API Specification Doc

(*SPARQL Named Query Ontology and Service*)

Version	Date	Description
1.0	05-05-2018	Running simple queries
1.1	25-06-2018	Running parameterized queries with multiple conditions

Index

Ontology

Introduction

Ontology in computer science is a formal representation of properties and relations of data, concepts and entities. Ontology in this project is stored in .ttl file. Terse RDF Triple Language (Turtle) is a file format which is used in RDF (Resource Description Framework) data model for expressing data. Semantic web modelling tool TopBraid Composer was used to create ontology for project.

IRI and prefixes

IRIs may be written as relative or absolute IRIs or prefixed names.

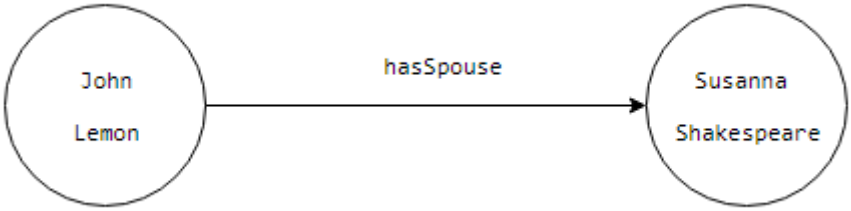
A prefixed name is a prefix label and a local part, separated by a colon ":". A prefixed name is turned into an IRI by concatenating the IRI associated with the prefix and the local part. The '@prefix' or 'PREFIX' directive associates a prefix label with an IRI.

TopBraid Composer gives ability to define own prefixes.
Examples of prefixes defined by user:

Name	Prefix
being	<http://tutorial.topbraid.com/being#> .
animal	<http://tutorial.topbraid.com/animal#> .
person	<http://tutorial.topbraid.com/person#> .
query	<http://tutorial.topbraid.com/query#> .

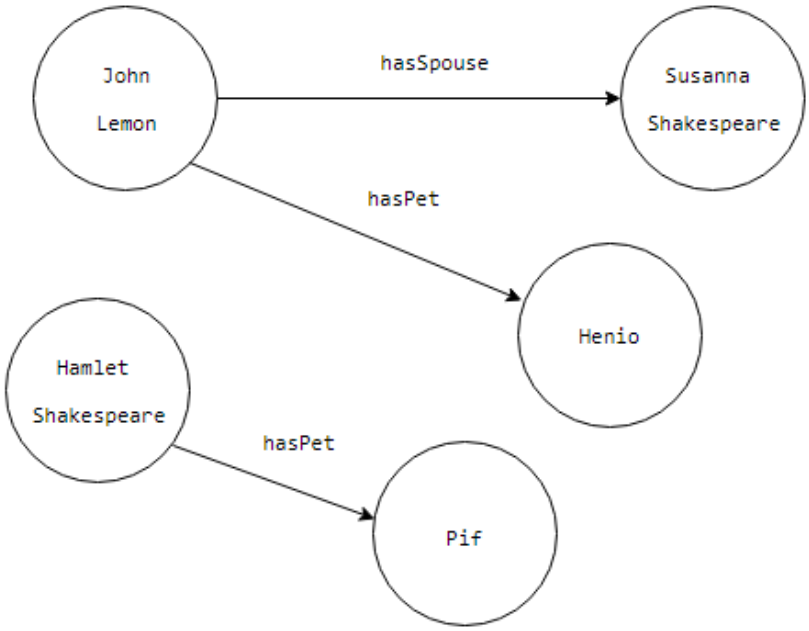
RDF Graph

Characteristic feature of RDF data model is a triple. It is a statement composed of three elements: subject, predicate and object. Subject and object are represented as nodes in graph and predicate is a relation between them (edge in graph). Picture 1.1 shows example of triple.



Picture 1.1. Example of triple.

Data is retrieved from RDF format using SPARQL. *SPARQL Protocol and RDF Query Language* is a RDF query language. Example of retrieving data is shown below.



Picture 1.2. RDF graph.

