Standards Track Work Product

OASIS OPEN PROJECTS

OSLC Core Version 3.0. Part 1: Overview

Project Specification Draft 04
20 December 2019

This stage:

https://docs.oasis-open-projects.org/oslc-op/core/v3.0/psd04/oslc-core.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/core/v3.0/psd04/oslc-core.pdf

Previous stage:

http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csprd03/part1-overview/oslc-core-v3.0-csprd03-part1-overview.html (Authoritative)
http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csprd03/part1-overview/oslc-core-v3.0-csprd03-part1-overview.pdf

Latest stage:

https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.pdf

Latest version:

https://open-services.net/spec/core/latest

Latest editor's draft:

https://open-services.net/spec/core/latest-draft

Open Project:

OASIS OSLC OP

Project Chairs:

Jim Amsden (jamsden@us.ibm.com), IBM
Andrii Berezovskyi (andriib@kth.se), KTH

Editor:

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

Additional components:

This specification is one component of a Work Product that also includes:

- *OSLC Core Version 3.0. Part 1: Overview (this document)*. oslc-core.html
- *OSLC Core Version 3.0. Part 2: Discovery*. discovery.html
- *OSLC Core Version 3.0. Part 3: Resource Preview*. resource-preview.html
- *OSLC Core Version 3.0. Part 4: Delegated Dialogs*. dialogs.html
- *OSLC Core Version 3.0. Part 5: Attachments*. attachments.html
- *OSLC Core Version 3.0. Part 6: Resource Shape*. resource-shape.html
- *OSLC Core Version 3.0. Part 7: Vocabulary*. core-vocab.html

Related work:

This specification is related to:

- *OSLC Core Version 3.0: Link Guidance*. Work in progress. Current draft: https://oslc-op.github.io/oslc-specs/notes/link-guidance.html

RDF Namespaces:

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

Abstract:

Defines the overall approach to Open Services for Lifecycle Collaboration (OSLC) based specifications and capabilities that extend and complement W3C Linked Data Platform [LDP]. OSLC Core 3.0 constitutes the approach outlined in this document and capabilities referenced in other documents.

Status:

This document was last revised or approved by the OASIS OSLC OP on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Open Project are listed at https://open-services.net/about/.

Comments on this work can be provided by opening issues in the project repository or by sending email to the project's public comment list oslc-op@lists.oasis-open-projects.org.

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:
[OSLC-Core-3.0-Part1]
OSLC Core Version 3.0. Part 1: Overview. Edited by Jim Amsden. 20 December 2019. OASIS Project Specification Draft 04. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/psd04/oslc-core.html. Latest stage: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.html.

Notices

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.

Table of Contents

1. Introduction

This section is non-normative.

Information Technology (IT) enterprises are constantly addressing demands to do more with less. To meet this demand they need more efficient development processes and supporting tools. This has resulted in demand for better support of integrated system and software processes. Enterprises want solutions (such as software or hardware development tools) from different vendors, open source projects and their own proprietary components to work together. This level of integration, however, can become quite challenging and unmanageable. In order to enable integration between a heterogeneous set of tools and components from various sources, there is a need for a sufficient supporting architecture that is loosely coupled, minimal, and standardized. OSLC is based on World Wide Web and Linked Data principles, such as those defined in the W3C Linked Data Platform [LDP], to create a cohesive set of specifications that can enable products, services, and other distributed network resources to interoperate successfully [LDP].

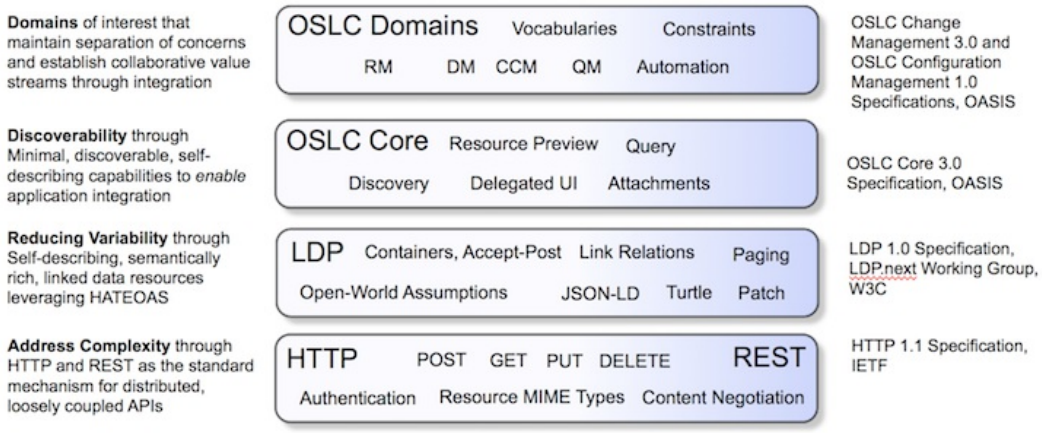

Fig. 1 OSLC Core 3.0 Architecture

OSLC is motivated by domain-driven scenarios that inspire standardization of common capabilities across disciplines such as change management, requirements management, and quality management, as well as by cross-domain scenarios such as Application Lifecycle Management (ALM) & DevOps, Product Lifecycle Management (PLM), and Integrated Service Management (ISM). The OSLC approach focuses on software lifecycle management to ensure it meets a core set of scenarios and requirements. Nonetheless, it can be used by tools belonging to any other domains and cross-domain scenarios such as Internet of Things, back office application integration, and customer relationship management.

The OSLC Core specifications provide additional capabilities that expand on the W3C LDP capabilities, as needed, to enable key integration scenarios. These capabilities define the essential and common technical elements of OSLC domain specifications and offer guidance on common concerns for creating, updating, retrieving, and linking to lifecycle resources based on W3C [LDP].

