match is and information about the assumptions and approximations made in the match.

## SETTING UP CHAIN

1. After downloading the CHAI project, run the following four files that can be found in *spsm/s-match-source* called *0cloneSMatch*, *2buildSMatch*, *3createBinary*, *4extractBinary*.

Doing this should create the following folders in that directory: *s-match-parent*, *s-match-core*, *s-match-io*, *s-match-logic*, *s-match-nlp*, *s-match-nlp-annotation*, *s-match-nlp-opennlp*, *s-match-spsm*, *s-match-wordnet*, *s-match-examples*, *s-match-utils*, *s-match-spsm-prolog*.

2. Open Eclipse and select *File > New > Project > Java Project* and enter the name for the project. Uncheck the *Use Default Location* radio button and set the location to point to where this project was downloaded to and click *Next*.
3. Select the *Libraries* tab and then click on *Add External JARS*, navigate to where this project was downloaded and into *spsm/s-match/lib* and select all the files in this directory and select *Open*. The files in *jena/apache-jena-2.12.1/lib* should also be added in the same way.
4. Then click on the *Add Library* button and make sure that the *JRE System Library* & *JUnit 4* libraries have been added to the project.

**Note:** If you want to access the libraries imported into the project after set-up then you should right click on the project in the *Package Explorer* on the left-hand side of Eclipse and choose *Build Path/Configure Build Path...*

5. Finally, click on *Finish*. You should now have access to all the source and testing files for this project.



## RUNNING FILES IN CHAIN

1. Select the file that you want to run by double clicking on the file in the *Package Explorer* panel on the left.
2. To run, click on the green start button on the top panel of Eclipse and there should be appropriate messages printed to the console to let you know what is happening.

**Note:** There are some files that will not do anything when you try to run them. You can check what should happen when you run a specific file by looking up that file in this document.

## FILE STRUCTURE

TBC

## TESTING RESULT DOCUMENTS

All tests will write their results to a text file that can be found in the *outputs/testing* folder. These files have been structured to give details about each individual test including what schemas CHAI is being run with, what the expected result is and what the actual result is that is returned.

**Note:** When running the testing files for the first time you might need to select the option to run the file using *JUnit*.

## BREAKDOWN OF TESTING DOCUMENTS

### Task 1 Testing Documents

- *SPSM\_Test\_Cases.java* (which produces *Call\_SPSM\_Tests.txt*)
- *Match\_Results\_Test\_Cases.java* (which produces *Filter\_Limit\_Tests.txt*)
- *SPSM\_Filter\_Results\_Test\_Cases.java* (which produces *Task\_1\_Tests.txt*)

These files can be run at once by running *Task1\_Test\_Suite.java*

### Task 2 Testing Documents

- *Repair\_Schema\_Test\_Cases.java* (which produces *Schema\_Repair\_Tests.txt*)
- *Create\_Query\_Test\_Cases.java* (which produces *Create\_Queries\_Tests.txt*)
- *Run\_Query\_Test\_Cases.java* (which produces *Run\_Queries\_Tests.txt*)

These files can be run at once by running *Task2\_Test\_Suite.java*

### Task 4 Testing Documents

- *Schema\_From\_Query\_Test\_Cases.java* (which produces *Schema\_From\_Query\_Tests.txt*)

# DESCRIPTION OF IMPLEMENTATION FILES

## MATCH\_STRUC.JAVA

### BASIC FUNCTIONALITY

Data structure that has been created to store the results that are returned after calling SPSM. The structure allows for easy interaction to find out information about the match details with a certain target schema.

### DESIGN

This java file represents a class that is responsible for creating an object that is used to store details about a schema/query to allow repairs to take place to successfully query a dataset.

### PRIVATE VARIABLES/FIELDS

#### ***similarity***

This is the variable that stores the similarity value between the source and this current target schema. (double)

#### ***dataset***

This variable holds the name of the target schema that this structure is representing the results for. (String)

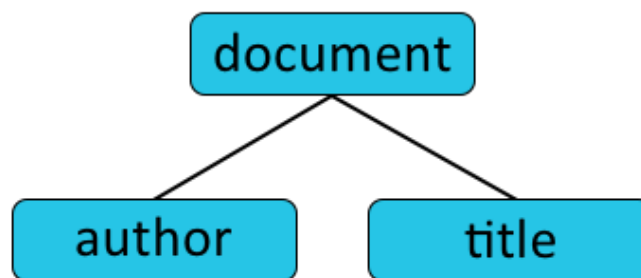
#### ***matches***

List of arrays containing the match concepts between the source and this current target schema. (ArrayList of String arrays)

### ***repairedSchemaTree***

Tree structure that represents the repaired schema in a tree form. (Node)

**For example:** If the schema is *document(title,author)* then the tree will look like the following where *document* is the root node with two child nodes with the values *author* and *title*,



### ***repairedSchema***

This variable holds the repaired schema in a format that can easily be read. (String)

### ***numMatches***

Stores the total number matches which is calculated by the number of elements in the matches ArrayList. (int)

### ***query***

Stores the query that has been produced using the repaired schema (*String*)

## METHODS

### ***public Match\_Struc (double sim, String targetSchema)***

Constructor that is responsible for initialising the Match\_Struc variable with the similarity value and the name of the target schema.