Query below returns couples of owner and pet where owner has spouse.

```
SELECT DISTINCT ?owner ?pet
WHERE {
      ?owner person:hasPet ?pet .
      ?owner person:hasSpouse ?spouse
}
```

Result:

owner	pet
John Lemon	Henio

SPARQL query can contains filters to return more detailed results.
Query below returns couples of male owner and male pet where owner has spouse and pet is male.

```
SELECT DISTINCT ?owner ?pet
WHERE {
      ?owner person:hasPet ?pet .
      ?owner person:hasSpouse ?spouse .
      ?pet being:gender ?gender .
FILTER((?gender='male') )
}
```

Ontology Schema

Ontology which was used in project contains two classes: Query and Condition. First of them is used to define queries and second allows user to specify filters.

Query

Query object contains following fields:

Field	Description
`name`	Datatype Property, which helps user to understand query function
`description`	Datatype Property, explains the purpose of query
`queryString`	Datatype Property, which contains text of query
`queryCondition`	Object Property, defines Condition objects used in query

```
query:Query_1
  rdf:type query:Query ;
  query:description "Information about animals with wings." ;
  query:name "Animals with wings" ;
  query:queryCondition query:hasWings ;
  query:queryString "SELECT *
                     WHERE { ?subject animal:hasWings
                     ?object . ?subsubject (rdfs:subClassOf)*
                     ?subject . ?instance a ?type .
                     FILTER(( ${hasWings}) && ( ?type = ?subsubject)) .}" ;
```

.

Condition

Condition object contains following Datatype Property fields:

Field	Description
defaultOperator	Default filter condition operator
defaultValue	Default filter condition value
conditionName	Helps user to understand the condition
conditionDesc	Explains the purpose of the condition
conditionType	Type of value in filter condition (xsd:boolean, xsd:string, etc.)
variable	Name of the variable used in filter condition

```
query:hasWings
  rdf:type query:Condition ;
  query:defaultOperator "=" ;
  query:defaultValue "true" ;
  query:conditionDesc "Animal has wings?" ;
  query:conditionName "Wings" ;
  query:conditionType "xsd:boolean" ;
  query:variable "object" ;
.
```

showQueriesTTL

List all available definitions of queries with their conditions.

Request

Method	URL
GET	api/showQueriesTTL/

Response

Status	Example Response

200	[
	  {
	  "name": "Query_3",
	  "description": "Persons in son-in-law relationship",
	  "label": "Son-in-law",
	  "uri": "http://topbraid.com/query#Query_3",
	  "query": "SELECT ?x ?spouse
	  WHERE { ?x person:hasDaughter ?daughter .
	  ?spouse (person:hasSpouse)+|^person:hasSpouse
	  ?daughter . ?spouse being:gender ?gender .
	  FILTER(?gender = 'male') . }",
	  "conditions": [
	  {
	  "name": "gender",
	  "description": "Gender of son-in-law",
	  "label": "Gender",
	  "uri": "http://tutorial.topbraid.com/query#gender",
	  "defaultOperator": "=",
	  "defaultValue": "male",
	  "conditionType": "xsd:string",
	  "variable": "gender"
	  }
	  ]
	  }
	]

500	{"error":"Something went wrong. Please try again."}

Parameter	Type	Description
name	string	Query / condition name in database
label	string	Query / condition name shown to user, easy for him to understand
description	string	Query / condition short description. Explains the purpose of query / condition to user
uri	string	Unique URI of Query / condition. Consists of database namespace and name of query / condition
query	string	Query formed in SPARQL language
conditions	list	List of definitions of query conditions
defaultOperator	string	Default query condition operator. Inserted into query every time user doesn't specify operator for this condition
defaultValue	string	Default query condition value. Inserted into query every time user doesn't specify value for this condition
conditionType	string	Query condition type (string / boolean / numeric)
variable	string	Variable name of query condition specified in SPARQL query

runQueryWithFilter

Run specific query with given operators and values of query conditions. Request body must have specified form. Response is a list of result records.

