



OSLC Resource Shape 3.0

OASIS Working Draft

20 January 2016

Specification URIs

This version:

Latest versions:

none

Latest editor's draft:

<https://tools.oasis-open.org/version-control/svn/oslc-core/trunk/specs/resource-shape.html>

Technical Committee:

[OASIS OSLC Lifecycle Integration Core \(OSLC Core\) TC](#)

Chairs:

[Jim Amsden](#)

[Martin Sarabura](#)

Editors:

Jim Amsden, [IBM](#), jamsden@us.ibm.com

Arthur Ryman

Namespaces:

<http://open-services.net/ns/core#>

Abstract:

This document defines a high-level RDF vocabulary for specifying the *shape* of RDF resources. The shape of an RDF resource is a description of the set of triples it is expected to contain and the integrity constraints those triples are required to satisfy. Applications of shapes include validating RDF data, documenting RDF APIs, and providing metadata to tools, such as form and query builders, that handle RDF data.

Status:

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current OASIS publications can be found in the [OASIS standards page](#) at <https://www.oasis-open.org/standards>.

Stable Working Draft: received and incorporated TC feedback. (Almost) Ready for Public Review.

This document was published by the [OASIS OSLC Lifecycle Integration Core \(OSLC Core\) TC](#) as a Working Draft. If you wish to make comments regarding this document, please send them to oslc-core-comments@lists.oasis-open.org ([subscribe](#), [archives](#)). All comments are welcome.

Notices

Copyright © OASIS Open © 2016. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

Table of Contents

- 1. [Introduction](#)
- 2. [Terminology](#)
- 3. [Conformance](#)
 - 3.1 [Conventions Used in This Document](#)
- 4. [Motivation](#)
 - 4.1 [Describing RDF APIs](#)
 - 4.2 [Describing RDF Datasets](#)
 - 4.3 [Describing Indexable Data](#)
- 5. [Requirements](#)
- 6. [Shape Resources](#)
 - 6.1 [Overview](#)
 - 6.2 [Associating and Applying Shapes](#)
 - 6.3 [A Running Example of Shape Resources](#)
- 7. [Vocabulary](#)
 - 7.1 [Vocabulary Details](#)
- 8. [Constraints](#)
 - 8.1 [ResourceShape Constraints](#)

- 8.2 [Property Constraints](#)
 - 8.3 [AllowedValues Constraints](#)
 - 9. [Recommended Extensions](#)
 - 9.1 [Inverse Properties](#)
 - 9.2 [Disconnected Graphs](#)
 - 9.3 [Datatype Facets](#)
 - 9.4 [Extending Shapes](#)
 - 9.5 [SPARQL Semantics](#)
 - A. [Acknowledgements](#)
-

1. Introduction

This section is non-normative.

Linked Data has proved to be an effective way to look up and navigate information at web scale. It is also emerging as a compelling architectural basis for web applications [[RW Linked Data](#)]. The principles for designing Linked Data web applications are standardized by the W3C Linked Data Platform Specification 1.0 [[LDP](#)].

RDF is at the core of Linked Data. The design principles for Linked Data [[Linked Data Design Issues](#)] include the following rule:

When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)

This emergence of RDF as an important application development technology for Linked Data and other domains has brought to light a serious gap in the associated technology stack, namely the absence of a way to describe the integrity constraints imposed by an application on the RDF documents it processes. Here the term “integrity constraint” denotes any characteristic of data required by or enforced by an application. Common examples of integrity constraints include the mandatory presence of properties, the cardinality of relations, and restrictions on the allowed values of properties.

Well-established programming technologies such as relational databases, object-oriented programming languages, and XML utilize [closed-world assumptions](#) to provide schemas or type definitions that encode integrity constraints. Applications utilizing RDF and [open-world assumptions](#) may need a similar mechanism to close the world down, and frequently start to use RDFS or OWL for this purpose. However, this is a misapplication of those technologies.

It is well-known that the semantics of RDFS and OWL are defined in terms of inference rules, not constraints. Reasoners will infer new triples in an attempt to satisfy RDFS or OWL ontologies. Sometimes this is exactly what a programmer wants, but if errors are present in the data, then the inferred triples will be nonsense and, worse, they may mask the presence of errors.

This document describes the Resource Shape specification, a high-level RDF vocabulary for describing the *shape* of RDF resources. Here the term “RDF resource” is used to denote an LDPR resource that has an RDF document among its set of available representations. The term “shape” is used because it is often useful to visualize an RDF document as a topological object, namely as a graph consisting of nodes interconnected by arcs. Throughout this document the terms “RDF resource”, “RDF document”, and “RDF graph” are used more or less interchangeably albeit somewhat imprecisely.

The shape of an RDF graph includes both a description of its expected contents (properties, types) within some operational context (e.g. GET, POST) and the integrity constraints that must be satisfied by the contents in that context.

Resource shapes specify a standard way of describing resources and constraints on resources that may be used in defining among other things, OSLC domain resources. There are other means that OSLC domains may use in addition to or instead of resource shapes such as W3C [[SHACL](#)] or [[JSONSchema](#)]. OSLC servers should describe their resources using resource shapes if they wish to integrate with [[OSLCCore2](#)] clients or servers that are expecting resource shapes. OSLC Domain specifications are expected to specify how the constraints on their domain elements are expressed and supported by servers.

This specification begins with a brief discussion of the use cases and requirements that the Resource Shape specification addresses. It then describes all aspects of the current specification in detail. Finally, it concludes with some recommendations for extensions based on implementation experience.

2. Terminology

Terminology is based on OSLC Core Overview [[OSLCCore3](#)], W3C Linked Data Platform [[LDP](#)], W3C's Architecture of the World Wide Web [[WEBARCH](#)], Hyper-text Transfer Protocol [[HTTP11](#)].