### ***public void setSimilarity (double sim)***

Sets the similarity of this match to be the value passed in as the parameter, *sim*.

### ***public void setDatasetSchema (String targetSchema)***

Sets the target schema to be the String that is passed in as the parameter, *targetSchema*.

### ***public void addMatch (String[] match)***

Responsible for adding another match concept to the list of matches stored in the ArrayList<String[]> called *matches*.

### ***public int getNumMatches ()***

Returns the total number of matches based on the number of elements inside *matches*.

### ***public double getSimValue ()***

Returns the similarity value for that target schema.

### ***public String getDatasetSchema ()***

Returns the name of the target schema.

***public ArrayList<String[]> getMatches ()***

Returns the ArrayList of arrays containing the match concept details.

***public String[] getMatchAtIndex (int index)***

Returns the specific match concept detail at the specified index passed as a parameter, *index*.

***public void setRepairedSchemaTree (Node schemaTree)***

Sets the repaired schema tree from the parameter, *schemaTree*.

***public void setRepairedSchema (String stringSchema)***

Sets the String version of the repaired schema from the parameter, *stringSchema*.

***public String getRepairedSchema ()***

Returns the String version of the repaired schema.

***public Node getRepairedSchemaTree ()***

Returns the repaired schema as a tree structure.

***public void setQuery(String matchQuery)***

Sets the query for this match structure.

***public String getQuery()***

Returns the query for this structure.



## ASSOCIATED TESTING DOCUMENTS

*SPSM\_Test\_Cases.java* (which produces *Call\_SPSM\_Tests.txt*)

*Match\_Results\_Test\_Cases.java* (which produces *Filter\_Limit\_Test.txt*)

*SPSM\_Filter\_Results\_Test\_Cases.java* (which produces *Task\_1\_Tests.txt*)

*Task1\_Test\_Suite.java* (which runs the testing files for task 1)

## RUNNING THIS FILE

-

## CALL\_SPSM.JAVA

### BASIC FUNCTIONALITY

This class is responsible for initially taking in both the target and source schemas before calling SPSM with these schemas. It will then store the results as an ArrayList of Match\_Struc objects.

### DESIGN

#### Inputs

- ArrayList of type Match\_Struc objects where you want the results to be stored
- String version of source schema
- String version of target schema

#### Process

- Calls SPSM with the source and target schema, then picks out the returned results and stores as Match\_Struc objects into the ArrayList that was passed in as an input

#### Outputs

- ArrayList of type Match\_Struc objects containing the results from calling SPSM **OR** null if there has been a problem with calling SPSM

### PRIVATE VARIABLES/FIELDS

#### ***targetList***

Used temporarily for splitting the target schema String if there are more than one target schema within the String. This allows the application to then call SPSM for each individual target schema. (*String[]*)

## METHODS

***public ArrayList<Match\_Struc> getSchemas (ArrayList<Match\_Struc> results, String sourceSchema, String targetSchema)***

Retrieve the schemas from the user, either through the console or as a parameter which gets written to their respective files. From there we can start to make a call to SPSM with the current target.

***public ArrayList<Match\_Struc> makeCallToSPSM (ArrayList<Match\_Struc> results, String currTarget)***

Makes the call to SPSM using a bash file called *call-spsm.sh*. This should allow for SPSM to run and then write the results out as a serialised object to *serialised-results.ser*. It will then call the appropriate method to read this file and store the results in a Match\_Struc object.

***public ArrayList<Match\_Struc> recordSerialisedResults (ArrayList<Match\_Struc> results, String targetSchema)***

Responsible for reading the serialised object from *serialised-results.ser* and calls the appropriate method that will then go through the different pieces of information stored within this object and stores what is required for CHAI.

***public ArrayList<Match\_Struc> readObject (ArrayList<Match\_Struc> results, String targetSchema, IContextMapping<INode> mapping)***

Reads the information stored in the object read in and picks out the data that is required for use by CHAI.

***public String getNodePathString (INode node)***

Responsible for taking in a *INode* object and turns it into something readable so that we can store the match concepts in a convenient way.

