RDF/JS: Data model specification

RDF/JS: Data model specification

Draft Community Group Report 05 March 2019

Latest editor's draft:
https://github.com/rdfjs/data-model-spec/

Bug tracker:
File a bug (open bugs)

Editors:
Thomas Bergwinkl (Zazuko)
Ruben Verborgh (Ghent University – imec)

Authors:
Thomas Bergwinkl (Zazuko)
Michael Luggen (Bern University of Applied Sciences)
elf Pavlik
Blake Regalia (STKO Lab @ UCSB)
Piero Savastano (Freelance Data Scientist)
Ruben Verborgh (Ghent University – imec)

Version control:
Github Repository

Abstract

Status of This Document

This specification was published by the RDF JavaScript Libraries Community

This document provides a specification of a low level interface definition representing RDF data independent of a serialized format in a JavaScript environment. The task force which defines this interface was formed by RDF JavaScript library developers with the wish to make existing and future libraries interoperable. This definition strives to provide the minimal necessary interface to enable interoperability of libraries such as serializers, parsers and higher level accessors and manipulators.

If you wish to make comments regarding this document, please send them to public-rdfjs@w3.org (subscribe, archives).

Table of Contents

§ 1. Design elements and principles

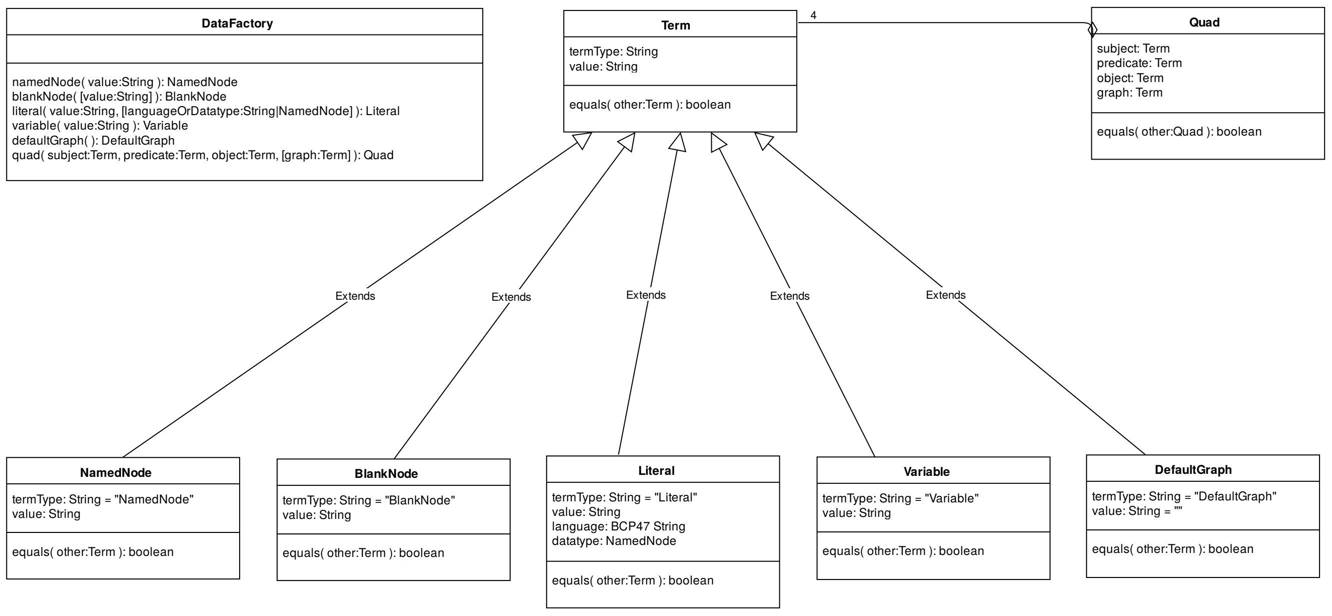
- We define data interfaces to represent **quads**, **named nodes**, **blank nodes**,

literals and **variables**.