This specification uses the following terminology:

applicability

Shapes associated with a resource apply to that resource if the shape's `oslc:describes` property matches the type of the associated resource, or the shape has no `oslc:describes` property in which case it applies to all resources associated with that

shape. For all shapes that "apply to" or describes an associated resource, that resource should satisfy the constraints defined by those shapes. See [Associating and Applying Shapes](#) for the definition of how shapes are associated with resources, what associated shapes apply to a resource, and what it means if multiple shapes apply to a single resource.

association

This specification defines three contexts in which a shape may become associated with a described resource. The described resource itself may link to a shape using the property [oslc:instanceShape](#), the described resource may be the entity request to or response from a REST service whose service description links to a shape using the property [oslc:resourceShape](#), or the the resource may be the object value of a Property whose [oslc:valueShape](#) links to the constraining shape. More than one shape may become associated with a described resource in a given context. The shapes associated with a resource are the set of shapes to be checked for applicability. Not all the associated shapes are necessarily applicable. Refer to the definition of applicability above.

datatype property

A datatype property is a defined property that takes literal values.

defined property

A defined property is a property, such as `dcterms:description` or `oslc_cm:status`, of a described resource that is defined by some shape applicable to the described resource in a given context. A shape is linked to its defined properties using the property [oslc:property](#). A described resource type might have a given defined property in some contexts but not in others. For example, the body of a POST request might have fewer defined properties than the created resource.

described resource

A described resource is the resource whose RDF graph is being **described by a shape**.

document

A representation of a resource. For example, an RDF document may represent a resource.

graph

An RDF document may be regarded as a directed, labeled multi-graph, or, simply, a graph, in which each triple of the document corresponds to an arc of the graph. The source of the arc is the subject its triple. The target of the arc is the object of its triple. The label of the arc is the predicate of its triple.

object property

An object property is a defined property that links to a resource, referred to as the object resource.

object resource

The object resource is the resource that is linked to by an object property.

shape

A resource of type [oslc:ResourceShape](#) that describes the contents of and constraints on some set of described resources.

3. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this specification are to be interpreted as described in [RFC2119].

3.1 Conventions Used in This Document

Sample resource representations are provided in `text/turtle` format [[TURTLE](#)].

The following common URI prefixes are used throughout this specification:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ex: <http://example.org/>.
@prefix ext: <http://example.org/extension#>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix oslc: <http://open-services.net/ns/core#>.
@prefix oslc_cm: <http://open-services.net/ns/cm#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

Fig. 1 Commonly used URI prefixes.

4. Motivation

This section is non-normative.

This section briefly describes the main use cases that motivate an RDF graph shape language. For a more complete discussion of this topic, refer to the W3C RDF Validation Workshop [\[RDF-VAL\]](#).

4.1 Describing RDF APIs

This section is non-normative.

Any application that provides programmatic services should also provide application programming interface (API) documentation to programmers so that they can consume those services. Applications that process RDF, including Linked Data web applications, are no different than other applications in this respect.

Although API documentation is normally directed towards human programmers, there are also important use cases where other programs need to understand the API. For example, consider a generic form-building tool that can generate a user interface for creating or updating resources. Such a tool needs to understand the expected contents of the resource so it can generate a user interface. It also needs to understand the applicable integrity constraints so it can validate user input before sending it to the service provider. Furthermore, the service provide may apply different constraints for creation versus update operations.

4.2 Describing RDF Datasets

This section is non-normative.

Users must understand the contents of an RDF dataset in order to write SPARQL queries. Understanding the integrity constraints can help users write better SPARQL queries. For example, if a property is known to always be present, then there is no need to wrap it in an **OPTIONAL** clause.

Query building tools can take advantage of shape information. For example, consider a query builder that allows the user to define a query that filters results by selecting allowed values from a list. In principle, the query builder could dynamically query the dataset to determine the list of values present. However, such a query could be slow if the dataset was large. In addition, only those values present at the time of the query would appear in the list. In contrast, if the query builder had access to shape information, then it could avoid the potentially expensive query and present the user with the complete list of allowed values, whether or not they were actually present in the dataset at that time.

4.3 Describing Indexable Data

This section is non-normative.

Resource indexers may be looking for certain types of structured data. For example, a web crawler might be indexing product descriptions and pricing for a marketplace application. The web crawler could provide a shape to describe the data it is looking for. Web application developers would then be able to provide that information in the web pages of their commerce site and thereby have their sites included in the index.

5. Requirements

This section is non-normative.

This section describes some high-level requirements for an RDF resource shape language. For simplicity, members of a shape language are referred to as shape resources. It is to be understood that the validity and meaning of a shape resource depends on any other shape resources it is linked to.

1. A shape language must be able to express the most commonly occurring aspects of RDF resources in high-level terms, using familiar concepts. Interested parties must be able to both understand existing shapes and author new ones.
2. A shape resource must be processable using readily available RDF technologies. These include RDF parsers and SPARQL processors.
3. The constraints defined by a shape resource must be verifiable using commonly available technologies, with acceptable performance. The performance criteria depend on the use case. For example, a tool checking user input must respond quickly. A tool validating an entire dataset may run as a batch job, but should complete within a reasonable amount of time (e.g. hours, not years).
4. A shape language must be extensible so that application-specific aspects of resources can be expressed. A trade-off of expressiveness at the expense of ease-of-understanding is acceptable.
5. It must be possible to apply different shapes to the same resources in different contexts in order to use the resources for different purposes.

6. Shape Resources

This section describes the Resource Shape specification (aka the *Shapes* specification) which is part of [\[OSLCCore3\]](#). The Shapes

specification defines:

- the contents and meaning of shape resources, and
- how to associate shape resources with resources and services

6.1 Overview

A resource shape is a resource that describes the contents of, and constraints on, the RDF representation of other resources. A shape resource itself has an RDF representation which uses the terms defined by the `oslc:` vocabulary. The term “shape resource” or simply “shape” is sometimes used as shorthand for the more verbose phrase “the RDF representation of a shape resource” where this can lead to no confusion. The following sections describe all of the RDF vocabulary terms used in shape resources.

The Shapes specification is based on a simple conceptual model of resources that works well in practice, but is somewhat biased towards the view that the RDF representation of a resource looks like a set of property-value pairs on that resource. The Shapes specification works well when the resource being described appears as a subject node in its RDF graph and all other nodes are connected to the resource node by a path consisting of one or more properties. Each property-value pair is represented by a triple in which the subject is the resource, the predicate is the property, and the object is the value. The value may be either a literal or a resource. When the object is a resource, that resource may itself be described by another shape. Thus the Shapes specification is powerful enough to describe complex graphs. Although the Shapes specification works well in practice, it cannot describe arbitrary RDF graphs. This limitation is discussed below in [Recommended Extensions](#).

