Linked Data Platform 1.0

W3C Recommendation 26 February 2015

This version:
http://www.w3.org/TR/2015/REC-ldp-20150226/
Latest published version:
http://www.w3.org/TR/ldp/
Latest editor's draft:
http://www.w3.org/2012/ldp/hg/ldp.html
Test suite:
https://dvcs.w3.org/hg/ldpwg/raw-file/default/tests/ldp-testsuite.html
Implementation report:
https://dvcs.w3.org/hg/ldpwg/raw-file/default/tests/reports/ldp.html
Previous version:
http://www.w3.org/TR/2014/PR-ldp-20141216/
Editors:
Steve Speicher, IBM Corporation
John Arwe, IBM Corporation
Ashok Malhotra, Oracle Corporation

Please check the **errata** for any errors or issues reported since publication.

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

Linked Data Platform (LDP) defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Please see the Working Group's implementation report.

This W3C Recommendation was published by the Linked Data Platform Working Group. Discussions of this document are on public-ldp-comments@w3.org (subscribe, archives).

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 14 October 2005 W3C Process Document.

Table of Contents

1. Introduction
2. Terminology
    2.1 Conventions Used in This Document
3. Conformance
4. Linked Data Platform Resources
    4.1 Introduction
    4.2 Resource
    4.3 RDF Source
    4.4 Non-RDF Source
5. Linked Data Platform Containers
    5.1 Introduction
    5.2 Container
    5.3 Basic
    5.4 Direct

5.5 Indirect
6. Notable information from normative references
6.1 Architecture of the World Wide Web
6.2 HTTP 1.1
6.3 RDF
7. HTTP Header Definitions
7.1 The Accept-Post Response Header
7.2 Preferences on the Prefer Request Header
8. Link Relations
8.1 describedby
9. Security Considerations
A. Acknowledgements
B. Change History
C. References
C.1 Normative references
C.2 Informative references

1. Introduction

This section is non-normative.

This specification describes the use of HTTP for accessing, updating, creating and deleting resources from servers that expose their resources as Linked Data. It provides clarifications and extensions of the rules of Linked Data [LINKED-DATA]:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs, so that they can discover more things

This specification discusses standard HTTP and RDF techniques used when constructing clients and servers that create, read, and write Linked Data Platform Resources. LDP Primer provides an entry-level introduction with many examples in the context of a fictional application. LDP Best Practices and Guidelines discusses best practices that you should use, and anti-patterns you should avoid, when constructing these clients and servers.

This specification defines a special type of Linked Data Platform Resource: a Container. Containers are very useful in building application models involving collections of resources, often homogeneous ones. For example, universities offer a collection of classes and a collection of faculty members, each faculty member teaches a collection of courses, and so on. This specification discusses how to work with containers. Resources can be added to containers using standard HTTP operations like POST (see section 5.2.3 HTTP POST).

The intention of this specification is to enable additional rules and layered groupings of rules as additional specifications. The scope is intentionally narrow to provide a set of key rules for reading and writing Linked Data that most, if not all, other specifications will depend upon and implementations will support.

This specification provides some approaches to deal with large resources. An extension to this specification provides the ability to break large resource representations into multiple paged responses [LDP-PAGING].

For context and background, it could be useful to read Linked Data Platform Use Case and Requirements [LDP-UCR] and section 6. Notable information from normative references.

2. Terminology

Terminology is based on W3C's Architecture of the World Wide Web [WEBARCH] and Hyper-text Transfer Protocol ([RFC7230], [RFC7231], [RFC7232]).

Link
A relationship between two resources when one resource (representation) refers to the other resource by means of a URI [WEBARCH].

Linked Data
As defined by Tim Berners-Lee [LINKED-DATA].

Client
A program that establishes connections for the purpose of sending one or more HTTP requests [RFC7230].

Server
A program that accepts connections in order to service HTTP requests by sending HTTP responses.

The terms "client" and "server" refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others.

HTTP enables the use of intermediaries to satisfy requests through a chain of connections. There are three common forms of HTTP intermediary: proxy, gateway, and tunnel. In some cases, a single intermediary might act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request. [RFC7230].