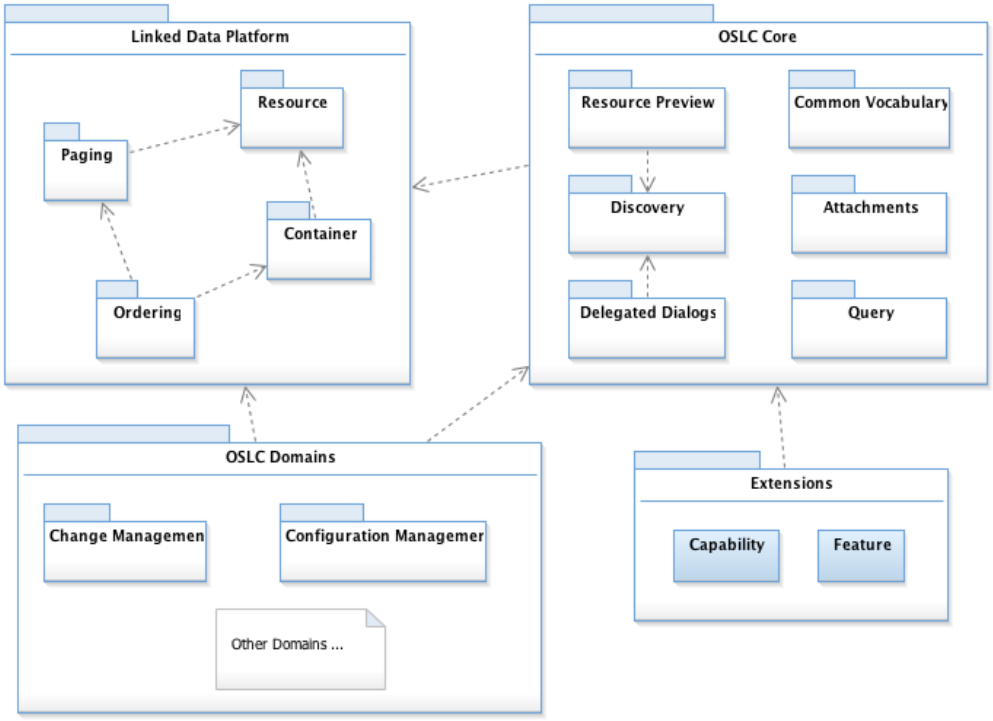

Fig. 2 OSLC Core 3.0 Overview

As seen in Fig. 2 OSLC Core 3.0 Overview, there are a number of capabilities developed in different standards organizations and working groups. The arrows represent either dependencies or extensions to some specifications or capabilities. OSLC domain specifications may depend on OSLC Core 3.0 specifications as scenarios motivate. However, a leading goal is to minimize and eliminate unnecessary dependencies to simplify adoption, which may result in no dependency on OSLC Core 3.0 specifications for some OSLC domains.

This work is an evolution from the OSLC Core 2.0 [OSLCCore2] efforts, taking the experience gained from that effort along with the common foundation on W3C LDP, to produce an updated set of specifications that are simpler, built on layered capabilities and easier to adopt.

1.1 Terminology

Terminology uses and extends the terminology and capabilities of W3C Linked Data Platform [LDP], W3C's Architecture of the World Wide Web [WEBARCH] and Hyper-text Transfer Protocol [HTTP11].

OSLC Server

LDP Server that also supports capabilities defined by at least on OSLC-based specification. See Server [HTTP11] and LDP Server [LDP].

OSLC Client

LDP Client that uses capabilities defined by some OSLC-based specifications. See Client [HTTP11] and LDP Client [LDP]. A particular software component or application could be an OSLC Server supporting a set of domains, and an OSLC Client of other domains depending on its needs.

Domain

A topic area of a specific focus area and/or collection of disciplines. Often OASIS OSLC-affiliated TCs are organized around a domain.

OSLC Core Specifications

Specifications that cover specific capabilities that are often needed across various domains. They are created, authored and endorsed by the OASIS OSLC Open Project. Can be abbreviated to Core Specifications.

OSLC Domain Specifications

Specifications that cover a domain need, including existing open-services.net specifications and new specifications created and authored by OASIS OSLC-affiliated TCs. Can be abbreviated to Domain Specifications

Resource Shape

The set of properties (triples) that constrain a resource for specific operations (i.e. creation, update or query), and for each property, their value types, allowed values and cardinality.

Some industry terms that are often referred to (not exhaustive):

Product Lifecycle Management (PLM)

The process of managing the entire lifecycle of a product from its conception, through design and manufacture, to service and disposal.

Systems Engineering

An interdisciplinary field of engineering that focuses on how to design and manage complex engineering systems over their life cycles.

Application Lifecycle Management (ALM)

The marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, quality and release management.

DevOps

A software development method that stresses communication, collaboration and integration between software developers and Information Technology(IT) professionals in support of continuous delivery.

IT Service Management (ITSM)

The implementation and management of quality information technology services. IT service management is performed by IT service providers through people, process and information technology.

1.1.1 Deprecated terms

Previous revisions of OSLC-based specifications [OSLCCore2], used terminology that may no longer be relevant, accurate or needed any more. Some of those deprecated terms are:

Provider (*deprecated*)

See Server [HTTP11].

Consumer (*deprecated*)

See Client [HTTP11].

1.2 References

1.2.1 Normative references

[HTTP11]

R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7230.html

[LDP]

Steve Speicher; John Arwe; Ashok Malhotra. *Linked Data Platform 1.0*. 26 February 2015. W3C Recommendation. URL: https://www.w3.org/TR/ldp/

[LDPPaging]

S. Speicher; J. Arwe; A. Malhotra. *Linked Data Platform Paging 1.0*. Working Group Note. URL: https://dvcs.w3.org/hg/ldpwg/raw-

file/default/ldp-paging.html

[OSLCCCM1]

Nick Crossley. *OSLC Configuration Management 1.0*. Draft. URL: https://raw.githack.com/oslc-op/oslc-specs/master/specs/config/oslc-config-mgt.html

[OSLCCore2]

S. Speicher; D. Johnson. *OSLC Core 2.0*. Finalized. URL: https://open-services.net/bin/view/Main/OslcCoreSpecification

[OSLCQuery3]

Jim Amsden; S. Padgett; S. Speicher; David Honey. *OSLC Query Version 3.0*. Public Review Draft Project Specification. URL: http://open-services.net/specifications/query/oslc-query.html

[OSLCShapes]

Arthur Ryman; Jim Amsden. *OSLC Resource Shape 3.0*. Project Specification. URL: http://open-services.net/specifications/core/resource-shape.html

[OSLCTRS3]

Nick Crossley, Axel Reichwein. *OSLC Tracked Resource Set Version 3.0*. Draft. URL: https://raw.githack.com/oslc-op/oslc-specs/master/specs/trs/tracked-resource-set.html