Request

Method	URL
POST	api/runQueryWithFilter/

Parameter	Example value

queryRequest	{
	  "id": 0,
	  "conditions": [
	  {
	  "operator": "=",
	  "uri": "http://topbraid.com/query#gender",
	  "value": "male",
	  "variable": "gender"
	  }
	  ]
	}

Parameter	Type	Description
id	string	Query internal API id
conditions	list	Query conditions list
operator	string	Query condition operator specified by user
uri	string	Unique URI of query condition. Consists of database namespace and name of query condition
value	string	Value of condition specified by user
variable	string	Variable name in condition specified in SPARQL query

Response

Status	Example Response

200	[
	    {
	        "result": [
	            {
	                "varName": "person",
	                "varValue": "William_Shakespeare"
	            },
	            {
	                "varName": "spouse",
	                "varValue": "John_Lemon"
	            }
	        ]
	    }
	]

500	{"error":"Something went wrong. Please try again."}

Parameter	Type	Description
result	list	List of query result variables and their values
varName	string	Name of query result variable. Result table can be built based on these as its the column name
varValue	string	Value of query result variable

runQuery

Run specific query with default operators and values in query conditions. Used to determine query form readable for user when listing queries. Request body must have specified form. Response is a list of result records.

Request

Method	URL
POST	api/runQuery/

Parameter	Example value

queryRequest	{
	  "id": 0
	}

Parameter	Type	Description
id	string	Query internal API ID

Response

Status	Example Response

200	[
	  {
	  "result": [
	  {
	  "varName": "person",
	  "varValue": "William_Shakespeare"
	  },
	  {
	  "varName": "spouse",
	  "varValue": "John_Lemon"
	  }
	  ]
	  }
	]

500	{"error":"Something went wrong. Please try again."}

Parameter	Type	Description
result	list	List of query result variables and their values
varName	string	Name of query result variable. Result table can be built based on these as its the column name
varValue	string	Value of query result variable

Services

TTLReaderService

Reads TTL file. Extracts model, queries and prefixes.

Query information is extracted from TTL file with following queries.
Query for listing all available queries:

```
SELECT DISTINCT ?uri ?name ?description ?query
WHERE {
    ?uri query:queryString ?query .
    ?uri query:name ?name .
    ?uri query:description ?description .
}
```

Query for listing all conditions of specified query:

```
SELECT DISTINCT *
WHERE {
       ?uri query:queryCondition ?condUri .
       ?condUri query:conditionName ?name .
       OPTIONAL { ?condUri query:defaultValue ?default } .
       ?condUri query:conditionDesc ?description .
       ?condUri query:variable ?var .
       ?condUri query:conditionType ?type .
       ?condUri query:defaultOperator ?operator .
FILTER (
       ?uri = <QUERY_URI>)
}
```

Prefixes are extracted from TTL file and used by QueryBuilderService, every time a query is composed.

QueryBuilderService

Handles queries.

The SPARQL query is composed based on request given by user. Conditions are built based on given values and operators and then inserted into SPARQL query. If user didn’t provide value or operator, the default ones are used.

Sends composed query to QueryExecutorService and handles the result.

QueryExecutorService

Runs parameterized query on “database”. In current version the TTL file serves as a database. Queries are executed on model extracted from the TTL file.

Package org.apache.jena.query contains ARQ - a query engine for Jena, implementing SPARQL.