## ASSOCIATED TESTING DOCUMENTS

*SPSM\_Test\_Cases.java* (which produces *Call\_SPSM\_Tests.txt*)

*SPSM\_Filter\_Results\_Test\_Cases.java* (which produces *Task\_1\_Tests.txt*)

*Task1\_Test\_Suite.java* (which runs the testing files for task 1)

## RUNNING THIS FILE

Running this file will call SPSM with the source and target schemas that have been declared in the *main* method and will then print out the results that have been returned from SPSM by printing out the target schema, it's similarity to the source schema and the match concepts.

## BEST\_MATCH\_RESULTS.JAVA

### BASIC FUNCTIONALITY

Responsible for taking results after calling SPSM and then filtering to ensure each result is over the determined threshold, cut the number of results to the desired  $n$  and sort them so that the match with the highest similarity is at the top of the list.

### DESIGN

#### Inputs

- ArrayList of type Match\_Struc containing a list of matches returned from calling SPSM
- The threshold that you want the similarity value of the results to have as a minimum
- The maximum number of results you want returned

#### Process

- After taking in the list of results after calling SPSM, we remove any elements in the list that are not equal to or greater than the threshold value and then sort the list so that elements are in descending order based on their similarity value. Finally, the list is restricted in size based on the maximum number of elements that were requested

#### Outputs

- ArrayList of Match\_Struc objects that have been filtered based on the threshold value and limit that has been passed in

## PRIVATE VARIABLES/FIELDS

-

## METHODS

***public ArrayList<Match\_Struc> getThresholdAndFilter (ArrayList<Match\_Struc> results, Double threshVal, int limNum)***

Start off by taking in the results, threshold value and the number of results wanted but if these values haven't been passed in through parameters, then ask the user to enter them through the console. It will then strip off any matches that are lower than the threshold value that has been entered.

***public ArrayList<Match\_Struc> sortResultingMatches (ArrayList<Match\_Struc> filteredRes, int limitNo)***

This method will sort the resulting matches after they have been filtered to ensure that only those greater than or equal to the threshold value are left. These matches are sorted so that the matches with the highest similarity value is at the top of the list.

***public void displayResults (ArrayList<Match\_Struc> res)***

This method is used to print out the results after they have been filtered and sorted.

## ASSOCIATED TESTING DOCUMENTS

*Match\_Results\_Test\_Cases.java* (which produces *Filter\_Limit\_Tests.txt*)

*SPSM\_Filter\_Results\_Test\_Cases.java* (which produces *Task\_1\_Tests.txt*)

*Task1\_Test\_Suite.java* (which runs the testing files for task 1)

## RUNNING THIS FILE

Running this file will use the `Match_Struc` variables in the list of results and filter the list to ensure that anything in the list is over a specified threshold and that there are a number of entries if specified by the user. The *main* method will then print out each element in the list after filtering and sorting the list.

## REPAIR\_SCHEMA.JAVA

### BASIC FUNCTIONALITY

Responsible for taking in an ArrayList of Match\_Struc objects and going through each one and creating a repaired schema based on the relation between the original source and target schemas.

### DESIGN

#### Inputs

- ArrayList of Match\_Struc objects

#### Process

- For each of the elements in the list passed in, create a repaired schema based on the details from comparing the source and target schemas after calling SPSM

#### Outputs

- ArrayList of Match\_Struc objects where each element now has a repaired schema attribute/field

### PRIVATE VARIABLES/FIELDS

-



## METHODS

***public ArrayList<Match\_Struc> prepare (ArrayList<Match\_Struc> matchRes)***

Takes in the ArrayList of objects and starts processing them in turn to create a repaired schema.

***public Match\_Struc repairSchema(Match\_Struc matchStructure)***

Creates a tree structure of the schema and adds that as a field to the Match\_Struc object based on the relation between the source and target schemas. It also adds a String version of the repaired schema as a field to the Match\_Struc object.

***public Node modifyRepairedTree(Node root, String concept)***

Using the relation concept data, the tree is modified to ensure that there are only links to parts of the schema that we think there is a match with. This tree is then returned.

## ASSOCIATED TESTING DOCUMENTS

*Repair\_Schema\_Test\_Cases.java* (which produces *Schema\_Repair\_Tests.txt*)

*Create\_Query\_Test\_Cases.java* (which produces *Create\_Queries\_Tests.txt*)

## RUNNING THIS FILE

Running this file will go take in the source and target schemas, call SPSM and then create the repaired schema based on the match details between these two original schemas. The results will be printed to the console.

# NODE.JAVA

## BASIC FUNCTIONALITY

Structure that is being used to store the repaired schema tree that consists of a node that has a value and a list of children nodes.

## DESIGN

This java file represents a class that is responsible for creating a tree object that is used to store the breakdown of a schema.