[OpenIDConnect]

OpenID Connect. URL: http://openid.net/connect/

[RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[rfc6749]

D. Hardt, Ed.. *The OAuth 2.0 Authorization Framework*. October 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6749

1.2.2 Informative references

[HTML5]

Ian Hickson; Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Theresa O'Connor; Silvia Pfeiffer. *HTML5*. 27 March 2018. W3C Recommendation. URL: https://www.w3.org/TR/html5/

[LDBestPractices]

Linked Data Best Practices. URL: https://jazz.net/wiki/bin/view/LinkedData/BestPractices

[SHACL]

Holger Knublauch; Arthur Ryman. *Shapes Constraint Language (SHACL)*. Draft. URL: https://w3c.github.io/data-shapes/shacl/

[Turtle]

Eric Prud'hommeaux; Gavin Carothers. *RDF 1.1 Turtle*. 25 February 2014. W3C Recommendation. URL: https://www.w3.org/TR/turtle/

[WEBARCH]

Ian Jacobs; Norman Walsh. *Architecture of the World Wide Web, Volume One*. 15 December 2004. W3C Recommendation. URL: https://www.w3.org/TR/webarch/

[rdf11-concepts]

Richard Cyganiak; David Wood; Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. 25 February 2014. W3C Recommendation. URL: https://www.w3.org/TR/rdf11-concepts/

1.3 RDF Namespaces

OSLC Core defines the namespace URI of `http://open-services.net/ns/core#` with a namespace prefix of `oslc`.

OSLC Core uses the following prefixes:

Prefix	Namespace
dcterms	http://purl.org/dc/terms/
foaf	http://xmlns.com/foaf/0.1/
ldp	http://www.w3.org/ns/ldp#
owl	http://www.w3.org/2002/07/owl#
prov	http://www.w3.org/ns/prov#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
vann	http://purl.org/vocab/vann/
vs	http://www.w3.org/2003/06/sw-vocab-status/ns#
xsd	http://www.w3.org/2001/XMLSchema#

1.4 Typographical Conventions and Use of RFC Terms

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].

2. Goals/Motivation

This section is non-normative.

The primary goal of OSLC is to enable integration of federated, shared information across tools that support different, but related domains. OSLC was initially focused on development of Information Technology (IT) solutions involving processes, activities, work products and supporting tools for Application Lifecycle Management (ALM). However, OSLC capabilities could be applicable to other domains. The specific goals for OSLC Core 3.0 are to build on the existing OSLC Core 2.0 specifications to further facilitate the development and integration of domains and supporting tools that address additional integration needs. Specifically:

- Integration is based on an open standard, and not controlled by any single vendor
- OSLC 3.0 is based on the new W3C Linked Data Platform standard which provides a solid foundation for reading and writing linked data resources
- The specifications are simpler, more consistent and will potentially be more attractive to, and easier to consume by new integrations
- There are some new capabilities including Attachments, inverse link labels, traceability and impact types
- Domain vocabularies can be improved for data consistency and removing data gaps
- All Resource Shapes are provided in machine readable [Turtle] files

The following guiding principles were used to govern the evolution of OSLC and guide the development of the OSLC Core 3.0 specifications.

Scenario-driven

Every capability should be linked back to key integration scenarios that motivate its need. These are important not only for knowing that the correct specification content is being developed but also to assist with implementers understanding the intended usage and in developing relevant test cases.

Incremental

Specifications should be developed in an incremental fashion that not only validates the technical approaches but also delivers integration value sooner.

Loose-coupling

Specifications should support a model where clients have little to no knowledge about server implementation-specific behaviors in order to support key integration scenarios. As a result, clients should be unaffected by any server application software or data model implementation changes. Similarly, client software should be able to be independently changed without changes to server software.

Minimalistic

Specification authors should strive to find not only the simplest solution that would work for a given scenario but allows for easy adoption. Authors should avoid solutions that offer additional capabilities which may inhibit adoption of necessary capabilities.

Capability Based

A capability is the ability to perform actions to achieve outcomes described by scenarios through the use of specific technologies. Capabilities should be incrementally defined as independent focused specifications and independently discoverable at runtime. Even though there may be some generally agreed upon best practices for capability publication and discovery, each capability should define how it is published and discovered. The Core OSLC capabilities are defined in this specification.

Vocabularies

Various OSLC MS-affiliated TCs, or any specification development body that is authoring specifications for specific domains of knowledge, should minimally define vocabularies and the semantics behind the various terms. Some consideration should be given for global reuse when terms are used for cross domain queries and within other domain resource shape definitions. Domain specifications are the definition of an OSLC capability, and how those vocabulary terms are used in LDP interactions by both the clients and servers of that capability. The specification should include defining resource shapes that describe resources based on a set of vocabulary terms, which introduces any domain specific constraints on the vocabulary's usage.

OSLC domain vocabularies should follow the recommended best practices for managing RDF vocabularies described at [LDBestPractices].

3. Architecture

This section is non-normative.

In support of the previously stated goals and motivation, it is desired to have a consistent and recommended architecture. The architecture needs to support scenarios requiring a protocol to access, create, update and delete resources. [LDP] is the foundation for this protocol. Resources need to relate, or link, to one another utilizing a consistent, standard and web-scale data model. Resource Description Framework (RDF) [rdf11-concepts] is the foundation for this. The ability to work with these data models over HTTP protocols, is based on [LDP].

Some scenarios require the need to integrate user interface components: either within a desktop or mobile web-browser, mobile device application, or rich desktop applications. For these scenarios the technology is rapidly evolving and changing. Priority should be based on existing standards such as HTML5, with use of `iframe` and `postMessage()`. [HTML5]

OSLC Core specification documents elaborate on the conformance requirements leveraging these various technologies and approaches.

As the primary goals have been outlined around lifecycle integration, some scenarios may require exploration of new (or different) approaches and technologies. As with all specification development efforts, the OSLC Open Project will advise, develop and approve such efforts.

4. OSLC Core 3.0 Capabilities

This section is non-normative.