***Linked Data Platform Resource* (LDPR)**
A HTTP resource whose state is represented in any way that conforms to the simple lifecycle patterns and conventions in section 4. Linked Data Platform Resources.

***Linked Data Platform RDF Source* (LDP-RS)**
An LDPR whose state is fully represented in RDF, corresponding to an RDF graph. See also the term RDF Source from [rdf11-concepts].

***Linked Data Platform Non-RDF Source* (LDP-NR)**
An LDPR whose state is *not* represented in RDF. For example, these can be binary or text documents that do not have useful RDF

representations.

Linked Data Platform Container **(LDPC)**
A LDP-RS representing a collection of linked documents (RDF Document [rdf11-concepts] or information resources [WEBARCH]) that responds to client requests for creation, modification, and/or enumeration of its linked members and documents, and that conforms to the simple lifecycle patterns and conventions in section 5. Linked Data Platform Containers.

Linked Data Platform Basic Container **(LDP-BC)**
An LDPC that defines a simple link to its contained documents (information resources) [WEBARCH].

Linked Data Platform Direct Container **(LDP-DC)**
An LDPC that adds the concept of membership, allowing the flexibility of choosing what form its membership triples take, and allows members to be any resources [WEBARCH], not only documents.

Linked Data Platform Indirect Container **(LDP-IC)**
An LDPC similar to a LDP-DC that is also capable of having members whose URIs are based on the content of its contained documents rather than the URIs assigned to those documents.

Membership
The relationship linking an LDPC and its member LDPRs, which can be different resources than its contained documents. The LDPC often assists with managing the membership triples, whether or not the LDPC's URI occurs in them.

Membership triples
A set of triples that lists an LDPC's members. A LDPC's membership triples all have one of the following patterns:

membership-constant-URI	*membership-predicate*	*member-derived-URI*
member-derived-URI	*membership-predicate*	*membership-constant-URI*

The difference between the two is simply which position member-derived-URI occupies, which is usually driven by the choice of *membership-predicate*. Most predicates have a natural forward direction inherent in their name, and existing vocabularies contain useful examples that read naturally in each direction. `ldp:member` and `dcterms:isPartOf` are representative examples.

Each linked container exposes properties (see section 5.2.1 General) that allow clients to determine which pattern it uses, what the actual *membership-predicate* and *membership-constant-URI* values are, and (for containers that allow the creation of new members) what value is used for the *member-derived-URI* based on the client's input to the creation process.

Membership predicate
The predicate of all an LDPC's membership triples.

Containment
The relationship binding an LDPC to LDPRs whose lifecycle it controls and is aware of. The lifecycle of the contained LDPR is limited by the lifecycle of the containing LDPC; that is, a contained LDPR cannot be created (through LDP-defined means) before its containing LDPC exists.

Containment triples
A set of triples, maintained by the LDPC, that lists documents created by the LDPC but not yet deleted. These triples **always** have the form: *(LDPC URI, ldp:contains , document-URI).*

Minimal-container triples
The portion of an LDPC's triples that would be present when the container is empty. Currently, this definition is equivalent to all the LDPC's triples minus its containment triples, and minus its membership triples (if either are considered part of its state), but if future versions of LDP define additional classes of triples then this definition would expand to subtract out those classes as well.

LDP-server-managed triples
The portion of an LDP's triples whose behavior is constrained directly by this specification; for example, membership triples and containment triples. This portion of resources' content does *not* include constraints imposed outside of LDP, for example by other specifications that the server happens to support, or by server implementation decisions.

2.1 Conventions Used in This Document

The namespace for LDP is `http://www.w3.org/ns/ldp#`.

Sample resource representations are provided in `text/turtle` format [turtle].