## PRIVATE VARIABLES/FIELDS

### ***value***

Holds the value of the node such as the predicate or parameter name (*String*)

### ***children***

The list of nodes that are the children of the current node (*List<Node>*)

## METHODS

### ***public Node(String val)***

The constructor that creates the node structure and sets the value of that node.

### ***public void addChild(Node child)***

Adds the node, *child*, to the list of children nodes.

***public String getValue()***

Returns the value of the current node as a String.

***public List<Node> getChildren()***

Returns the list of children nodes.

***public ArrayList<String> getChildrenValues()***

Returns a list containing the values of all the children nodes.

***public Boolean hasChildren()***

Returns whether or not the node has any children nodes.

***public int getNumChildren()***

Returns the total number of children nodes.

***public String printTree()***

Prints out the tree as a repaired schema.

***public String printChildren(List<Node> childrenList)***

Prints out the children in the tree to create the repaired schema as a *String*.

***public void addToTree(String[] paramParts)***

Modifies the tree to add a list of parameters correctly into the current tree.

## ASSOCIATED TESTING DOCUMENTS

*Repair\_Schema\_Test\_Cases.java* (which produces *Schema\_Repair\_Tests.txt*)

## RUNNING THIS FILE

-

## ONTOLOGY\_STRUC.JAVA

### BASIC FUNCTIONALITY

Structure that is used to store the ontology structure being used to create the queries.

### DESIGN

This java file represents a class that is responsible for storing the details of an ontology file to allow for other methods to access whilst creating queries.

### PRIVATE VARIABLES/FIELDS

#### ***name***

The name of the ontology structure (*String*)

#### ***link***

The web link of the ontology structure (*String*)

#### ***properties***

The list of key words or properties associated with the structure (*String[]*)

### METHODS

#### ***public Ontology\_Struc(String ontName, String ontLink, String[] ontProperties)***

Constructor for creating an ontology structure.

***public void setName(String ontName)***

Sets the name of the structure.

***public String getName()***

Returns the name of the structure.

***public void setLink(String ontLink)***

Sets the link for the structure.

***public String getLink()***

Returns the link of the structure.

***public void setProperties(String[] ontProperties)***

Sets the properties or key words associated with the structure.

***public String[] getProperties()***

Returns the list of key words or properties associated with the structure.

## ASSOCIATED TESTING DOCUMENTS

*Create\_Query\_Test\_Cases.java* (which produces *Create\_Queries\_Tests.txt*)

## RUNNING THIS FILE

-

## CREATE\_QUERY.JAVA

### BASIC FUNCTIONALITY

Responsible for using the repaired schema to create either a sepa or dbpedia query, this query is then added as a field onto the Match\_Struc object structure.

### DESIGN

#### Inputs

- ArrayList of Match\_Struc objects with elements containing repaired schemas that you want to create queries for
- The query type as a string (either sepa or dbpedia)
- The directory of the dataset if local as a string otherwise null
- The maximum number of results you want returned

#### Process

- For each of the elements in the list of Match\_Struc objects, we create either a sepa or dbpedia query based on the repaired schema that is stored as a field of a Match\_Struc element. This query is then stored as an attribute/field

#### Outputs

- ArrayList of Match\_Struc with elements now containing a query as a string

### PRIVATE VARIABLES/FIELDS

-

## METHODS

***public ArrayList<Match\_Struc> createQueryPrep(ArrayList<Match\_Struc> matchRes, String queryType, String additionalInfo)***

Sets up so that the methods can create a query for each of the repaired schemas in the list of Match\_Struc objects.

***public String createSepaQuery(Match\_Struc matchDetails, String datafileDir, int noResults)***

Creates structure for a local sepa query.

***public String createDbpediaQuery(Match\_Struc matchDetails, int noResults)***

Creates structure for a remote dbpedia query.

***public ArrayList<Ontology\_Struc> setupPrefixes(JSONArray jsonArr, ArrayList<Ontology\_Struc> ontologies)***

Reads ontology file and processes it into Ontology\_Struc objects.

***public String writePrefix(ArrayList<Ontology\_Struc> ontologies)***

Writes prefixes for query through reading the Ontology\_Struc objects.

***public String dataMatching(List<Node> children, ArrayList<Ontology\_Struc> ontologies)***

Finds what ontology file a specific key word or property is from or relates to.



## ASSOCIATED TESTING DOCUMENTS

*Create\_Query\_Test\_Cases.txt* (which produces *Create\_Queries\_Tests.txt*)

## RUNNING THIS FILE

Running this file will setup a sepa or dbpedia query based on the repaired schema produced from calling SPSM on the target and source schemas. These details are specified in the *main* method. It also prints out the repaired schema and the query string created.