The following sections and referenced documents define the capabilities for OSLC Core 3.0. These documents comprise the multi-part specification for OSLC Core 3.0. They represent common capabilities that servers may provide and that may be discovered and used by clients. Although OSLC Core could be useful on its own, it is intended to specify capabilities that are common across many domains. Servers will generally specify conformance with specific domain specifications, and those domain specifications will describe what parts of OSLC Core are required for conformance. This allows servers to implement the capabilities they need in a standard way without the burden of implementing capabilities that are not required. The purpose of the OSLC Core Discovery capability is to allow clients to determine what capabilities are provided by a server. Any provided capability must meet all the conformance criteria for that capability as defined in the OSLC Core 3.0 specifications.

This implies that any capability that is discoverable is essentially optional, and once discovered, the capability is provided as defined in the applicable OSLC specifications. Servers should support OSLC Discovery, but Discovery itself is also an optional capability as servers could provide other means of informing specific clients of supported OSLC capabilities that could be utilized directly. For example, a server might provide only preview dialogs on specific resources and nothing else.

4.1 Resource Constraints

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 that handle RDF data such as form and query builders.

Shapes are different than vocabularies in that shapes may change with new revisions of resource definitions, whereas vocabularies should evolve in place in a compatible manner.

Constraints on OSLC Core and Domain resources **SHOULD** be described using [OSLCShapes] which is included as part of the OSLC Core multi-part specifications. Servers **MAY** use other constraint languages such as [SHACL] to define resource constraints. [core-1]

OSLC Domain specifications **SHOULD** use the following URI pattern when publishing each individual resource shape:

```
http://open-services.net/ns/[vocab short name]/shapes/[version]/[shape-name]
```

[core-2]

For example, for Change Management 3.0, a shape describing the base Change Request resource type might have the shape URI:

```
http://open-services.net/ns/cm/shapes/3.0/changerequest
```

Not all shapes would necessarily be updated at the same time.

To allow different versions of individual shapes to be reused in different versions of a domain specification while still allowing a client to browse the set of possible shapes, domains **SHOULD** provide an LDPC for all the shapes for a spec version, at a URI defined by the following pattern:

```
http://open-services.net/ns/[vocab short name]/shapes/[SPEC-version]
```

[core-3]

For example, for Change Management 3.0, there should be a container at: `http://open-services.net/ns/cm/shapes/3.0` with members such as:

```
http://open-services.net/ns/cm/shapes/3.0/changerequest
http://open-services.net/ns/cm/shapes/3.0/somenewshape
http://open-services.net/ns/cm/shapes/2.0/unchangedshape
```

4.2 Authentication

Authentication determines how a user of a client identifies themselves to a server to ensure the user has sufficient privileges to access

resources from that server, and provides a mechanism for servers to control access to resources.

OSLC 3.0 servers **MAY** protect resources with HTTP Basic Authentication. OSLC Services that use HTTP Basic Authentication **SHOULD** do so only via SSL. [core-4]

OSLC 3.0 servers **SHOULD** protect resources with [rfc6749] Authentication utilizing [OpenIDConnect]. [core-5]

4.3 Resource Discovery

Resource Discovery defines a common approach for HTTP/LDP-based servers to be able to publish their RESTful API capabilities and how clients can discover and use them.

4.4 Resource Representations

OSLC resource representations come in many forms and are subject to standard HTTP and mechanisms for content negotiation.

OSLC domain specifications specify the representations needed for the specific scenarios that they are addressing, and should recognize that different representations are appropriate for different purposes. For example, browser oriented scenarios might be best addressed by JSON or Atom format representations.

OSLC domain specifications are also expected to follow common practices and conventions that are in concert with existing industry standards and which offer consistency across domains. All of the OSLC specifications are built upon the standard RDF data model, allowing OSLC to align with the Linked-Data Platform [LDP]. In addition, all OSLC specifications have adopted the convention to illustrate most examples using Turtle and/or JSON-LD representations and will typically require these representations to enable consistency across OSLC implementations.

OSLC Services **MUST** support at least one RDF resource serialization format, and **SHOULD** support as many serialization formats as possible through content negotiation. [core-6]

OSLC Services **SHOULD** provide and accept RDF documents in Turtle format (identified by the MIME-type 'text/turtle') and in JSON-LD format (identified by the MIME-type 'application/ld+json') representations for each OSLC resource for compatibility with LDP 1.0. [core-7]

OSLC Services **SHOULD** provide and accept RDF/XML representations for each OSLC resource to preserve compatibility with [OSLCCore2]. [core-8]

OSLC Services **MAY** provide and accept existing standard or emerging standard formats such as XML, HTML, and the Atom Syndication Format. [core-9]

4.5 Common Vocabulary

Common Vocabulary Terms defines a number of commonly used RDF vocabulary terms and resources (shapes), that have broad applicability across various domains.

4.6 Resource Operations

Resource Operations specify how clients create, read, update and delete resources managed by servers.

OSLC Core defines a set of HTTP REST services for a set of capabilities that facilitate flexible, loosely coupled tool integration. These services may be augmented by varous OSLC domain specifications that specify the capabilities, vocabularies and constraints that define specific resources managed by these integration capabilities.

OSLC Servers **MUST** implement standard HTTP protocols and **MUST** at least support `GET` requests that respond with resources in some RDF serialization format. [core-10]

OSLC Services use HTTP for create, retrieve, update and delete operations on resources. OSLC Services **MUST** comply with the HTTP specification [HTTP11]. [core-11]

Because the update process may involve first getting a resource, modifying it and then later putting it back to the server, there is the possibility of a conflict, e.g. some other client may have have updated the resource since the GET. To mitigate this problem, OSLC Services **SHOULD** use the HTTP `If-Match` header on a PUT request: [core-12]

- If the HTTP `If-Match` header is missing OSLC Services **SHOULD** return HTTP Bad Request (400) status code to indicate that the header is required. [core-13]
- If the HTTP `If-Match` header is present OSLC Services **MUST** behave as described in the HTTP specification, returning an HTTP

Precondition Failed (412) error to indicate that the header does not match. [core-14]

- If the HTTP `If-Match` header is present and it matches, but there is some other problem or conflict with the update then OSLC Services **MAY** return an HTTP Conflict (409) to indicate that problem. [core-15]

An OSLC Service **SHOULD** preserve property values that are not part of the resource definition or Resource Shape known by the server (unknown property values). An OSLC Service **MUST** return a 4xx status code if it decides not to persist any of the unknown property values (in accordance with the LDP specification §4.2.4.4). [core-16]

An OSLC Client **MUST** preserve any unknown property values between requests to the OSLC Services for all HTTP verbs except PATCH (in accordance with the LDP specification §4.3.1.11). [core-17]

4.7 Selective Properties

OSLC Services **MAY** support a technique called Selective Properties to enable clients to retrieve only selected property values. [core-18]

By adding the key=value pair `oslc.properties`, specified below, to a resource URI, a client can request a new resource with a subset of the original resource's values. An additional key=value pair `oslc.prefix` can be used to define prefixes used to identify the selected properties.