- Instances of the interfaces created with different libraries should be interoperable.
- Interfaces do *not* specify how instances are stored in memory.
- Interfaces mandate specific pre-defined methods such as `.equals()`.
- Factory functions (e.g., `quad()`) or methods (e.g., `store.createQuad()`) create instances.
- Interfaces may have additional implementation specific properties.
- We don't define any validation of given values (e.g. IRI, URI, CURIE). Implementations that apply validation should make this fact clear in their documentation.

A list of these properties maintained on the RDFJS Representation Task Force wiki .

§ 2. Data interfaces


DataFactory
namedNode(value:String): NamedNode
blankNode([value:String]): BlankNode
literal(value:String, [languageOrDatatype:String|NamedNode]): Literal
variable(value:String): Variable
defaultGraph( ): DefaultGraph
quad(subject:Term, predicate:Term, object:Term, [graph:Term]): Quad
Term
termType: String
value: String
equals(other:Term): boolean
4
Quad
subject: Term
predicate: Term
object: Term
graph: Term
equals(other:Quad): boolean
Extends
NamedNode
termType: String = "NamedNode"
value: String
equals(other:Term): boolean
BlankNode
termType: String = "BlankNode"
value: String
equals(other:Term): boolean
Literal
termType: String = "Literal"
value: String
language: BCP47 String
datatype: NamedNode
equals(other:Term): boolean
Variable
termType: String = "Variable"
value: String
equals(other:Term): boolean
DefaultGraph
termType: String = "DefaultGraph"
value: String = ""
equals(other:Term): boolean

§ 2.1 *Term* interface

WebIDL

```
interface Term {
  attribute string termType;
  attribute string value;
  boolean equals(optional Term? other);
};
```

Term is an abstract interface.

termType contains a value that identifies the concrete interface of the term, since

Term itself is not directly instantiated. Possible values include `"NamedNode"`, `"BlankNode"`, `"Literal"`, `"Variable"` and `"DefaultGraph"`.

`value` is refined by each interface which extends Term.

`equals()` returns `true` when called with parameter `other` on an object `term` if all of the conditions below hold:

- `other` is *neither* `null` nor `undefined`;
- `term.termType` is the same string as `other.termType`;
- `other` follows the additional constraints of the specific `Term` interface implemented by `term` (e.g., `NamedNode`, `Literal`, …);

otherwise, it returns `false`.

§ 2.2 *NamedNode* interface

WebIDL

```
interface NamedNode : Term {
  attribute string termType;
  attribute string value;
  boolean equals(optional Term? other);
};
```

`termType` contains the constant `"NamedNode"`.

`value` the IRI of the named node (example: `"http://example.org/resource"`).

`equals()` returns `true` if all general `Term.equals` conditions hold and `term.value` is the same string as `other.value`; otherwise, it returns `false`.

§ 2.3 *BlankNode* interface

WebIDL

```
interface BlankNode : Term {
  attribute string termType;
  attribute string value;
  boolean equals(optional Term? other);
};
```

`termType` contains the constant `"BlankNode"`.

`value` blank node name as a string, without any serialization specific prefixes, e.g. when parsing, if the data was sourced from Turtle, remove `"_:"`, if it was sourced from RDF/XML, do not change the blank node name (example: `"blank3"`)

`equals()` returns `true` if all general `Term.equals` conditions hold and `term.value` is the same string as `other.value`; otherwise, it returns `false`.

§ 2.4 *`Literal`* interface

WebIDL

```
interface Literal : Term {
  attribute string termType;
  attribute string value;
  attribute string language;
  attribute NamedNode datatype;
  boolean equals(optional Term? other);
};
```

`termType` contains the constant `"Literal"`.

`value` the text value, unescaped, without language or type (example: `"Brad Pitt"`)

`language` the language as lowercase BCP-47 [BCP47] string (examples: `"en"`, `"en-gb"`) or an empty string if the literal has no language.

`datatype` a `NamedNode` whose IRI represents the datatype of the literal.