## RUN\_QUERY.JAVA

### BASIC FUNCTIONALITY

Responsible for running either a sepa or dbpedia query.

### DESIGN

#### Inputs

- Match\_Struc object containing the query as a string that you want to run
- The type of query as a string (either sepa or dbpedia)
- The directory of the dataset as a string if local otherwise null

#### Process

- The query stored as a field on the Match\_Struc object that has been passed in gets extracted and then the query gets setup and run with the details of this dependent on whether it is a sepa or dbpedia query

#### Outputs

- Results from running the query or error messages if the query was unable to run are printed to the console

### PRIVATE VARIABLES/FIELDS

-

## METHODS

**public boolean runQuery(Match\_Struc current, String queryType, String datasetDir)**

Responsible for selecting the correct method for running either a sepa or dbpedia query based on the *queryType* parameter that has been passed in.

**public boolean runSepaQuery(String query, String datasetToUseDir, Match\_Struc currMatchStruc)**

Responsible for running the sepa query that has been passed into this method.

**public boolean runDbpediaQuery(String query, Match\_Struc currMatchStruc)**

Responsible for running the dbpedia query that has been passed into this method.

## ASSOCIATED TESTING DOCUMENTS

*Run\_Query\_Test\_Cases* (which produces *Run\_Queries\_Test.txt*)

## RUNNING THIS FILE

Running this file will run either a sepa or dbpedia query depending on the setup in the *main method*. It also prints out the query that it is trying to run followed by the results if the query has run successfully with data returned.

## SCHEMA\_FROM\_QUERY.JAVA

### BASIC FUNCTIONALITY

Responsible for taking in either a sepa or dbpedia query that is valid and creating a CHAIn schema for that specific query.

### DESIGN

#### Inputs

- Query as a string
- The type of the query as a string (either sepa or dbpedia)

#### Process

- Parts of the query are extracted to create a schema represented as a tree which is then stored in a Match\_Struc object

#### Outputs

- Match\_Struc containing the details of the schema created from the query that was originally passed in

### PRIVATE VARIABLES/FIELDS

-

## METHODS

### **public Match\_Struc getSchemaFromQuery(String query, String queryType)**

Initial method that gets passed in the query as a string and the type of query it is (either sepa or dbpedia). This information is then used to determine whether we need to parse a sepa or dbpedia query and calls methods accordingly to get this information. Also responsible for returning the result as a Match\_Struc object with the schema stored as the repaired schema string and also represented as a tree.

### **public Match\_Struc schemaFromSepaQuery(String query, Match\_Struc res)**

Responsible for taking in the sepa query and using parts of this query in order to create a schema specific for it. This schema is then added to the *res* parameter as a string and represented as a tree before returning it.

### **public Match\_Struc schemeFromDbpediaQuery(String query, Match\_Struc res)**

Responsible for taking in the dbpedia and using parts of this query in order to create a schema specific for it. This schema is then added to the *res* parameter as a string and represented as a tree before returning it.

### **public Node createTreeFromSchemaString(String schemaStr)**

Takes in the schema represented as a string and parses this string to create the tree representation of the schema which then gets returned.

## ASSOCIATED TESTING DOCUMENTS

*Schema\_From\_Query\_Test\_Cases.java* (which produces  
*Schema\_From\_Query\_Tests.txt*)

## RUNNING THIS FILE

Running this file will create a schema for either a sepa or dbpedia query depending on the setup in the *main method*. It then prints out the schema that has been created.

# DESCRIPTION OF TESTING FILES

## SPSM\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing the output results from *Call\_SPSM.java* which takes in source and target schemas and then calls SPSM before reading in results as a serialised object.

After running this test, the results will have been documented in *outputs/testing/Call\_SPSM\_Test.txt* where you will be able to see the test number, the target and source schemas that SPSM has been called with, the expected result and the actual result. This test should be hand checked.

### EXPECTED RESULTS

---

#### TEST 1.1

This test calls SPSM with empty source and target schemas, therefore, the size of the results returned is 0.

---

#### TEST 2.1

This test calls SPSM with a source schema but no target schema, therefore, the size of the results returned is 0.

---

#### TEST 2.2

This test calls SPSM with a target schema but no source schema, therefore, the size of the results returned is 0.

---

#### TEST 2.3

This test calls SPSM with the same source and target schema, *author(name)*. Therefore, the similarity value returned should be 1.0 with 2 individual matches.



---

#### TEST 2.4

This test calls SPSM with a single source schema and three target schemas, where only two of these target schemas have a level of similarity with the source schema. Where *author(name, document)* has a similarity of 1.0 and 2 individual matches and *reviewAuthor(firstname, lastname, review)* has a similarity of 0.75 and 2 individual matches.