The `oslc.properties` key=value pair lets you specify the set of properties to be included in the response. Both immediate and nested properties may be specified. A nested property is a property that belongs to the resource referenced by another property. Nested properties are enclosed in brace brackets, and this nesting may be done recursively, i.e. a nested property may contain other nested properties.

For example, suppose we have a bug report resource at the following URL:

Example 1: Bug Report URL

```
http://example.com/bugs/4242
```

Suppose this bug resource has properties such as `dcterms:title`, `dcterms:description`, and `dcterms:creator`, and that `dcterms:creator` refers to a person resource that has properties such as `foaf:givenName` and `foaf:familyName`. Suppose you want a representation of the bug report that includes its `dcterms:title` and the `foaf:givenName` and `foaf:familyName` of the person referred to by its `dcterms:creator`. The following URL illustrates the use of the `oslc.properties` query value to include those properties:

Example 2: Selective Properties URL

```
http://example.com/bugs/4242?oslc.properties=dcterms:title,dcterms:creator{foaf:givenName,foaf:familyName}
```

The `oslc.properties` pair is defined by the oslc_properties term in the following BNF grammar:

```
oslc_properties ::= "oslc.properties=" properties
properties      ::= property ("," property)*
property        ::= identifier | wildcard | nested_prop
nested_prop     ::= (identifier | wildcard) "{" properties "}"
wildcard        ::= "*"
identifier      ::= PrefixedName
PrefixedName    ::= /* see "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/#rPrefixedName */
```

In our examples of `oslc.properties`, property names include a URI prefix, i.e. dcterms: or foaf:. Here we assume that OSLC has predefined the Dublin Core (dcterms:) and Friend of a Friend (foaf:) prefixes. However, OSLC resources should also be open to new content, which means that new properties may not have predefined URI prefixes. We therefore need a way to define new URI prefixes in resource requests. The oslc.prefix value lets you specify URI prefixes used in property names. For example, suppose the foaf: prefix was not predefined. The following URL illustrates the use of the oslc.prefix value to define it:

Example 3: Example URL with oslc.prefix

```
http://example.com/bugs/4242?oslc.prefix=foaf=<http://xmlns.com/foaf/0.1/>&oslc.properties=foaf:lastName,...
```

The syntax of the `oslc.prefix` is defined by the oslc_prefix term in the following BNF grammar:

```
oslc_prefix ::= "oslc.prefix=" prefix_defs
prefix_defs ::= prefix_def ("," prefix_def)*
prefix_def  ::= prefix "=" uri_ref_esc
prefix      ::= PN_PREFIX
PN_PREFIX   ::= /* see "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/#rPN_PREFIX */
uri_ref_esc ::= /* an angle bracket-delimited URI reference in which > and \ are \-escaped. */
```

OSLC Core specifies a number of predefined PrefixDefinitions for convenience. OSLC Domain specifications may specify additional pre-defined PrefixDefinitions for their purposes.

An OSLC Server **SHOULD** support the following PrefixDefinitions:

- dcterms: <http://purl.org/dc/terms/>
- foaf: <http://xmlns.com/foaf/0.1/>
- owl: <http://www.w3.org/2002/07/owl#>
- rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- xsd: <http://www.w3.org/2001/XMLSchema#>
- rdfs: <http://www.w3.org/2000/01/rdf-schema#>
- ldp: <http://www.w3.org/ns/ldp#>
- oslc: <http://open-services.net/ns/core#>
- oslc_acc: <http://open-services.net/ns/core/acc#>
- trs: <http://open-services.net/ns/core/trs#>

4.8 Resource Preview

Resource Preview specifies a technique to get a minimal HTML representation of a resource identified by a URL. Applications often use this representation to display a link with an appropriate icon, a label, or display a small or large preview when a user makes some gesture over a link.

4.9 Delegated Dialogs

Delegated Dialogs allow one application to embed a creation or selection UI into another using HTML `iframe` elements and JavaScript code. The embedded dialog notifies the parent page of events using HTML5 `postMessage`.

4.10 Query

OSLC servers will often manage large amounts of potentially complex link-data entities. Practical use of this information will require some query capability that minimally supports selection of matching elements, filtering of desired properties and ordering. OSLC Core defines a query capability that is relatively simple, can be implemented on a wide range of existing server architectures, and provides a standard, data source independent query mechanism. The purpose of this query capability is to support tool integration through a common query mechanism.

OSLC Servers **MAY** support a *Query Capability* as defined in [OSLCQuery3] to enable clients to perform selection and projection operations in order to retrieve a selected subset of resources and property values from an LDPC. [core-19]

4.11 Resource Paging

When a client requests a resource, the client should expect that the entire resource will be returned in the response, with all property values. This can be problematic because, in some cases, resources may be so large that a client might not want to retrieve the entire resource in one HTTP response.

One solution for clients that are sensitive to response size is to check the response size before loading. This method is described in the next section. Another solution is to use Resource Paging.

Resource Paging specifies a capability for servers to make the state of large resources or long lists of resources available as a list of smaller subset resources (pages) whose representation is easier to produce by the server and consume by clients. Resource paging is particularly useful in handling results from the query capability or the contents of an LDP container.

4.11.1 Checking Response Size

Clients that do not wish to load large resources **MAY** use the HTTP HEAD method to determine the size of a resource. [core-20]

When responding to an HTTP HEAD request, according to [HTTP11] the server **SHOULD** include an HTTP Content-Length header that indicates the size of the resource as the "decimal number of octets." [core-21]

4.11.2 Unstable Paging

Because HTTP is a stateless protocol and OSLC Services manage resources that can change frequently, OSLC clients should assume that the resources returned on a given page may change over time.

4.11.3 Stable Paging

Some OSLC Servers might wish to guarantee stable paging, meaning that the sequence of pages returned from the server represent a snapshot in time and will not change as the client pages through them. OSLC specifications that require stable paging should state this requirement and specify to which resources it applies.