Commonly used namespace prefixes:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix foaf:    <http://xmlns.com/foaf/0.1/>.
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ldp:     <http://www.w3.org/ns/ldp#>.
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#>.
```

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 MAY, MUST, MUST NOT, RECOMMENDED, SHOULD, and SHOULD NOT are to be interpreted as described in [RFC2119].

The status of the sections of Linked Data Platform 1.0 (this document) is as follows:

- 1. Introduction: **non-normative**
- 2. Terminology: **normative**
- 3. Conformance: **normative**
- 4. Linked Data Platform Resources: **normative**

- 5. Linked Data Platform Containers: **normative**
- 6. Notable information from normative references: **non-normative**
- 7. HTTP Header Definitions: **normative**
- 8. Security Considerations: **non-normative**
- A. Acknowledgements: **non-normative**
- B. Change History: **non-normative**
- C.1 Normative references: **normative**
- C.2 Non-normative references: **non-normative**

A conforming ***LDP client*** is a conforming HTTP client [RFC7230] that follows the rules defined by LDP in section 4. Linked Data Platform Resources and also section 5. Linked Data Platform Containers.

A conforming ***LDP server*** is a conforming HTTP server [RFC7230] that follows the rules defined by LDP in section 4. Linked Data Platform Resources when it is serving LDPRs, and also section 5. Linked Data Platform Containers when it is serving LDPCs. LDP does not constrain its behavior when serving other HTTP resources.

4. Linked Data Platform Resources

4.1 Introduction

This section is non-normative.

Linked Data Platform Resources (***LDPRs***) are HTTP resources that conform to the simple patterns and conventions in this section. HTTP requests to access, modify, create or delete LDPRs are accepted and processed by LDP servers. Most LDPRs are domain-specific resources that contain data for an entity in some domain, which could be commercial, governmental, scientific, religious, or other.

Some of the rules defined in this document provide clarification and refinement of the base Linked Data rules [LINKED-DATA]; others address additional needs.

The rules for Linked Data Platform Resources address basic questions such as:

- What resource representations should be used?
- How is optimistic collision detection handled for updates?
- What should client expectations be for changes to linked-to resources, such as type changes?
- How can the server make it easy for the client to create resources?
- How do I GET the representation of a large resource broken up into pages?

Additional non-normative guidance is available in the LDP Best Practices and Guidelines that addresses questions such as:

- What literal value types should be used?
- Are there some typical vocabularies that should be reused?
- What guidelines exist when interacting with LDPRs that are common but are not universal enough to specify normatively?

The following sections define the conformance rules for LDP servers when serving LDPRs.

LDP-RS's representations may be too big, one strategy is to break up the response representation into client consumable chunks called pages. A separate LDP specification outlines the conformance rules around pagination [LDP-PAGING].

A LDP server can manage two kinds of LDPRs, those resources whose state is represented using RDF (LDP-RS) and those using other formats (LDP-NR). LDP-RSs have the unique quality that their representation is based on RDF, which addresses a number of use cases from web metadata, open data models, machine processable information, and automated processing by software agents [rdf11-concepts]. LDP-NRs are almost anything on the Web today: images, HTML pages, word processing documents, spreadsheets, etc. and LDP-RSs hold metadata associated with LDP-NRs in some cases.

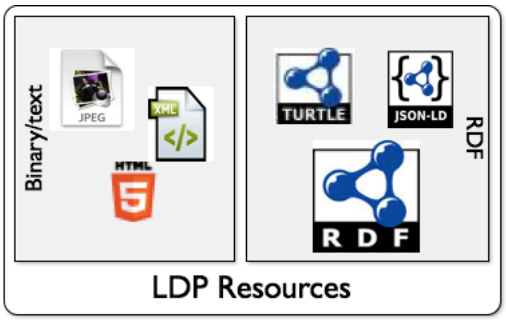

Fig. 1 Samples of different types of LDPRs

The LDP-NRs and LDP-RSs are simply sub-types of LDPRs, as illustrated in Fig. 2 Class relationship of types of Linked Data Platform Resources.

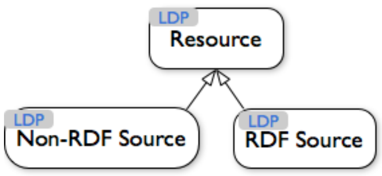

Fig. 2 Class relationship of types of Linked Data Platform Resources

4.2 Resource

4.2.1 General

4.2.1.1 LDP servers MUST at least be HTTP/1.1 conformant servers [RFC7230].

4.2.1.2 LDP servers MAY host a mixture of LDP-RSs and LDP-NRs. For example, it is common for LDP servers to need to host binary or text resources that do not have useful RDF representations.

4.2.1.3 LDP server responses MUST use entity tags (either weak or strong ones) as response `ETag` header values, for responses that contain resource representations or successful responses to HTTP `HEAD` requests.

4.2.1.4 LDP servers exposing LDPRs MUST advertise their LDP support by exposing a HTTP `Link` header with a target URI of `http://www.w3.org/ns/ldp#Resource`, and a link relation type of `type` (that is, `rel="type"`) in all responses to requests made to an LDPR's HTTP `Request-URI` [RFC5988].