---

#### TEST 3.1

This test calls SPSM with a single source schema and two target schemas, however, there are no matches between the source and target schemas. Therefore, the size of the results returned is 0.

---

#### TEST 3.2

This test calls SPSM with a single source schema and two target schemas where only one of these target schemas matches with the source. So only one of the targets has match data. This target is *author(name)* that has a similarity of 1.0 and 2 individual matches.

---

#### TEST 3.3

This test calls SPSM with a single source schema and four target schemas where only three of these target schemas matches with the source. Therefore, we have three matches in our list of results. Where *author(name)* has a similarity of 1.0 and 2 individual matches, *paperWriter(firstname, surname, paper)* has a similarity of 0.5 and 2 individual matches and *reviewAuthor(firstname, lastname, review)* has a similarity of 0.75 and 2 individual matches.

---

#### TEST 4.1

This test calls SPSM with a single source and target schema that has a similarity value of 1.0 and 1 individual match.

---

#### TEST 4.2

This test calls SPSM with a single source and target schema that do not have any similarity, therefore, the size of the results returned is 0.

---

#### TEST 4.3

This test calls SPSM with a single source and target schema that do not have any similarity, therefore, the size of the results returned is 0.

---

#### TEST 4.4

This test calls SPSM with a single source and target schema that have a similarity value of 0.5 and 2 individual matches.

---

#### TEST 4.5

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

---

#### TEST 5.1

This test calls SPSM with a single source and target schema where it was expected that there would be a similarity of 1.0 returned, however, a similarity of 0.5 was returned with the expected 5 individual matches.

---

#### TEST 5.2

This test calls SPSM with a single source and target schema that returns a similarity of 0.6 and 5 individual matches.

---

#### TEST 5.3

This test calls SPSM with a single source and target schema that do not have a similarity, therefore, the size of the results is 0.

---

#### TEST 5.4

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

---

#### TEST 5.5

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

---

**TEST 5.6**

This test calls SPSM with a single source and target schema where the similarity between the two is 0.625 and 12 individual matches.

---

**TEST 5.7**

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

---

**TEST 5.8**

This test calls SPSM with a single source and target schema where there are no similarities between the two, therefore, the size of the results returned is 0.

---

**TEST 6.1**

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

**TEST 6.2**

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

**TEST 6.3**

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

**TEST 6.4**

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

**TEST 6.5**

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

#### TEST 6.6

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

---

#### TEST 6.7

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

## MATCH\_RESULTS\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing *Best\_Match\_Results.java* file to make sure that when we pass in the results that we have received from SPSM, we get returned only results that have similarity values over the threshold value that are sorted and that we only have n number of results as requested.

After running this test, the results will have been documented in *outputs/testing/Filter\_Limit\_Test.txt* where you will be able to see the test number, what the threshold value and limit values are that *Best\_Match\_Results.java* got called with, the expected result and the actual result. This test should be hand checked.

### EXPECTED RESULTS

---

#### TEST 1 – FAIL WITH LIMIT

This test calls *Best\_Match\_Results.java* with a threshold value of 1.0 and a limit of 5, therefore, the size of the results is 0 because there are no entries matching this criteria.

---

#### TEST 2 – MULTIPLE SUCCESSES MATCH

This test calls *Best\_Match\_Results.java* with a threshold value of 0.6 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 2 afterwards meaning that there are 2 entries with a threshold greater than or equal to 0.6.

---

#### TEST 3 – MULTIPLE SUCCESSES MATCH

This test calls *Best\_Match\_Results.java* with a threshold value of 0.2 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes

the size of the results to be 4 afterwards meaning that there are 4 entries with a threshold greater than or equal to 0.2.

---

#### TEST 4 – SUCCESS WITH LARGE LIMIT

This test calls *Best\_Match\_Results.java* with a threshold value of 0.2 and a limit of 5 which causes the size of the results to be 4 afterwards meaning that there are 4 entries with a threshold values greater than or equal to 0.2.

---

#### TEST 5 – SINGLE FAIL MATCH

This test calls *Best\_Match\_Results.java* with a threshold value of 0.5 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 0.5.

---

#### TEST 6 – EMPTY MATCHES

This test calls *Best\_Match\_Results.java* with a threshold value of 0.2 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 0.5.

---

#### TEST 7 – SUCCESS WITH LIMIT

This test calls *Best\_Match\_Results.java* with a threshold value of 0.2 and a limit of 3 which causes the size of the results to be 3 afterwards meaning that there are at least 3 entries with a threshold value greater than or equal to 0.2.

---

#### TEST 8 – MULTIPLE FAIL MATCHES

This test calls *Best\_Match\_Results.java* with a threshold value of 1.0 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 1.0.