If the literal has a language, its datatype has the IRI `"http://www.w3.org/1999/02`

`/22-rdf-syntax-ns#langString"`. Otherwise, if no datatype is explicitly specified, the datatype has the IRI `"http://www.w3.org/2001/XMLSchema#string"`.

`equals()` returns `true` if all general `Term.equals` conditions hold, `term.value` is the same string as `other.value`, `term.language` is the same string as `other.language`, and `term.datatype.equals(other.datatype)` evaluates to `true`; otherwise, it returns `false`.

§ 2.5 *`Variable`* interface

WebIDL

```
interface Variable : Term {
  attribute string termType;
  attribute string value;
  boolean equals(optional Term? other);
};
```

`termType` contains the constant `"Variable"`.

`value` the name of the variable without leading `"?"` (example: `"a"`).

`equals()` returns `true` if all general `Term.equals` conditions hold and `term.value` is the same string as `other.value`; otherwise, it returns `false`.

§ 2.6 *`DefaultGraph`* interface

WebIDL

```
interface DefaultGraph : Term {
  attribute string termType;
  attribute string value;
  boolean equals(optional Term? other);
};
```

An instance of `DefaultGraph` represents the default graph. It's only allowed to assign a `DefaultGraph` to the `graph` property of a `Quad`.

`termType` contains the constant `"DefaultGraph"`.

`value` contains an empty string as constant value.

`equals()` returns `true` if all general `Term.equals` conditions hold; otherwise, it returns `false`.

§ 2.7 ***Quad*** interface

WebIDL

```
interface Quad {
  attribute Term subject;
  attribute Term predicate;
  attribute Term object;
  attribute Term graph;
  boolean equals(optional Quad? other);
};
```

`subject` the subject, which is a `NamedNode`, `BlankNode` or `Variable`.

`predicate` the predicate, which is a `NamedNode` or `Variable`.

`object` the object, which is a `NamedNode`, `Literal`, `BlankNode` or `Variable`.

`graph` the named graph, which is a `DefaultGraph`, `NamedNode`, `BlankNode` or `Variable`.

NOTE

`Triple` *MUST* be represented as `Quad` with `graph` set to a `DefaultGraph`

`equals()` returns `true` when called with parameter `other` on an object `quad` if all of the conditions below hold:

- `other` is *neither* `null` nor `undefined`;
- `quad.subject.equals(other.subject)` evaluates to `true`;
- `quad.predicate.equals(other.predicate)` evaluates to `true`;
- `quad.object.equals(other.object)` evaluates to a `true`;
- `quad.graph.equals(other.graph)` evaluates to a `true`;

otherwise, it returns `false`.

§ 2.8 ***DataFactory*** interface

WebIDL

```
interface DataFactory {
  NamedNode namedNode(string value);
  BlankNode blankNode(optional string value);
  Literal literal(string value, optional (string or NamedNode)
languageOrDatatype);
  Variable variable(string value);
  DefaultGraph defaultGraph();
  Quad quad(Term subject, Term predicate, Term object, optional Term?
graph);
};
```

For default values of the instance properties and valid values requirements, see the individual interface definitions.

`namedNode()` returns a new instance of `NamedNode`.

`blankNode()` returns a new instance of `BlankNode`. If the value parameter is undefined a new identifier for the blank node is generated for each call.

`literal()` returns a new instance of `Literal`. If `languageOrDatatype` is a `NamedNode`, then it is used for the value of `datatype`. Otherwise `languageOrDatatype` is used for the value of `language`.

`variable()` returns a new instance of `Variable`. This method is optional.

`defaultGraph()` returns an instance of `DefaultGraph`.

`quad()`returns a new instance of `Quad`. If `graph` is `undefined` or `null` it *MUST* set `graph` to a `DefaultGraph`.

§ A. References

§ A.1 Normative references

[BCP47]

Tags for Identifying Languages. A. Phillips; M. Davis. IETF. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

§ A.2 Informative references

[WEBIDL]

Web IDL. Boris Zbarsky. W3C. 15 December 2016. W3C Editor's Draft. URL: https://heycam.github.io/webidl/

↑