> Note: The HTTP `Link` header is the method by which servers assert their support for the LDP specification on a specific resource in a way that clients can inspect dynamically at run-time. This is **not** equivalent to the presence of a *(subject-URI, `rdf:type`, `ldp:Resource`)* triple in an LDP-RS. The presence of the header asserts that the server complies with the LDP specification's constraints on HTTP interactions with LDPRs, that is it asserts that the resource has Etags, supports OPTIONS, and so on, which is not true of all Web resources.
>
> Note: A LDP server can host a mixture of LDP-RSs and LDP-NRs, and therefore there is no implication that LDP support advertised on one HTTP `Request-URI` means that other resources on the same server are also LDPRs. Each HTTP `Request-URI` needs to be individually inspected, in the absence of outside information.

4.2.1.5 LDP servers MUST assign the default base-URI for [RFC3987] relative-URI resolution to be the HTTP `Request-URI` when the resource already exists, and to the URI of the created resource when the request results in the creation of a new resource.

4.2.1.6 LDP servers MUST publish any constraints on LDP clients' ability to create or update LDPRs, by adding a Link header with an appropriate context URI, a link relation of `http://www.w3.org/ns/ldp#constrainedBy`, and a target URI identifying a set of constraints [RFC5988], to all responses to requests that fail due to violation of those constraints. For example, a server that refuses resource creation requests via HTTP PUT, POST, or PATCH would return this `Link` header on its 4xx responses to such requests. The same `Link` header MAY be provided on other responses. LDP neither defines nor constrains the representation of the link's target resource. Natural language constraint documents are therefore permitted, although machine-readable ones facilitate better client interactions. The appropriate context URI can vary based on the request's semantics and method; unless the response is otherwise constrained, the default (the effective request URI) SHOULD be used.

4.2.2 HTTP GET

4.2.2.1 LDP servers MUST support the HTTP `GET` method for LDPRs.

4.2.2.2 LDP servers MUST support the HTTP response headers defined in section 4.2.8 HTTP OPTIONS for the HTTP `GET` method.

4.2.3 HTTP POST

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes no new requirements for LDPRs.

Clients can create LDPRs via `POST` (section 5.2.3 HTTP POST) to a LDPC, via `PUT` (section 4.2.4 HTTP PUT), or any other methods allowed for HTTP resources. Any server-imposed constraints on LDPR creation or update must be advertised to clients.

4.2.4 HTTP PUT

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPRs.

Any server-imposed constraints on LDPR creation or update must be advertised to clients.

4.2.4.1 If a HTTP `PUT` is accepted on an existing resource, LDP servers MUST replace the entire persistent state of the identified resource with the entity representation in the body of the request. LDP servers MAY ignore LDP-server-managed properties, and MAY ignore other properties such as `dcterms:modified` and `dcterms:creator` if they are handled specially by the server (for example, if the server overrides the value or supplies a default value). Any LDP servers that wish to support a more sophisticated merge of data provided by the client with existing state stored on the server for a resource MUST use HTTP `PATCH`, not HTTP `PUT`.

4.2.4.2 LDP servers SHOULD allow clients to update resources without requiring detailed knowledge of server-specific constraints. This is a consequence of the requirement to enable simple creation and modification of LDPRs.

4.2.4.3 If an otherwise valid HTTP `PUT` request is received that attempts to change properties the server does not allow clients to modify, LDP servers MUST fail the request by responding with a 4xx range status code (typically 409 Conflict). LDP servers SHOULD provide a corresponding response body containing information about which properties could not be persisted. The format of the 4xx response body is not constrained by LDP.

> Non-normative note: Clients might provide properties equivalent to those already in the resource's state, e.g. as part of a GET/update

representation/PUT sequence, and those PUT requests are intended to work as long as the LDP-server-managed properties are identical on the GET response and the subsequent PUT request. This is in contrast to other cases like write-once properties that the server does not allow clients to modify once set; properties like this are under client and/or server control but are not constrained by LDP, so they are not LDP-server-managed triples.

4.2.4.4 If an otherwise valid HTTP `PUT` request is received that contains properties the server chooses not to persist, e.g. unknown content, LDP servers MUST respond with an appropriate 4xx range status code [RFC7231]. LDP servers SHOULD provide a corresponding response body containing information about which properties could not be persisted. The format of the 4xx response body is not constrained by LDP. LDP servers expose these application-specific constraints as described in section 4.2.1 General.