Note that because stable paging implementations are based on server-side state, it is possible that the state will expire. Implementations **MAY** use the HTTP response code 410 (Expired) to indicate to clients that the link they requested has expired. [core-22]

4.11.4 Response Information

A response info resource representation describes information about a paged HTTP response body in which it appears.

Resource representations returned via Resource Paging **MUST** include a resource of type oslc:ResponseInfo, as defined in part 7 of this multi-part specification: Vocabulary. [core-23]

The subject resource URI of the oslc:ResponseInfo resource representation **MUST** be the HTTP request URI or the URI from subsequent redirects. The response representation **MAY** also include properties from subject resources different from the one identified by the request URI. [core-24]

Below is an example using the OSLC Core RDF/XML representation guidance, that illustrates how the oslc:ResponseInfo resource representation is included in addition to the blog entry resource representation.

Example 4: Resource Paging, partial response with response info resource representation

```
GET http://example.com/blogs/entry/1?oslc.paging=true&amp;pageno=2

<rdf:RDF
   xmlns:oslc_blog="http://open-services.net/ns/bogus/blogs#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://http://xmlns.com/foaf/0.1/"
   xmlns:dcterms="http://purl.org/dc/terms/">

   <oslc_blog:Entry
      rdf:about="http://example.com/blogs/entry/1">
      &lt;!-- partial property values of of the blog entry --&gt;
   </oslc_blog:Entry>

   <oslc:ResponseInfo rdf:about="http://example.com/blogs/entry/1?oslc.paging=true&amp;pageno=2">
      <oslc:nextPage rdf:resource="http://example.com/blogs/entry/1?oslc.paging=true&amp;pageno=3" />
   </oslc:ResponseInfo>

</rdf:RDF>
```

Refer to the OSLC vocabulary specification for further information on the ResponseInfo resource.

4.11.5 Using Resource Paging

OSLC Services **MAY** support [LDPPaging] to enable clients to retrieve large LDP resources (LDPRs) a page at a time. [core-25]

OSLC Services **SHOULD** support OSLC paging as described in this section to ensure compatibility with OSLC 2.0 server implementations. [core-26]

To request a paged version of a resource, a client **MUST** add at least one "key=value" pair to the query component of the resource URI: Either oslc.paging=true, or oslc.pageSize, or both. [core-27]

When responding to a request that includes oslc.paging=true in the URI, a server **MAY** return a representation that contains a subset of the resource's property values. [core-28]

By adding oslc.pageSize to the query component of the resource URI, the client requests that the server respond with a specific number of property values. For example, oslc.pageSize=20 indicates to the server that the client would like 20 values per page.

When responding to a request that includes oslc.pageSize in the URI, a server **SHOULD** return a representation that contains the requested number of property values. [core-29]

When Resource Paging is used, the values of a multi-valued property **MAY** be split across resource pages. [core-30]

When Resource Paging is used, each property value **MUST** be represented in its entirety and not split across multiple partial resource pages. [core-31]

When a page is returned and it is NOT the last page in the sequence, then it **SHOULD** include an oslc:ResponseInfo (see above), which contains a resource-valued property oslc:nextPage that links to a resource that represents the next page of property-values. [core-32]

When paging is unstable, by the time a client follows an oslc:nextPage link there may no longer be a next page, in this case the server **MAY** respond with an HTTP 404 Page Not Found status code. [core-33]

4.11.6 Provider Response Size Limitations

When a client requests a resource that an OSLC Service considers to be too large to return in one response and the client has not indicated that it desires paging (via oslc.paging or oslc.pageSize), the OSLC Service **MAY** redirect the client to a representation that contains partial information about the resource. [core-34]

When the OSLC Service opts to redirect the client to a partial representation, it **MUST** return an HTTP Status 302 redirect to a URL that includes either oslc.paging, or oslc.pageSize, or both. [core-35]

If the URL includes oslc.pageSize then the value should be a value that is acceptable to the service. The client may choose to follow the redirect and receive a representation that contains partial information about the resource.

On receiving a resource representation, OSLC Clients should check for the presence of an oslc:nextPage value to determine if the representation contains partial information about the resource. If the value is present, then paging is in effect and the representation contains partial information about the resource.

4.12 Attachments

Attachments describes a minimal way to manage attachments related to web resources using LDP-Containers and Non-RDF Source [LDP].

4.13 Tracked Resource Sets

OSLC defines a Tracked Resource Set capability that allows servers to expose a set of resources in a way that enables clients to discover the exact set of resources in the set, to track ongoing changes affecting resources in the set. This allows OSLC servers to expose a live feed of linked data in a way that permits clients to build, aggregate, and maintain live, searchable information based on that linked data.

OSLC Servers **MAY** support a *Tracked Resources Set* capability as defined in [OSLCTRS3] to enable OSLC data consumers and providers flexible ways of sharing information. [core-36]

4.14 Configuration Management

OSLC defines a Configuration Management capability for managing versions and configurations of linked data resources from multiple domains. Using client and server applications that implement the configuration management capability, a team administrator can create configurations of versioned resources contributed from tools and data sources across the lifecycle. These contributions can be assembled into aggregate (global) configurations that are used to resolve references to artifacts in a particular and reproducible context.

OSLC Clients and Servers **MAY** support a *Configuration Management* capability as defined in [OSLCCCM1]. [core-37]

4.15 Error Responses

Error responses returned by servers in response to requests are defined in Common Vocabulary Terms, Errors.

5. Version Compatibility

This section is non-normative.

OSLC is intended to provide a foundation for (lifecycle) application interoperability. A significant number of OSLC domains, and client and server implementations already exist and are in common use. Interoperability issues between applications on incompatible OSLC versions could result in negative impact to end users. One of the goals of the OSLC initiative is to mitigate or eliminate the need for lock-step version upgrades, where clients or servers target one version of a specification and break when new versions are introduced -- requiring all services to be upgraded simultaneously.

OSLC Core and domain specifications will each be versioned independently, and may specify version numbers. Existing OSLC 2.0 clients and servers use the OSLC-Core-Version header described in OSLC Core 2.0 Specification Versioning to indicate what OSLC version they expect or support. But exposing version numbers in OSLC implementations could lead to interoperability issues. Ultimately each domain will decide its compatibility needs. OSLC Core 3.0 does not introduce any changes that would break existing OSLC 2.0 clients. Because of this, there is no need for OSLC Core 3.0 to require servers or clients to utilize an OSLC-Core-Version header with a value of 3.0.