The following diagram summarizes the main concepts and relations used in this specification:

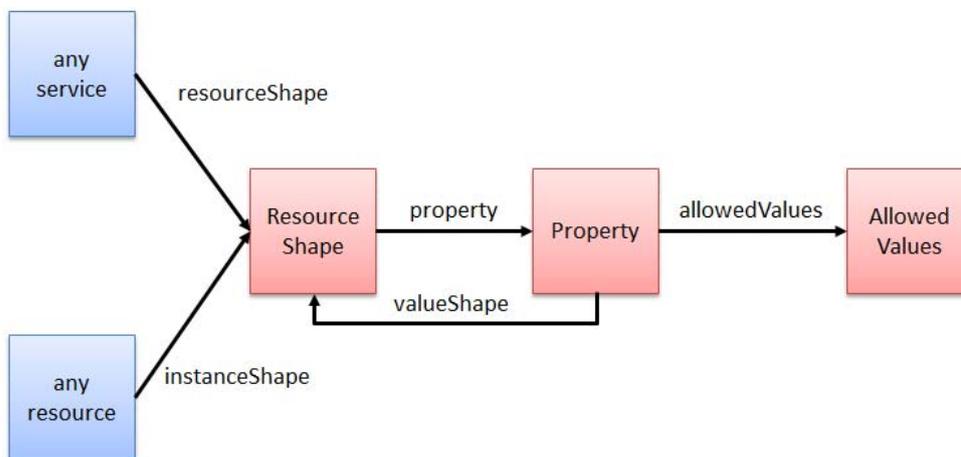


Fig. 2 Diagram of main concepts and relations.

In this diagram the boxes represent resource types and the arrows represent the relations between them that are defined by the Shapes specification. The two boxes on the left represent external types of resources that use shapes. The other three boxes represent the resource types that are defined by the Shapes specification.

The box labeled “Resource Shape” represents a shape. A shape is a resource of `rdf:type` [oslc:ResourceShape](#). A shape describes a set of resources. A shape is basically a set of defined properties that any resource of that shape is expected to contain. A shape contains one or more defined properties.

The box labeled “Property” represents a defined property. A defined property is resource of `rdf:type` [oslc:Property](#). The arrow labeled “property” represents the aggregation relation between a shape and its defined properties. The predicate of this relation is [oslc:property](#).

Each [oslc:Property](#) resource has a set of properties that describe the defined property and the constraints on its use within any resource that the given shape applies to. These include a description of the values of the defined property and its occurrence within the resource. The value of a defined property may be a literal, a resource, or either. If the value of a defined property is a resource, then defined property may refer to another [oslc:ResourceShape](#) resource that describes the value resource. This relation is depicted by the arrow labeled “valueShape”. The predicate of this relation is [oslc:valueShape](#).

The value of a defined property may be constrained to take one of an allowed set of values. In some cases, the allowed set of values may be large and be used in many shapes. In this case it is useful to put the allowed values in a separate resource so they can be easily reused. The box labeled “Allowed Values” represents a resource of `rdf:type` [oslc:AllowedValues](#). The arrow labeled “allowedValues” represents the relation between a defined property and its set of allowed values. The predicate of this relation is [oslc:allowedValues](#).

A [\[REST\]](#) service may describe aspects of its interface contract using shapes. For example, a REST service may provide a URI where new resources can be created via HTTP POST. This service could describe the expected contents of POST request bodies using shapes. Similarly, a REST service may provide a URI that represents a container of resources and could describe those resources using shapes. The box labeled “any service” represents any REST service description. The arrow labeled “resourceShape” represents the predicate [oslc:resourceShape](#) which is a property of the service description resource.

Similarly, any resource can describe its own contents by linking to a shape resource. The box labeled “any resource” represents any resource. The arrow labeled “instanceShape” represents the predicate [oslc:instanceShape](#) which is a property of the resource.

6.2 Associating and Applying Shapes

In general, the relation between shapes and resources is many-to-many. Given a resource R there **MAY** be zero or more shapes S associated with it. This specification defines three ways to associate shapes with a resource, namely using [oslc:instanceShape](#), [oslc:resourceShape](#), and [oslc:valueShape](#). Other specifications **MAY** define additional mechanisms.

1. Resource [oslc:instanceShape](#) ResourceShape - directly associates a constraining shape with a resource.
2. Service [oslc:resourceShape](#) ResourceShape - associates a constraining shape with the entity request or response resource of a service (e.g., a creation or query service).
3. Property [oslc:valueShape](#) ResourceShape - associates a constraining shape with the resource that is the object value of a property of a resource.

Not all shapes associated with a resource are necessarily applicable to it. Let S be associated with R. S is said to *apply* to R in the following two cases:

Case 1: Generic Shape

S applies to R when S has no [oslc:describes](#) properties, in which case it is a generic shape and applies to any associated resource.

Case 2: Typed Shape

S applies to R when S is linked via [oslc:describes](#) to a rdf:type T and R has rdf:type T.

If no shapes are associated with a resource then there are no implied constraints on that resource.

If one or more shapes are associated with a resource then at least one of those **SHOULD** be applicable to that resource. If no associated shape applies to a resource then this **SHOULD** normally be interpreted as an error condition.

If exactly one shape applies to a resource then that resource **SHOULD** satisfy all the constraints defined by that shape.

If more than one shape applies to a resource then that resource **SHOULD** satisfy all the constraints defined by all the shapes (AND semantics). However, the specification for a service description **MAY** define alternate semantics. For example, a service **MAY** require that the resource satisfy the constraints defined by at least one of the shapes (OR semantics).

6.3 A Running Example of Shape Resources

This section presents a simple running example to illustrate the Shapes specification. For more examples, refer to [\[Linked Data Interfaces\]](#), [\[Shapes-LDOW2013\]](#), and [\[Shapes-RDF-VAL\]](#).

Consider a simple bug tracking service in which each bug has just two properties: a summary and status. The summary is required, but the status is optional. The summary is given by the property [dcterms:title](#), and the status by [oslc_cm:status](#). The [oslc_cm:status](#) is constrained to take one of the values “Submitted”, “InProgress”, or “Done”. The RDF type of a bug is [oslc_cm:ChangeRequest](#).

6.3.1 Example of a Valid Bug

This section is non-normative.