4.2.4.5 LDP clients SHOULD use the HTTP `If-Match` header and HTTP `ETags` to ensure it isn't modifying a resource that has changed since the client last retrieved its representation. LDP servers SHOULD require the HTTP `If-Match` header and HTTP `ETags` to detect collisions. LDP servers MUST respond with status code 412 (Condition Failed) if `ETag`s fail to match when there are no other errors with the request [RFC7232]. LDP servers that require conditional requests MUST respond with status code 428 (Precondition Required) when the absence of a precondition is the only reason for rejecting the request [RFC6585].

4.2.4.6 LDP servers MAY choose to allow the creation of new resources using HTTP `PUT`.

4.2.5 HTTP DELETE

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes no new blanket requirements for LDPRs.

Additional requirements on HTTP `DELETE` for LDPRs within containers can be found in section 5.2.5 HTTP DELETE.

4.2.6 HTTP HEAD

Note that certain LDP mechanisms rely on HTTP headers, and HTTP generally requires that `HEAD` responses include the same headers as `GET` responses. Thus, implementers should also carefully read sections 4.2.2 HTTP GET and 4.2.8 HTTP OPTIONS.

4.2.6.1 LDP servers MUST support the HTTP `HEAD` method.

4.2.7 HTTP PATCH

Per [RFC5789], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPRs.

Any server-imposed constraints on LDPR creation or update must be advertised to clients.

4.2.7.1 LDP servers that support `PATCH` MUST include an `Accept-Patch` HTTP response header [RFC5789] on HTTP `OPTIONS` requests, listing patch document media type(s) supported by the server.

4.2.8 HTTP OPTIONS

This specification imposes the following new requirements on HTTP `OPTIONS` for LDPRs beyond those in [RFC7231]. Other sections of this specification, for example PATCH, Accept-Post, add other requirements on `OPTIONS` responses.

4.2.8.1 LDP servers MUST support the HTTP `OPTIONS` method.

4.2.8.2 LDP servers MUST indicate their support for HTTP Methods by responding to a HTTP `OPTIONS` request on the LDPR's URL with the HTTP Method tokens in the HTTP response header `Allow`.

4.3 RDF Source

The following section contains normative clauses for Linked Data Platform RDF Source.

4.3.1 General

4.3.1.1 Each LDP RDF Source MUST also be a conforming LDP Resource as defined in section 4.2 Resource, along with the restrictions in this section. LDP clients MAY infer the following triple: one whose subject is the LDP-RS, whose predicate is `rdf:type`, and whose object is `ldp:Resource`, but there is no requirement to materialize this triple in the LDP-RS representation.

4.3.1.2 LDP-RSs representations SHOULD have at least one `rdf:type` set explicitly. This makes the representations much more useful to client applications that don't support inferencing.

4.3.1.3 The representation of a LDP-RS MAY have an `rdf:type` of `ldp:RDFSource` for Linked Data Platform RDF Source.

4.3.1.4 LDP servers MUST provide an RDF representation for LDP-RSs. The HTTP `Request-URI` of the LDP-RS is typically the subject of most triples in the response.

4.3.1.5 LDP-RSs SHOULD reuse existing vocabularies instead of creating their own duplicate vocabulary terms. In addition to this general rule, some specific cases are covered by other conformance rules.

4.3.1.6 LDP-RSs predicates SHOULD use standard vocabularies such as Dublin Core [DC-TERMS], RDF [rdf11-concepts] and RDF Schema [rdf-schema], whenever possible.

4.3.1.7 In the absence of special knowledge of the application or domain, LDP clients MUST assume that any LDP-RS can have multiple `rdf:type` triples with different objects.

4.3.1.8 In the absence of special knowledge of the application or domain, LDP clients MUST assume that the `rdf:type` values of a given LDP-RS can change over time.

4.3.1.9 LDP clients SHOULD always assume that the set of predicates for a LDP-RS of a particular type at an arbitrary server is open, in the sense that different resources of the same type may not all have the same set of predicates in their triples, and the set of predicates that are used in the state of any one LDP-RS is not limited to any pre-defined set.

4.3.1.10 LDP servers MUST NOT require LDP clients to implement inferencing in order to recognize the subset of content defined by LDP. Other specifications built on top of LDP may require clients to implement inferencing [rdf11-concepts]. The practical implication is that all content defined by LDP must be explicitly represented, unless noted otherwise within this document.

4.3.1.11 A LDP client MUST preserve all triples retrieved from a LDP-RS using HTTP `GET` that it doesn't change whether it understands the predicates or not, when its intent is to perform an update using HTTP `PUT`. The use of HTTP `PATCH` instead of HTTP `PUT` for update avoids this burden for clients [RFC5789].

4.3.1.12 LDP clients MAY provide LDP-defined hints that allow servers to optimize the content of responses. section 7.2 Preferences on the Prefer Request Header defines hints that apply to LDP-RSs.

4.3.1.13 LDP clients MUST be capable of processing responses formed by a LDP server that ignores hints, including LDP-defined hints.

4.3.2 HTTP GET