If an OSLC 2.0 client accesses an OSLC 3.0 server, the 3.0 server will always respond to the client in a manner that is compatible with 2.0. The response may include additional headers and entity request or response body information defined by OSLC Core 3.0, but this information will be simply ignored by the 2.0 clients. There will be no missing or invalid information since OSLC Core 3.0 is designed to be compatible with OSLC Core 2.0.

For OSLC clients that access 2.0 servers:

- OSLC 1.0 clients - get 1.0 responses by default from 2.0 servers and don't interoperate with 3.0 servers.
- OSLC 2.0 clients - interact with 2.0 servers that may or may not implement 1.0 by using the OSLC-Core-Version header.
- OSLC 3.0 Clients - if they are accessing 2.0 servers that might return 1.0 responses without the header, then they would have to provide the header as specified in OSLC Core 2.0 Specification Versioning,

OSLC Core 3.0 does not address compatibility with versions of OSLC prior to [OSLCCore2]. Servers wishing to support compatibility with versions prior to 2.0 should follow OSLC Core 2.0 Specification Versioning.

It is worth noting that in OSLC 2.0 specifications, normative specification text had precedence over machine-readable content, as opposed to this specification.

6. Conformance

OSLC Servers **MUST** implement all the mandatory requirements in the normative sections of specification, and **SHOULD** follow all the guidelines and recommendations in these specifications. [core-38]

OSLC Servers **MAY** implement any of the capabilities described in this specification, and if a capability is supported, all of the mandatory requirements specified in the normative sections of the multi-part specification elaborating the capability **MUST** be implemented. [core-39]

OSLC Servers **MUST** implement the OSLC Core vocabulary as defined in OSLC Core Version 3.0. Part 7: Vocabulary. [core-40]

Part 2 of this document defines the mandatory requirements for the OSLC discovery capability. OSLC Servers **SHOULD** provide the discovery capability in order for clients to determine what services are supported by the server and how to access them. OSLC Servers that implement discovery **MUST** implement all of the mandatory requirements specified in the normative sections of part 2 of this multi-part specification. [core-41]

Part 3 of this document defines the mandatory requirements for the OSLC resource preview capability. OSLC Servers that implement resource preview **MUST** implement all of the mandatory requirements specified in the normative sections of part 3 of this multi-part specification. [core-42]

Part 4 of this document defines the mandatory requirements for the OSLC delegated creation and selection dialog capability. OSLC Servers that implement delegated dialogs **MUST** implement all of the mandatory requirements specified in the normative sections of part 4 of this multi-part specification. [core-43]

Part 5 of this document defines the mandatory requirements for the OSLC attachents capability. OSLC Servers that implement attachments **MUST** implement all of the mandatory requirements specified in the normative sections of part 5 of this multi-part specification. [core-44]

Part 6 of this document defines the mandatory requirements for the OSLC resource shape constraints capability. OSLC Servers **SHOULD** support resource shapes in order to describe OSLC server managed resources, and validate creation and update requests. OSLC Servers that implement resources shapes **MUST** implement all of the mandatory requirements specified in the normative sections of part 6 of this multi-part specification. [core-45]

All conformance clauses are summarized in the following table.

Clause Number	Requirement
core-1	Constraints on OSLC Core and Domain resources **SHOULD** be described using [OSLCShapes] which is included as part of the OSLC Core multi-part specifications. Servers **MAY** use other constraint languages such as [SHACL] to define resource constraints.
core-2	OSLC Domain specifications **SHOULD** use the following URI pattern when publishing each individual resource shape: `http://open-services.net/ns/[vocab short name]/shapes/[version]/[shape-name]`
core-3	To allow different versions of individual shapes to be reused in different versions of a domain specification while still allowing a client to browse the set of possible shapes, domains **SHOULD** provide an LDPC for all the shapes for a spec version, at a URI defined by the following pattern: `http://open-services.net/ns/[vocab short name]/shapes/[SPEC-version]`
core-4	OSLC 3.0 servers **MAY** protect resources with HTTP Basic Authentication. OSLC Services that use HTTP Basic Authentication **SHOULD** do so only via SSL.
core-5	OSLC 3.0 servers **SHOULD** protect resources with [rfc6749] Authentication utilizing [OpenIDConnect].
core-6	OSLC Services **MUST** support at least one RDF resource serialization format, and **SHOULD** support as many serialization formats as possible through content negotiation.
core-7	OSLC Services **SHOULD** provide and accept RDF documents in Turtle format (identified by the MIME-type 'text/turtle') and in JSON-LD format (identified by the MIME-type 'application/ld+json') representations for each OSLC resource for compatibility with LDP 1.0.