The following listing shows the RDF representation of the bug [http://example.com/bugs/1](#) which satisfies the constraints defined by the bug tracking service:

EXAMPLE 1

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_cm: <http://open-services.net/ns/cm#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://example.com/bugs/1> a oslc_cm:ChangeRequest ;
  dcterms:title "Null pointer exception in web ui"^^rdf:XMLLiteral ;
  oslc_cm:status "Submitted" ;
  oslc:instanceShape <http://example.com/shape/oslc-change-request> .
```

6.3.2 Example of an Invalid Bug

This section is non-normative.

The following listing shows the RDF representation of the bug [http://example.com/bugs/2](#) which violates the constraints since its

oslc_cm:status property has two values:

EXAMPLE 2

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_cm: <http://open-services.net/ns/cm#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://example.com/bugs/2> a oslc_cm:ChangeRequest ;
  dcterms:title "Wrong arguments"^^rdf:XMLLiteral ;
  oslc_cm:status "Submitted", "InProgress" ;
  oslc:instanceShape <http://example.com/shape/oslc-change-request> .
```

6.3.3 Example of a Shape for Bugs

This section is non-normative.

We can represent the constraints defined by the bug tracking service using the shape resource <http://example.com/shape/oslc-change-request> which has the following RDF representation:

EXAMPLE 3

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix oslc_cm: <http://open-services.net/ns/cm#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@base <http://example.com/shape/> .

<oslc-change-request> a oslc:ResourceShape ;
  dcterms:title "Creation shape of OSLC Change Request"^^rdf:XMLLiteral ;
  oslc:describes oslc_cm:ChangeRequest ;
  oslc:property
    <oslc-change-request#dcterms-title> ,
    <oslc-change-request#oslc_cm-status> .

<oslc-change-request#dcterms-title> a oslc:Property ;
  oslc:propertyDefinition dcterms:title ;
  oslc:name "title" ;
  oslc:occurs oslc:Exactly-one .

<oslc-change-request#oslc_cm-status> a oslc:Property ;
  oslc:propertyDefinition oslc_cm:status ;
  oslc:name "status" ;
  oslc:occurs oslc:Zero-or-one ;
  oslc:allowedValues <status-allowed-values> .
```

The RDF representation contains one [oslc:ResourceShape](#) resource and two [oslc:Property](#) resources. The [oslc:ResourceShape](#) resource contains a short description of the shape using [dcterms:title](#), gives the RDF type of the resource being described using [oslc:describes](#), and lists the two properties contained in the resource being described using [oslc:property](#).

The contained properties, [dcterms:title](#) and [oslc_cm:status](#) are described in [oslc:Property](#) resources. Each [oslc:Property](#) resource specifies the URI of the RDF property it is defining using [oslc:propertyDefinition](#), and the occurrence of that property using [oslc:occurs](#). The occurrence of each property is specified using the terms [oslc:Exactly-one](#) and [oslc:Zero-or-one](#) which indicate that the properties are both single-valued. [dcterms:title](#) is required and [oslc_cm:status](#) is optional.

In this example, only the occurrence and allowed values constraints are used. The Shapes specification provides for properties to be constrained by value type, range, and several other aspects.

6.3.4 Example of Allowed Values for Status

This section is non-normative.

The definition of [oslc_cm:status](#) refers to <http://example.com/shape/status-allowed-values> using [oslc:allowedValues](#). That resource has the [rdf:type](#) [oslc:AllowedValues](#). It specifies the allowed values of the [oslc_cm:status](#) property. It has the following RDF representation:

EXAMPLE 4

```
@prefix oslc: <http://open-services.net/ns/core#> .