4.3.2.1 LDP servers MUST respond with a Turtle representation of the requested LDP-RS when the request includes an `Accept` header specifying `text/turtle`, unless HTTP content negotiation *requires* a different outcome [turtle].

> *Non-normative note:* In other words, Turtle must be returned by LDP servers in the usual case clients would expect (client requests it) as well as cases where the client requests Turtle or other media type(s), content negotiation results in a tie, and Turtle is one of the tying media types. For example, if the `Accept` header lists `text/turtle` as one of several media types with the highest relative quality factor (`q=` value), LDP servers must respond with Turtle. HTTP servers in general are not required to resolve ties in this way, or to support Turtle at all, but LDP servers are. On the other hand, if Turtle is one of several requested media types, but another media type the server supports has a higher relative quality factor, standard HTTP content negotiation rules apply and the server (LDP or not) would not respond with Turtle.

4.3.2.2 LDP servers SHOULD respond with a `text/turtle` representation of the requested LDP-RS whenever the `Accept` request header is absent [turtle].

4.3.2.3 LDP servers MUST respond with a `application/ld+json` representation of the requested LDP-RS when the request includes an `Accept` header, unless content negotiation or Turtle support *requires* a different outcome [JSON-LD].

4.4 Non-RDF Source

The following section contains normative clauses for Linked Data Platform Non-RDF Source.

4.4.1 General

4.4.1.1 Each LDP Non-RDF Source MUST also be a conforming LDP Resource in section 4.2 Resource. LDP Non-RDF Sources may not be able to fully express their state using RDF [rdf11-concepts].

4.4.1.2 LDP servers exposing an LDP Non-RDF Source MAY advertise this by exposing a HTTP `Link` header with a target URI of `http://www.w3.org/ns/ldp#NonRDFSource`, and a link relation type of `type` (that is, `rel="type"`) in responses to requests made to the LDP-NR's HTTP `Request-URI` [RFC5988].

5. Linked Data Platform Containers

5.1 Introduction

This section is non-normative.

Many HTTP applications and sites have organizing concepts that partition the overall space of resources into smaller containers. Blog posts are grouped into blogs, wiki pages are grouped into wikis, and products are grouped into catalogs. Each resource created in the application or site is created within an instance of one of these container-like entities, and users can list the existing artifacts within one. Containers answer some basic questions, which are:

1. To which URLs can I POST to create new resources?
2. Where can I GET a list of existing resources?
3. How do I get information about the members along with the container?

4. How can I ensure the resource data is easy to query?
5. How is the order of the container entries expressed? [LDP-PAGING]

This document defines the representation and behavior of containers that address these issues. There are multiple types of containers defined to support a variety of use cases, all that support a base set of capabilities. The contents of a container is defined by a set of triples in its representation (and state) called the containment triples that follow a fixed pattern. Additional types of containers allow for the set of members of a container to be defined by a set of triples in its representation called the membership triples that follow a consistent pattern (see the linked-to definition for the possible patterns). The membership triples of a container all have the same predicate, called the membership predicate, and either the subject or the object is also a consistent value – the remaining position of the membership triples (the one that varies) define the members of the container. In the simplest cases, the consistent value will be the LDPC resource's URI, but it does not have to be. The membership predicate is also variable and will often be a predicate from the server application vocabulary or the **`ldp:member`** predicate.

In LDP 1.0, there exists a way for clients to request responses that contain only partial representations of the containers. Applications may define additional means by which the response representations contain a filtered set of data and including (or excluding) additional details, those means are application-specific and not defined in this document.

This document includes a set of guidelines for creating new resources and adding them to the list of resources linked to a container. It goes on to explain how to learn about a set of related resources, regardless of how they were created or added to the container's membership. It also defines behavior when resources created using a container are later deleted; deleting containers removes membership information and possibly performs some cleanup tasks on unreferenced member resources.

The following illustrates a very simple container with only three members and some information about the container (the fact that it is a container and a brief title):

Request to **`http://example.org/c1/`**:

EXAMPLE 1