Clause Number	Requirement
core-8	OSLC Services **SHOULD** provide and accept RDF/XML representations for each OSLC resource to preserve compatibility with [OSLCCore2].
core-9	OSLC Services **MAY** provide and accept existing standard or emerging standard formats such as XML, HTML, and the Atom Syndication Format.
core-10	OSLC Servers **MUST** implement standard HTTP protocols and **MUST** at least support `GET` requests that respond with resources in some RDF serialization format.
core-11	OSLC Services use HTTP for create, retrieve, update and delete operations on resources. OSLC Services **MUST** comply with the HTTP specification [HTTP11].
core-12	Because the update process may involve first getting a resource, modifying it and then later putting it back to the server, there is the possibility of a conflict, e.g. some other client may have have updated the resource since the GET. To mitigate this problem, OSLC Services **SHOULD** use the HTTP `If-Match` header on a PUT request:
core-13	If the HTTP `If-Match` header is missing OSLC Services **SHOULD** return HTTP Bad Request (400) status code to indicate that the header is required.
core-14	If the HTTP `If-Match` header is present OSLC Services **MUST** behave as described in the HTTP specification, returning an HTTP Precondition Failed (412) error to indicate that the header does not match.
core-15	If the HTTP `If-Match` header is present and it matches, but there is some other problem or conflict with the update then OSLC Services **MAY** return an HTTP Conflict (409) to indicate that problem.
core-16	An OSLC Service **SHOULD** preserve property values that are not part of the resource definition or Resource Shape known by the server (unknown property values). An OSLC Service **MUST** return a 4xx status code if it decides not to persist any of the unknown property values (in accordance with the LDP specification §4.2.4.4).
core-17	An OSLC Client **MUST** preserve any unknown property values between requests to the OSLC Services for all HTTP verbs except PATCH (in accordance with the LDP specification §4.3.1.11).
core-18	OSLC Services **MAY** support a technique called Selective Properties to enable clients to retrieve only selected property values.
core-19	OSLC Servers **MAY** support a *Query Capability* as defined in [OSLCQuery3] to enable clients to perform selection and projection operations in order to retrieve a selected subset of resources and property values from an LDPC.
core-20	Clients that do not wish to load large resources **MAY** use the HTTP HEAD method to determine the size of a resource.
core-21	When responding to an HTTP HEAD request, according to [HTTP11] the server **SHOULD** include an HTTP Content-Length header that indicates the size of the resource as the "decimal number of octets."
core-22	Implementations **MAY** use the HTTP response code 410 (Expired) to indicate to clients that the link they requested has expired.
core-23	Resource representations returned via Resource Paging **MUST** include a resource of type oslc:ResponseInfo, as defined in part 7 of this multi-part specification: Vocabulary.
core-24	The subject resource URI of the oslc:ResponseInfo resource representation **MUST** be the HTTP request URI or the URI from subsequent redirects. The response representation **MAY** also include properties from subject resources different from the one identified by the request URI.
core-25	OSLC Services **MAY** support [LDPPaging] to enable clients to retrieve large LDP resources (LDPRs) a page at a time.
core-26	OSLC Services **SHOULD** support OSLC paging as described in this section to ensure compatibility with OSLC 2.0 server implementations.
core-27	To request a paged version of a resource, a client **MUST** add at least one "key=value" pair to the query component of the resource URI: Either oslc.paging=true, or oslc.pageSize, or both.
core-28	When responding to a request that includes oslc.paging=true in the URI, a server **MAY** return a representation that contains a subset of the resource's property values.
core-29	When responding to a request that includes oslc.pageSize in the URI, a server **SHOULD** return a representation that contains the requested number of property values.
core-30	When Resource Paging is used, the values of a multi-valued property **MAY** be split across resource pages.
core-31	When Resource Paging is used, each property value **MUST** be represented in its entirety and not split across multiple partial resource pages.
core-32	When a page is returned and it is NOT the last page in the sequence, then it **SHOULD** include an oslc:ResponseInfo (see above), which contains a resource-valued property oslc:nextPage that links to a resource that represents the next page of property-values.
core-33	When paging is unstable, by the time a client follows an oslc:nextPage link there may no longer be a next page, in this case the server **MAY** respond with an HTTP 404 Page Not Found status code.

Clause Number	Requirement
core-34	When a client requests a resource that an OSLC Service considers to be too large to return in one response and the client has not indicated that it desires paging (via oslc.paging or oslc.pageSize), the OSLC Service **MAY** redirect the client to a representation that contains partial information about the resource.
core-35	When the OSLC Service opts to redirect the client to a partial representation, it **MUST** return an HTTP Status 302 redirect to a URL that includes either oslc.paging, or oslc.pageSize, or both.
core-36	OSLC Servers **MAY** support a *Tracked Resources Set* capability as defined in [OSLCTRS3] to enable OSLC data consumers and providers flexible ways of sharing information.
core-37	OSLC Clients and Servers **MAY** support a *Configuration Management* capability as defined in [OSLCCCM1].
core-38	OSLC Servers **MUST** implement all the mandatory requirements in the normative sections of specification, and **SHOULD** follow all the guidelines and recommendations in these specifications.
core-39	OSLC Servers **MAY** implement any of the capabilities described in this specification, and if a capability is supported, all of the mandatory requirements specified in the normative sections of the multi-part specification elaborating the capability **MUST** be implemented.
core-40	OSLC Servers **MUST** implement the OSLC Core vocabulary as defined in OSLC Core Version 3.0. Part 7: Vocabulary.
core-41	Part 2 of this document defines the mandatory requirements for the OSLC discovery capability. OSLC Servers **SHOULD** provide the discovery capability in order for clients to determine what services are supported by the server and how to access them. OSLC Servers that implement discovery **MUST** implement all of the mandatory requirements specified in the normative sections of part 2 of this multi-part specification.
core-42	Part 3 of this document defines the mandatory requirements for the OSLC resource preview capability. OSLC Servers that implement resource preview **MUST** implement all of the mandatory requirements specified in the normative sections of part 3 of this multi-part specification.
core-43	Part 4 of this document defines the mandatory requirements for the OSLC delegated creation and selection dialog capability. OSLC Servers that implement delegated dialogs **MUST** implement all of the mandatory requirements specified in the normative sections of part 4 of this multi-part specification.
core-44	Part 5 of this document defines the mandatory requirements for the OSLC attachents capability. OSLC Servers that implement attachments **MUST** implement all of the mandatory requirements specified in the normative sections of part 5 of this multi-part specification.
core-45	Part 6 of this document defines the mandatory requirements for the OSLC resource shape constraints capability. OSLC Servers **SHOULD** support resource shapes in order to describe OSLC server managed resources, and validate creation and update requests. OSLC Servers that implement resources shapes **MUST** implement all of the mandatory requirements specified in the normative sections of part 6 of this multi-part specification.

Appendix A. Acknowledgements

This section is non-normative.

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

James Amsden, IBM (Chair)
Nick Crossley, IBM
Jad El-khoury, KTH Royal Institute of Technology
Ian Green, IBM
David Honey, IBM
Jean-Luc Johnson, Airbus Group SAS
Harish Krishnaswamy, Software AG, Inc.
Arnaud LeHors, IBM
Sam Padget, IBM
Martin Pain, IBM
Arthur Ryman, IBM
Martin Sarabura, PTC (Chair)
Steve Speicher, IBM

Appendix B. Change History

This section is non-normative.

Revision	Date	Editor	Changes Made
01	04 April 2017	Jim Amsden	CS was approved and published.
03	31 May 2018	Jim Amsden	Added predefined prefixes for common namespaces. Relaxed RDF serialization format requirements. Added normative references to OSLC Query, TRS and Configuration Management specifications.
04	26 June 2019	Jim Amsden	Added conformance section and migrated to Open Project