<http://example.com/shape/status-allowed-values> a oslc:AllowedValues ;
  oslc:allowedValue "Done" , "InProgress" , "Submitted" .
```

7. Vocabulary

The Shapes vocabulary consists of the following terms [defined by the oslc: vocabulary](#):

In addition, shape resources use the following terms from Dublin Core [[Dublin Core](#)]: [dcterms:description](#) and [dcterms:title](#).

7.1 Vocabulary Details

The namespace URI for this vocabulary is: <http://open-services.net/ns/core#>

All vocabulary URIs defined in the OSLC Core namespace.

7.1.1 RDFS Classes in this namespace

[AllowedValues](#), [Property](#), [ResourceShape](#),

7.1.2 AllowedValues

<http://open-services.net/ns/core#AllowedValues> *AllowedValues* is an RDFS class.

Allowed values for one property.

7.1.3 Property

<http://open-services.net/ns/core#Property> *Property* is an RDFS class.

A Property resource describes one allowed or required property of a resource.

7.1.4 ResourceShape

<http://open-services.net/ns/core#ResourceShape> *ResourceShape* is an RDFS class.

The Resource Shape used for creation, query and modify. Formally, a shape S applies to a resource R if there is a triple R *rdf:type* T and there is a triple S *oslc:describes* T, or if there is a triple R *oslc:instanceShape* S.

8. Constraints

Property tables are used below to describe the resources defined by the Shapes specification, namely [oslc:ResourceShape](#), [oslc:Property](#), and [oslc:AllowedValues](#). A property table is a tabular depiction of a subset of the information that can be specified using shapes. Each table describes one type of resource. Each row describes one property of the resource. Each column describes some aspect of the properties. The columns have the following meanings:

Prefixed Name

The compact IRI [[IRI](#)] of the defined property being described in the current row of the table. This corresponds to the [oslc:propertyDefinition](#) property.

Occurs

The number of times the defined property may occur. This corresponds to the [oslc:occurs](#) property. This property may have one of the values [oslc:Exactly-one](#), [oslc:One-or-many](#), [oslc:Zero-or-many](#), and [oslc:Zero-or-one](#).

Read-Only

true if the property is read-only. If omitted, or set to false, then the property is writable. Providers **SHOULD** declare a property read-only when changes to the value of that property will not be accepted after the resource has been created, e.g. on PUT/PATCH requests. Consumers should note that the converse does not apply: Providers **MAY** reject a change to the value of a writable property.

Value-Type

The type of the values of the defined property. This corresponds to the [oslc:valueType](#) property. A defined property that takes literal values is called a *datatype property*. A defined property that links to another resource is called an *object property*. The resource linked to is called the *object resource*. For datatype properties, this property gives the datatype URI of the literal values. For object properties, this property specifies whether the object resource has a URI (Resource), is a blank node (LocalResource), or is any resource (AnyResource). These values correspond to [oslc:Resource](#), [oslc:LocalResource](#), and [oslc:AnyResource](#).

Representation

For object properties, the location of the representation of the object resource. This corresponds to the [oslc:representation](#) property. The representation of the object resource may be contained in the same document as the described resource (Inline), or

it may be contained in a remote document (Reference), or it may be either (Either). These values correspond to [oslc:Inline](#), [oslc:Reference](#), and [oslc:Either](#).

Range

For object properties, the allowed `rdf:type(s)` of the object resources. This corresponds to the [oslc:range](#) property. This value may be Any if any type is allowed. The value Any corresponds to [oslc:Any](#).

Description

A description of the defined property. This corresponds to the [dcterms:title](#) property.

oslc:instanceShape Property

The `oslc:instanceShape` property is used to link any described resource with a shape resource that describes its contents.

oslc:resourceShape Property

The `oslc:resourceShape` property is used to link an application service description with a shape resource that describes some aspect of the service's API contract. A service description **MAY** be linked with zero or more shapes.

For example, in OSLC a resource that accepts POST requests to create new resources is referred to as a *creation factory*. The service description for a creation factory may link to one or more shape resources that describe the bodies of POST requests.

dcterms:description Property

`dcterms:description` is used to provide a description of [oslc:Property](#) resources. Its value **SHOULD** be a literal of type `rdf:XMLLiteral` that is valid content for an XHTML `<div>` element. If the value contains no XML markup then it **MAY** be represented as a plain text literal or `xsd:string`.

dcterms:title Property

`dcterms:title` is used to provide a summary of [oslc:ResourceShape](#) and [oslc:Property](#) resources. Its value **SHOULD** be a literal of type `rdf:XMLLiteral` that is valid content for an XHTML `` element. If the value contains no XML markup then it **MAY** be represented as a plain text literal or `xsd:string`.

8.1 ResourceShape Constraints

- **Name:** `ResourceShape`
- **URI:** `http://open-services.net/ns/core#ResourceShape`
- **Summary:** A shape resource describes the contents of and constraints on some set of described resources.

ResourceShape Properties

Prefix Name	Occurs	Read-only	Value-type	Representation	Range	Description
<code>dcterms:title</code>	Zero-or-one	true	XMLLiteral	Either	Unspecified	The summary of this shape.
<code>oslc:describes</code>	Zero-or-many	true	Resource	Reference	Unspecified	The described resource types that this shape applies to.
<code>oslc:hidden</code>	Zero-or-one	true	boolean	Either	Unspecified	Indicates the resource or property should not be displayed to users.
<code>oslc:property</code>	Zero-or-many	true	Resource	Inline	Unspecified	The defined properties that are expected to be contained in the described resources associated with this shape.

oslc:describes Property

`oslc:describes` is used to list the types of the described resources associated with this shape. Suppose that shape S is associated with described resource R, e.g. via an [oslc:resourceShape](#) or [oslc:instanceShape](#) link. If shape S describes type T and described resource R has type T then S describes the contents of and constraints on R.

For example, a creation factory may be able to create many different types of resources. The description of a given type of resource is specified by the associated shape resources that contain an [oslc:describes](#) link to that type.

oslc:property Property

`oslc:property` is used to list the defined properties that are expected to be contained in described resources associated with this

shape. The object of this property **MUST** be an [oslc:Property](#) resource whose representation is contained in the shape document.

If a described resource contains a property described by some [oslc:Property](#) resource, then a REST service is expected to process that property in some useful way as defined by the service's API contract. If there is no matching [oslc:Property](#) resource then the behavior of the service is undefined.

8.2 Property Constraints

- **Name:** `Property`
- **URI:** `http://open-services.net/ns/core#Property`
- **Summary:** Specifies the name, description, summary, occurrence, value type, allowed values, and several other aspects of the defined property.

Property Properties

<i>Prefixed Name</i>	<i>Occurs</i>	<i>Read-only</i>	<i>Value-type</i>	<i>Representation</i>	<i>Range</i>	<i>Description</i>
<code>dcterms:description</code>	Zero-or-one	true	XMLLiteral	Either	Unspecified	The description of the defined property.
<code>dcterms:title</code>	Zero-or-one	true	XMLLiteral	Either	Unspecified	The summary of the defined property.
<code>oslc:allowedValue</code>	One-or-many	true	string	Either	Unspecified	An allowed value of a defined property.
<code>oslc:allowedValues</code>	Zero-or-one	true	Resource	Reference	<code>oslc:AllowedValues</code>	The resource containing a set of allowed values of the defined property.
<code>oslc:defaultValue</code>	Zero-or-one	true	string	Either	Unspecified	The default value of the defined property.
<code>oslc:hidden</code>	Zero-or-one	true	boolean	Either	Unspecified	Indicates the resource or property should not be displayed to users.
<code>oslc:isMemberProperty</code>	Zero-or-one	true	boolean	Either	Unspecified	If true then the described resource is a container and the defined property is used for container membership.
<code>oslc:maxSize</code>	Zero-or-one	true	integer	Either	Unspecified	For string datatype properties, the maximum number of characters.
<code>oslc:name</code>	Exactly-one	true	string	Either	Unspecified	The local name of the defined property.
<code>oslc:occurs</code>	Exactly-one	true	Resource	Reference	Unspecified	The number of times the defined

<i>Prefixed Name</i>	<i>Occurs</i>	<i>Read-only</i>	<i>Value-type</i>	<i>Representation</i>	<i>Range</i>	<i>Description</i>
						property may occur.
<code>oslc:propertyDefinition</code>	Exactly-one	true	Resource	Reference	Unspecified	The URI of the defined or constrained property.
<code>oslc:range</code>	One-or-many	true	Resource	Reference	Unspecified	For object properties, an allowed object resource type.
<code>oslc:readOnly</code>	Zero-or-one	true	boolean	Either	Unspecified	n/a If true then the defined property cannot be directly written by clients, but may be updated indirectly by servers.
<code>oslc:representation</code>	Zero-or-one	true	Resource	Reference	Unspecified	For object properties, how the object resource is represented in the representation of the described resource.
<code>oslc:valueShape</code>	Zero-or-many	true	Resource	Reference	Unspecified	For object properties, the URI of a shape resource that describes the object resource.
<code>oslc:valueType</code>	Zero-or-one	true	Resource	Reference	Unspecified	The type of values of the defined property.

oslc:allowedValue Property

`oslc:allowedValue` is used to specify an allowed value of the defined property. The object of this property **SHOULD** be compatible with the type specified by the `oslc:valueType` property if present. An `oslc:Property` resource **MAY** contain one or more `oslc:allowedValue` properties and an optional `oslc:allowedValues` property which links to an `oslc:AllowedValues` resource. The complete set of allowed values is the union of all the values specified directly in the `oslc:Property` resource and the linked `oslc:AllowedValues` resource.

oslc:allowedValues Property

`oslc:allowedValues` specifies a link to an `oslc:AllowedValues` resource which defines a set of allowed values for the defined property. See `oslc:allowedValue` for a description of how the complete set of allowed values is defined.

oslc:defaultValue Property