```
GET /c1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 2

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "8caab0784220148bfe98b738d5bb6d13"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD,PUT
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

<http://example.org/c1/>
   a ldp:BasicContainer;
   dcterms:title "A very simple container";
   ldp:contains <r1>, <r2>, <r3>.
```

The preceding example is very straightforward: there are containment triples whose subject is the container, whose predicate is **`ldp:contains`**, and whose objects are the URIs of the contained resources, and there is no distinction between member resources and contained resources. A POST to this container will create a new resource and add it to the list of contained resources by adding a new containment triple to the container. This type of container is called a Linked Data Platform Basic Container.

Sometimes you have to build on existing models incrementally, so you have fewer degrees of freedom in the resource model. In these situations, it can be useful to help clients map LDP patterns onto existing vocabularies, or to include members not created via the container; LDP Direct Containers meet those kinds of needs. Direct Containers allow membership triples to use a subject other than the container itself as the consistent membership value, and/or to use a predicate from an application's domain model as the membership predicate.

Let's start with a pre-existing domain resource for a person's net worth, as illustrated immediately below, and then see how a Container resource can be applied in subsequent examples:

Request to **`http://example.org/netWorth/nw1/`**:

EXAMPLE 3

```
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 4

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "0f6b5bd8dc1f754a1238a53b1da34f6b"
```