Glossary

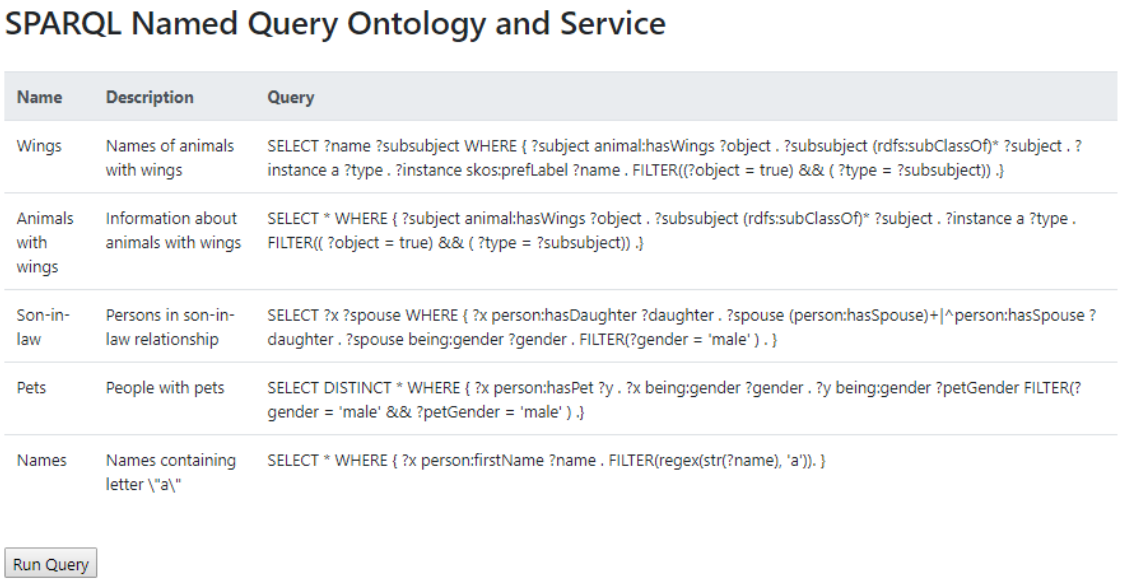
Conventions

- **Status** - HTTP status code of response.
- **TTL** - Terse RDF Triple Language.
- All the possible responses are listed under ‘Responses’ for each method. Only one of them is issued per request server.
- All response are in JSON format.
- All request bodies are in JSON format.
- All request conditions are mandatory unless explicitly marked as `[optional]`

Users guide

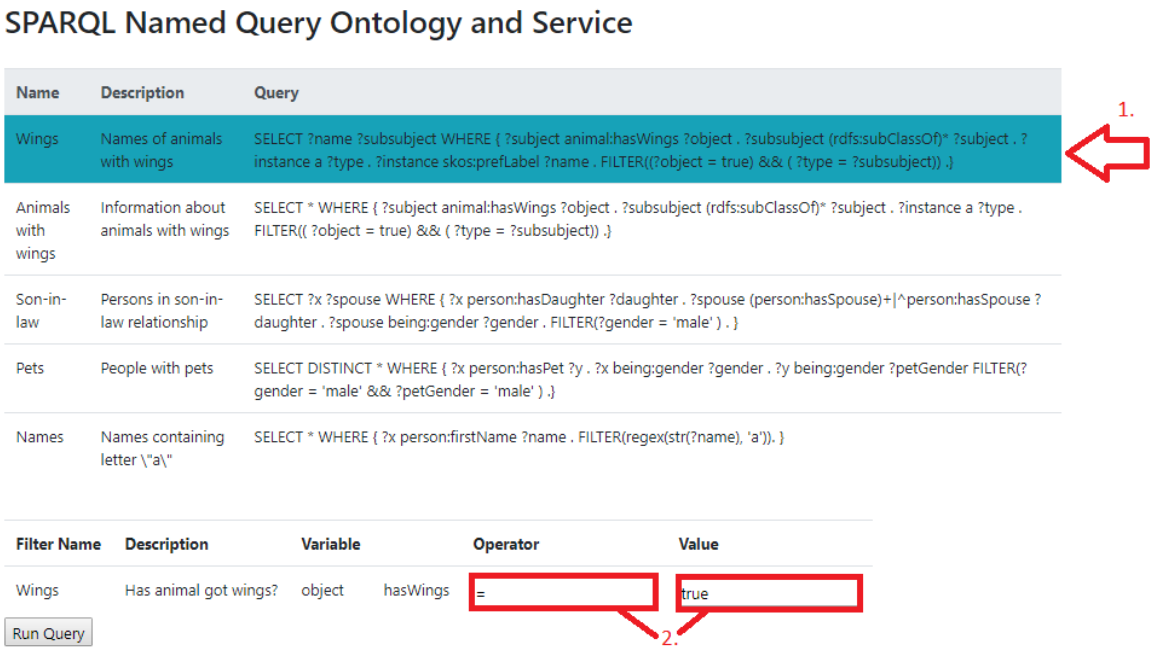
This section aims to present how to use an application, including setting filters, executing queries and exploring results.

Home page of application contains list of available queries. Each query has name and description. These attributes can help user to find specific query.



Picture 2.1. Home page of application.

After choosing one query from list, system will show filters connected to this query. User can change operator and value of filter.



Picture 2.2. Selecting query.

To run query user need to choose *Run Query* button. System will show query result below.

Filter Name	Description	Variable		Operator	Value
Wings	Has animal got wings?	object	hasWings	=	true

Run Query

Name	Value
subsubject	Peacock
name	Edward

Name	Value
subsubject	Peacock
name	James

Name	Value
subsubject	Ara
name	Jimmy

Name	Value
subsubject	Canary
name	Buddy

Result

Picture 2.3. Query result.