`oslc:defaultValue` specifies the default value for the defined property. The object of this property **SHOULD** be compatible with the

type specified by the [oslc:valueType](#) property if present.

A default value is normally used when creating resources. A service **SHOULD** use the default value to provide a value for a property if none is provided in the creation request.

A default value **MAY** be used by consumers of a described resource if the defined property is not present in the representation of the described resource. This mechanism is useful if a service introduces a new defined property but does not update all pre-existing described resources.

oslc:hidden Property

If present and true, `oslc:hidden` is used to indicate that the defined property is not normally presented to users. A consumer of the described resource **SHOULD NOT** display hidden defined properties to normal users. It **MAY** display hidden defined properties to administrative users.

oslc:isMemberProperty Property

If the `oslc:isMemberProperty` is present and true then the defined property is a container membership property similar to `rdfs:member`. The described resource is the container and the object resources are its members.

For example, OSLC Query Capabilities are query services that behave like containers for other resources. A defined property for which `oslc:isMemberProperty` is true links the container to its member resources.

The recent Linked Data Platform specification [[LDP](#)] elaborates the concept of resource container. It is therefore desirable to evolve the Shapes specification to align with the LDP concept of container membership. This topic is discussed below in [Linked Data Platform Containers](#).

oslc:name Property

`oslc:name` is used to specify the local name of the defined property. This is normally the portion of the defined property URI (see [oslc:propertyDefinition](#)) that follows the last hash (#) or slash (/).

oslc:maxLength Property

For datatype properties whose type is `xsd:string`, `oslc:maxLength` specifies the maximum number of characters in the defined property value. The absence of `oslc:maxLength` indicates that either there is no maximum size or that the maximum size is specified some other way.

oslc:occurs Property

`oslc:occurs` is used to specify the number of times that the defined property may occur. The value of this property **MUST** be one of the following individuals:

`oslc:Exactly-one`

The defined property **MUST** occur exactly once. It is required and single-valued.

`oslc:One-or-many`

The defined property **MUST** occur at least once. It is required and multi-valued.

`oslc:Zero-or-many`

The defined property **MAY** occur any number of times. It is optional and multi-valued.

`oslc:Zero-or-one`

The defined property **MUST** occur no more than once. It is optional and single-valued.

oslc:propertyDefinition Property

`oslc:propertyDefinition` is used to specify the URI of the the defined property.

oslc:range Property

`oslc:range` **MUST NOT** be used with datatype properties. It **MAY** be used with object properties. For object properties, `oslc:range` is used to specify an allowed `rdf:type` of the object resource. The value of this property **MAY** be any `rdf:type` URI or the following individual:

`oslc:Any`

This value specifies that there is no constraint on the type of the object resource.

oslc:readOnly Property

If present and true, `oslc:readOnly` is used to specify that the value of defined property is managed by the service, i.e. that it is read-only. It cannot be directly modified by consumers of the service. Services **SHOULD** ignore attempts to modify read-only properties, but **MAY** fail such requests. If a service ignores an attempt to modify a read-only property then it **SHOULD NOT** do so silently. A service **MAY** use the HTTP Warning header or some other means to indicate that the attempt to modify a read-only property has been ignored.

In this context, modification means a change in any object of the triples associated with the defined property. For example, a GET request followed by a PUT request would not modify the triples. A service **MUST NOT** interpret a PUT request that does not modify the triples associated with the defined property as a violation of the `oslc:readOnly` constraint.

Examples of read-only properties include creation and modification timestamps, the identity of who created or modified the described resource, and properties computed from the values of other properties.

When modifying a resource, it is natural for a consumer to first retrieve its current representation using a GET request. This request will return read-only properties along with read-write properties. If a service fails PUT requests that contain read-only properties then consumers will have to remove all read-only properties before submitting PUT requests. The behavior of ignoring read-only properties in PUT requests is therefore more convenient for consumers. Similarly, when copying a resource, a consumer would GET it first. It is therefore more convenient for consumers if the service ignores read-only properties also in POST requests.

oslc:representation Property

For object properties, `oslc:representation` is used to specify how the object resource is represented. The value of `oslc:representation` **MUST** be one of the following individuals:

oslc:Either

There is no constraint on the representation of the object resource.

oslc:Inline

The representation of the object resource **MUST** be present in the representation of the described resource.

oslc:Reference

The representation of the object resource **MUST NOT** be present in the representation of the described resource.

oslc:valueShape Property

For object properties, `oslc:valueShape` is used to specify a link to resource shape that describes the object resource.

oslc:valueType Property

Literal Value Types

For datatype properties, `oslc:valueType` specifies the literal value type. It **MUST** be one of the following individuals:

rdf:XMLLiteral

An XML fragment.

xsd:boolean

A boolean.

xsd:dateTime

A date-time.

xsd:decimal

A decimal number.

xsd:double

A double precision floating point number.

xsd:float

A single precision floating point number.

xsd:integer

An integer.

xsd:string

A string.

Resource Value Types

For object properties, `oslc:valueType` specifies how the object resource is identified. It **MUST** be one of the following individuals:

`oslc:AnyResource`

The object resource **MUST** be identified with either a URI or a blank node.

`oslc:LocalResource`

The object resource **MUST** be identified with a blank node. The term “local resource” is used because the scope of identifier is local to the representation.

`oslc:Resource`

The object resource **MUST** be identified with a URI.

8.3 AllowedValues Constraints

- **Name:** `AllowedValues`
- **URI:** `http://open-services.net/ns/core#AllowedValues`
- **Summary:** Defines a set of allowed values for a defined property.

AllowedValues Properties

<i>Prefixed Name</i>	<i>Occurs</i>	<i>Read-only</i>	<i>Value-type</i>	<i>Representation</i>	<i>Range</i>	<i>Description</i>
<code>oslc:allowedValue</code>	One-or-many	true	string	Either	Unspecified	An allowed value of a defined property.

9. Recommended Extensions

This section is non-normative.

This section describes some recommended extensions that have been identified through experience gained with using the Shapes specification in practice.

ext: Extensions Vocabulary

In the following sections the prefix `ext:` denotes the vocabulary URI for extensions `http://open-services.net/ns/core-ext#`.

The implicit assumption that the described resource appears as a subject node works well when the described resource is implemented using common programming technologies. Most programming languages have the concept of a structure, record, or class which naturally maps to a resource. The fields of the structure, record, or class naturally map to property-value pairs which become triples in RDF. However, this assumption fails to capture the full generality of RDF in two important cases: inverse properties and disconnected graphs. The following section describes the inverse property case. The section after that describes the disconnected graph case.

9.1 Inverse Properties

This section is non-normative.

In some cases the best RDF representation of a property-value pair may reuse a pre-existing property in which the described resource is the object and the property value is the subject. The reuse of properties is a best practice for enabling data interoperability. The fact that a pre-existing property might have the opposite direction should not be used as a justification for the creation of a new inverse property. In fact, the existence of both inverse and direct properties makes writing efficient queries more difficult since both the inverse and the direct property must be included in the query.

For example, suppose we are describing test cases and want to express the relations between test cases and the requirements that they validate. Further suppose that there is a pre-existing vocabulary for requirements that defines the property `ex:isValidatedBy` which asserts that the subject is validated by the object. In this case there is no need to define the inverse property `ex:validates`. Instead the representation of test case resources should use `ex:isValidatedBy` with the test case as the object and the requirement as the subject. This situation cannot be described by the current Shapes specification.

ext:isInverseProperty Property

A simple fix for this limitation is to introduce a new optional, single-valued, boolean datatype property on `oslc:Property` resources, e.g. `ext:isInverseProperty`. If present and true then the defined property is used in the opposite direction, i.e. the described resource is the object and the object resource is the subject.

ext:inverseProperty Property

Another possible fix for this limitation is to introduce a new optional, reference property on `oslc:Property` resources, `ext:inversePropertyName`. If a Property has a resource value type, then `ext:inversePropertyName` suggests a name for the inverse property. Tools that query incoming links could use this value to display the link in a more meaningful way.

9.2 Disconnected Graphs

This section is non-normative.

There is no requirement for a resource to be present as either the subject or object of any triple in its RDF representation. In practice the resource will appear as the subject in triples that describe the metadata of the resource, e.g. using terms from the Dublin Core [[DublinCore](#)] vocabulary. However, the resource may include other triples that are unrelated to the resource.

For example, consider birth certificates. A birth certificate is an information resource that contains statements about real-world objects, namely people. Consider a birth certificate for John Doe, and suppose that the URI of the birth certificate is `http://example.org/birthcertificate/johndoe`. Further suppose that John Doe is identified by the URI `http://example.org/id/person/johndoe`. John Doe's birth certification might contain the following triples:

```
<http://example.org/birthcertificate> dcterms:created "2013-10-31T10:42Z"^^xsd:dateTime .
<http://example.org/id/person/johndoe> a foaf:Person ;
    foaf:name "John Doe" ,
    foaf:birthday "10-29" .
```

Fig. 3 A disconnected graph.

The current Shapes specification is incapable of describing the birth certificate resource since the node for the person `http://example.org/id/person/johndoe` is disconnected from the node for birth certificate `http://example.org/birthcertificate/johndoe`. To work around this limitation, a resource designer is forced to introduce another triple as follows:

```
<http://example.org/birthcertificate> dcterms:created "2013-10-31T10:42Z"^^xsd:dateTime ;
    foaf:primaryTopic <http://example.org/id/person/johndoe> .
<http://example.org/id/person/johndoe> a foaf:Person ;
    foaf:name "John Doe" ,
    foaf:birthday "10-29" .
```

Fig. 4 A connected graph.

Adding another triple to connect the graph works in this simple example, and, arguably, might be a better design. However, this approach may not scale to more complex graphs. Furthermore, the shape author may not have the ability to modify the resource design. It is therefore desirable to remove this limitation.

ext:NodeShape Class

One design for handling disconnected nodes is to introduce new resources that describe them. Let `ext:NodeShape` be the `rdf:type` of a described node within the graph. The object of an `oslc:valueShape` property could be a resource of `rdf:type` either `oslc:ResourceShape` or `ext:NodeShape`.

ext:nodeShape Property

Let `ext:nodeShape` define the containment relation between a resource of `rdf:type` `oslc:ResourceShape` and a resource of `rdf:type` `ext:NodeShape`. A resource of `rdf:type` `ext:NodeShape` would itself be a shape except that it would not have `ext:nodeShape` properties, i.e. all disconnected nodes would be described at the top level of the graph.

9.3 Datatype Facets

This section is non-normative.

The current specification only allows the use of a small number of XML Schema datatypes. In practice, it is often necessary to further limit the set of allowed values. The XML Schema Datatypes specification [[XML Schema Datatypes](#)] defines a set of constraining facets which can be used to further restrict datatypes.

For example, suppose it is required to restrict the decimal property `ex:price` to two digits of fractional precision. In XML Schema, this requirement can be satisfied through the use of the `xsd:fractionDigits` facet as follows:

```
<xsd:element name="price">
  <xsd:simpleType>
    <xsd:restriction base="xsd:decimal">
      <xsd:fractionDigits value="2" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Fig. 5 XML Schema `xsd:fractionDigits` facet.

A straightforward way to extend the current Shapes specification is to allow XML Schema constraining facets as properties of

datatype properties. The presence of a facet property would be defined as being semantically equivalent to using a restriction of the base datatype specified by [oslc:valueType](#). The above ex:price example would be expressed as follows:

```
<#ex-price> a oslc:Property ;
  oslc:propertyDefinition ex:price ;
  oslc:name "price" ;
  oslc:occurs oslc:Exactly-one ;
  oslc:valueType xsd:decimal ;
  xsd:fractionDigits 2 .
```

Fig. 6 ex:price restricted using the xsd:fractionDigits facet.