---

#### TEST 9 – SINGLE SUCCESS MATCH

This test calls *Best\_Match\_Results.java* with a threshold value of 0.1 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 1 afterwards meaning that there was 1 entry with a threshold value greater than or equal to 0.1.

## SPSM\_FILTER\_RESULTS\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing *Call\_SPSM.java* and *Best\_Match\_Results.java* to make sure that after calling SPSM and filtering the results based on the restrictions entered, we receive the appropriate results afterwards.

After running this test, the results will have been documented in *outputs/testing/Task\_1\_Tests.txt* where you will be able to see the test number, the source and target schemas that SPSM was called with, what the threshold value and limit values are that *Best\_Match\_Results.java* got called with, the expected result and the actual result. This test should be hand checked.

### EXPECTED RESULTS

---

#### TEST 1 – SUCCESS SINGLE CALL

This test calls SPSM with a single source and target schema that are the same and filters the results with a threshold value of 0.6 and a limit of 0 (meaning that there are no restrictions on the number of results returned). This causes the size of the results to be 1 meaning that there is only 1 result returned from SPSM that matches the filtering criteria.

---

#### TEST 2 – FAIL WITH LIMIT

This test calls SPSM with a single source and target schema and filters the results with a threshold of 0.0 and a limit of 2. This causes the size of the results to be 0 meaning that there are no results returned from SPSM that matches the filtering criteria.

---

#### TEST 3 – SUCCESS MULTI CALL

This test calls SPSM with a single source and 4 target schemas and then filters the results with a threshold value of 0.6 and a limit of 0 (meaning that there are no

restrictions on the number of results returned). This causes the size of the results to be 2 meaning that there are 2 results that have been returned from SPSM that matches the filtering criteria.

---

#### TEST 4 – FAIL SINGLE CALL

This test calls SPSM with a single source and target schema and then filters the results with a threshold value of 0.0 and a limit of 0 (meaning that there are no restrictions on the number of results returned). This causes the size of the results to be 0 meaning that there are 0 results that have been returned from SPSM that matches the filtering criteria.

---

#### TEST 5 – SUCCESS WITH LIMIT

This test calls SPSM with a single source and 4 target schemas and then filters the results with a threshold value of 0.0 and a limit of 2. This causes the size of the results to be 2 meaning that there are 2 results that have been returned from SPSM that matches the filtering criteria.



### BASIC FUNCTIONALITY

Responsible for testing the element of CHAIn that after calling SPSM and filtering the results will try to create a repaired schema based on the match data between the source and target schemas.

After running this test, the results will have been documented in *outputs/testing/Schema\_Repair\_Tests.txt* where you will be able to see the test number, the source and target schemas that SPSM was called with, the expected result and the actual result. This test should be hand checked.

### EXPECTED RESULTS

---

#### TEST 1.1

This test calls SPSM with a single source and target schema with the expected repaired schema to be *car(colour, brand)*. However, this is an example that causes SPSM to crash causing no data to be returned to CHAIn so we have null results.

---

#### TEST 5.6

This test calls SPSM with a single source and target schema with the expected repaired schema to be *conference(paper(title, document(date(day, month, year), writer(name(first, second)))))*. The actual repaired schema that gets generated does match this, however, it should be noted that the parameters are in a slightly different order but still have the same meaning.

---

#### TEST 5.7

This test calls SPSM with a single source and target schema with the expected repaired schema to be *conference(paper(title, document(writer(name(first, second)))))*. However, this is an example that causes SPSM to crash causing no data to be returned to CHAIn so we have null results.

## CREATE\_QUERY\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing *Create\_Query.java* to ensure that after calling SPSM and repairing the target schema we can create the appropriate sepa or dbpedia query.

After running this test, the results will have been documented in *outputs/testing/Create\_Queries\_Tests.txt* where you will be able to see the test number, the schema that the query is being created based on, the expected result and the actual result.

### EXPECTED RESULTS

---

#### TEST 1.1 – SEPA QUERY

This test creates a sepa query based on the schema,

*waterBodyPressures(dataSource, identifiedDate, affectsGroundwater, waterBodyId)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.2 – SEPA QUERY

This test creates a sepa query based on the schema,

*water(timePeriod, geo, measure, resource)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.3 – SEPA QUERY

This test creates a sepa query based on the schema,

*waterBodyMeasures(timePeriod, geo, measure, resource)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.4 – SEPA QUERY

This test creates a sepa query based on the schema,

*waterBodyPressures(identifiedDate, waterBodyId, assessmentCategory, source)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.5 – SEPA QUERY

This test creates a sepa query based on the schema,

*waterBodyMeasures(waterBodyId, secondaryMeasure, dataSource)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.6 – SEPA QUERY

This test creates a sepa query based on the schema,

*surfaceWaterBodies(riverName, associatedGroundwaterId)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.7 – SEPA QUERY

This test creates a sepa query based on the schema,

*bathingWaters(catchment, localAuthority, lat, long)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.8 – SEPA QUERY

This test creates a sepa query based on the schema,

*surfaceWaterBodies(subBasinDistrict, riverName, altitudeTypology, associatedGroundwaterId)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 1.9 – SEPA QUERY

This test creates a sepa query based on the schema,

*bathingWaters(bathingWaterId)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 2.1 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*City(country, populationTotal)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 2.2 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Country*

This creates the appropriate query based on this schema.