Here is the complete list of XML Schema constraining facets as defined in *4.3 Constraining Facets* [[XML Schema Datatypes](#)]:

xsd:length

Specifies the exact number of characters in a string value. The string value **MUST** contain the specified exact number of characters.

xsd:minLength

Specifies the minimum number of characters in a string value. The string value **MUST** contain at least the specified minimum number of characters.

xsd:maxLength

Specifies the maximum number of characters in a string value. The string value **MUST** contain at most the specified maximum number of characters. Equivalent to using [oslc:maxSize](#).

xsd:pattern

Specifies a regular expression for a string value. The string value **MUST** match the regular expression.

This facet bears further discussion. First, is XSD regular expression language identical to that used in the SPARQL `regex()` function? SPARQL uses the `fn:matches()` function which is defined in XQuery 1.0 and XPath 2.0 [[XQuery and XPath](#)]. Second, the SPARQL `regex()` function accepts flags as an optional third argument. An additional property, `ext:patternFlags`, should be used to specify these flags.

xsd:enumeration

Specifies set of allowed values. The value **MUST** be contained in the specified set of allowed values. Not suitable for Shapes since it introduces complex XSD syntax and is redundant with [oslc:allowedValue](#). Use [oslc:allowedValue](#) instead.

xsd:whiteSpace

Constrains how whitespace is handled in a string value. Not applicable to Shapes.

xsd:maxInclusive

Specifies the inclusive maximum for a numeric value. The numeric value must be less than or equal to the specified maximum value.

xsd:maxExclusive

Specifies the exclusive maximum value for a numeric value. The numeric value must be less than the specified maximum value.

xsd:minExclusive

Specifies the exclusive minimum value for a numeric value. The numeric value must be greater than the specified minimum value.

xsd:minInclusive

Specifies the inclusive minimum value for a numeric value. The numeric value must be greater than or equal to the specified minimum value.

xsd:totalDigits

Specifies the total number of digits in a decimal value. The decimal value **MUST** contain at most the specified total number of digits.

xsd:fractionDigits

Specifies the number of digits in the fractional part of a decimal value. The decimal value **MUST** contain at most the specified number of digits in its fractional part.

9.4 Extending Shapes

This section is non-normative.

RDF specifications often support an open content model. For example, a bug tracking specification may define some base properties, such as severity and priority, but allow implementations to add custom properties, such as security impact and estimate cost to fix.

Now suppose an implementation of some base specification provides a shape resource that describes the extended resource design. With the current Shapes specification, all of the defined properties of the base shape would have to be copied into the extended shape. Clearly, it would simplify maintenance of shapes if the extended shape could link to the base shape and only define the custom properties.

ext:extendsShape Property

The ext:extendsShape property links an extended shape to a base shape, e.g. a base specification shape. The meaning of this link is that all of the content of the base shape is effectively copied into the extended shape. The explicit content of the extended shape would override the copied content of the base shape. However, care should be taken to avoid violating the integrity constraints of the base shape. Any resource that is valid with respect to the extended shape **SHOULD** also be valid with respect to the base shape. Validation with respect to the base shape **SHOULD** ignore the custom properties.

9.5 SPARQL Semantics

This section is non-normative.

This specification uses natural language to define the semantics of shape resources. In many cases, the semantics could be expressed using SPARQL queries [SPARQL 1.1]. Using SPARQL would both improve the precision of the specification and provide a starting point for reference implementations and test suites. The use of SPARQL in this context does not imply any restriction on the choice of other implementation technologies.

Given a shape and a described resource, it is of interest to determine if the described resource satisfies the constraints defined by the shape. Each constraint must be checked. If any fails, then the described resource does not satisfy the shape.

Many of the constraints can be expressed as SPARQL ASK queries. These queries return a true/false result. A query that returns true if the constraint is satisfied is called an *assertion*. A query that returns false if a constraint is satisfied is called an *exception*. Depending on the constraint it may be easier to express it one way or another. In addition, if a constraint is violated, then it is of interest to know how it is violated. A query that explains why a query is violated is called a *diagnostic*. Diagnostics can be expressed as SPARQL SELECT or SPARQL CONSTRUCT queries. If all assertions return true, all exceptions return false, and all diagnostics return nothing, then the described resource satisfies the constraints.

For example, suppose we create an RDF dataset for the simple bug tracking described above. Each bug is stored as a named graph in the dataset. Consider the occurrence constraint on the oslc_cm:status property. This constraint states that the oslc_cm:status property occurs at most once in each bug. The following listing contains a SPARQL diagnostic query that returns all bugs that violate this occurrence constraint, along with the actual occurrence count:

```
prefix oslc_cm: <http://open-services.net/ns/cm#>

select ?bug (count(?status) as ?status_count)
where {
  graph ?bug {
    ?bug a oslc_cm:ChangeRequest ;
        oslc_cm:status ?status
  }
}
group by ?bug
having (?status_count > 1)
```

Fig. 7 Listing 10. A diagnostic SPARQL query for an [oslc:Zero-or-one](#) occurrence constraint.

A. Acknowledgements

Additional Contributors:

The OSLC Resource Shape specification was initially developed by the OSLC Reporting Workgroup under the leadership of Tack Tong (IBM), with major contributions from Arthur Ryman (IBM) and Martin Nally (IBM). Members of OSLC workgroups found shapes to be applicable to other aspects of OSLC and so the specification was subsequently integrated into the OSLC Core specification by Dave Johnson (IBM).

Anamitra Bhattacharyya (IBM) provided valuable feedback based on implementation experience and raised the requirement for [Datatype Facets](#). Steve Speicher (IBM) and John Arwe (IBM) raised the requirement for [Extending Shapes](#).

Reviewers:

The following people carefully reviewed drafts of this document, provided corrections, and suggested many valuable improvements:

- John Arwe (IBM)
- Nick Crossley (IBM)
- Miguel Esteban Gutiérrez (Universidad Politécnica de Madrid)
- Arnaud Le Hors (IBM)
- Eric Prud'hommeaux (W3C)
- Steve Speicher (IBM)