---

#### TEST 2.3 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Astronaut(nationality)*

This creates the appropriate query based on this schema.

---

#### TEST 2.4 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Mountain(elevation)*

This creates the appropriate query based on this schema.

---

#### TEST 2.5 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Person(occupation, birthPlace)*

This creates the appropriate query based on this schema.

---

#### TEST 2.6 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Person(occupation, instrument)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 2.7 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Cave(location)*

This creates the appropriate query based on this schema.

---

#### TEST 2.8 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*FormulaOneRacer(races)*

This creates the appropriate query based on this schema.

---

#### TEST 2.9 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*River(length)*

This creates the appropriate query based on this schema.

---

#### TEST 2.10 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Royalty(parent)*

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

---

#### TEST 2.11 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*river(length)*

This creates the appropriate query based on this schema which is an empty string because we cannot make a query from this schema.

---

#### TEST 2.12 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*Stream(length)*

This creates the appropriate query based on this schema which is an empty string because we cannot make a query from this schema.

---

#### TEST 2.13 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

*River(Mountain(elevation))*

This creates the appropriate query based on this schema.

## RUN\_QUERY\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing *Run\_Query.java* to ensure that after creating a query, we can run it without any errors.

After running this test, the results will have been documented in *outputs/testing/Run\_Queries\_Tests.txt* where you will be able to see the test number, the query that is being run and whether the query has run successfully. If the query has not run successfully then there will be an error message printed.

### EXPECTED RESULTS

---

#### TEST 1.1 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyPressures* data file.

---

#### TEST 1.2 – SEPA QUERY

This test does not successfully run the query. This is because the application fails to find the dataset being defined in the query, *water.n3*, which is not in the sepa data files so this is expected.

---

#### TEST 1.3 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyMeasures* data file.

---

#### TEST 1.4 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyPressures* data file.



---

#### TEST 1.5 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyMeasures* data file.

---

#### TEST 1.6 – SEPA QUERY

This test successfully runs a sepa query that queries the *surfaceWaterBodies* data file.

---

#### TEST 1.7 – SEPA QUERY

This test successfully runs a sepa query that queries the *bathingWaters* data file.

---

#### TEST 1.8 – SEPA QUERY

This test successfully runs a sepa query that queries the *surfaceWaterBodies* data file.

---

#### TEST 1.9 – SEPA QUERY

This test successfully runs a sepa query that queries the *bathingWaters* data file.

---

#### TEST 1.10 – SEPA QUERY

This test does not successfully run the query. This is because the application fails to find the dataset being defined in the query, *waterBodyTemperatures.n3*, which is not in the sepa data files so this is expected.

---

#### TEST 2.1 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.2 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.3 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.4 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.5 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.6 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.7 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.8 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.9 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.10 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.11 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

---

#### TEST 2.12 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

---

#### TEST 2.13 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

---

#### TEST 2.14 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

---

#### TEST 2.15 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

## SCHEMA\_FROM\_QUERY\_TEST\_CASES.JAVA

### BASIC FUNCTIONALITY

Responsible for testing *Schema\_From\_Query.java* to ensure that we can create a valid schema from either a sepa or dbpedia query.

### EXPECTED RESULTS

---

#### TEST 1.1

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.2

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.3

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.4

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.5

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.6

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.7

This test creates a schema from a valid sepa query and returns the correct schema.

---

#### TEST 1.8

This test creates a schema from a valid sepa query and returns the correct schema.

---

**TEST 1.9**

This test creates a schema from a valid sepa query and returns the correct schema.

---

**TEST 1.10**

This test attempts to create a schema from an invalid sepa query (one of the prefixes are missing) and therefore returns null as expected.

---

**TEST 2.1**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.2**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.3**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.4**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.5**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.6**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.7**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.8**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.9**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.10**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.11**

This test creates a schema from a valid dbpedia query and returns the correct schema. Note that the query passed in is an empty string, therefore, the schema should be an empty string.

---

**TEST 2.12**

This test creates a schema from a valid dbpedia query and returns the correct schema.

---

**TEST 2.13**

This test attempts to create a schema from an invalid dbpedia query (one of the prefixes are missing) and therefore returns null as expected.