```
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/>
   a o:NetWorth;
   o:netWorthOf <http://example.org/users/JohnZSmith>;
   o:asset
      <assets/a1>,
      <assets/a2>;
   o:liability
      <liabilities/l1>,
      <liabilities/l2>,
      <liabilities/l3>.
```

In the preceding example, there is a `rdf:type` of `o:NetWorth` indicating the resource represents an instance of a person's net worth and a `o:netWorthOf` predicate indicating the associated person. There are two sets of same-subject, same-predicate triples; one for assets and one for liabilities. Existing domain-specific applications exist that depend on those types and predicates, so changing them *incompatibly* would be frowned upon.

It would be helpful to be able to use LDP patterns to manage the assets and liabilities-related triples. Doing so using a Basic Container would require duplicating much of the information with different types and predicates, which would be onerous for large resources. Direct Containers provide a middle ground, by giving LDP clients a way to map existing domain-specific resources to LDP's types and interaction models. In this specific example, it would be helpful to be able to manage the assets and liabilities triples consistently, for example by using LDP containers. One way to do this is to create two containers, one to manage assets and another liabilities, as separate HTTP resources. Existing clients have no need to interact with those containers, whereas LDP-enabled clients now have container URLs that they can interact with. The existing resource remains unchanged so that existing clients continue to function normally. This is illustrated in the set of related examples, one example per HTTP resource, starting with the LDP-RS example from before:

Request to `http://example.org/netWorth/nw1/`:

EXAMPLE 5

```
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 6

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "0f6b5bd8dc1f754a1238a53b1da34f6b"
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/>
   a o:NetWorth;
   o:netWorthOf <http://example.org/users/JohnZSmith>;
   o:asset
      <assets/a1>,
      <assets/a2>;
   o:liability
      <liabilities/l1>,
      <liabilities/l2>,
      <liabilities/l3>.
```

The structure of the net worth resource is completely unchanged, so any existing domain-specific applications continue to work without impact. LDP clients, on the other hand, have no way to understand that the asset and liability triples correspond in any way to LDP containers. For that, they need the next two resources.

The first container is a LDP Direct Container to manage assets. Direct Containers add the concept of membership and flexibility on how to specify the membership triples.

Request to `http://example.org/netWorth/nw1/assets/`:

EXAMPLE 7

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 8

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "6df36eef2631a1795bfe9ab76760ea75"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/assets/>
   a ldp:DirectContainer;
   dcterms:title "The assets of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:asset;
   ldp:contains <a1>, <a2>.
```

In the preceding asset container, the consistent membership value (membership-constant-URI, still in the subject position) is not the container itself – it is the (separate) net worth resource. The membership predicate is **`o:asset`**, from the domain model. A POST of an asset representation to the asset container will create a new asset and add it to net-worth's list of assets by adding a new membership triple to the resource and a containment triple to the container.

The second container is a LDP Direct Container to manage liabilities.

Request to **`http://example.org/netWorth/nw1/liabilities/`**:

EXAMPLE 9

```
GET /netWorth/nw1/liabilities/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 10

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "9f50da01f792281ddcfebe9788372d07"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/liabilities/>
   a ldp:DirectContainer;
   dcterms:title "The liabilities of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:liability;
   ldp:contains <l1>, <l2>, <l3>.
```

The preceding liability container is completely analogous to the asset container above, simply managing liabilities instead of assets and using the **`o:liability`** predicate.

To add a another liability, a client would POST something like this to the liability container:

Request to **`http://example.org/netWorth/nw1/liabilities/`**:

EXAMPLE 11

```
POST /netWorth/nw1/liabilities/ HTTP/1.1
Host: example.org
Accept: text/turtle
Content-Type: text/turtle
Content-Length: 63

@prefix o: <http://example.org/ontology#>.

<>
   a o:Liability.
   # plus any other properties that the domain says liabilities have
```

Response:

EXAMPLE 12

```
HTTP/1.1 201 Created
Location: http://example.org/netWorth/nw1/liabilities/l4
Date: Thu, 12 Jun 2014 19:56:13 GMT
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

Assuming the server successfully processes this request and assigns the URI `<http://example.org/netWorth/nw1/liabilities/l4>` to the newly created liability resource, at least two new triples would be added.

1. In the net worth resource, `<http://example.org/netWorth/nw1/> o:liability <liabilities/l4>`
2. In the liability container, `<http://example.org/netWorth/nw1/liabilities/> ldp:contains <l4>`.

You might wonder why we chose to create two new containers instead of making `http://example.org/netWorth/nw1/` itself a container. A single net worth container would be a fine design if `http://example.org/netWorth/nw1/` had only assets or only liabilities (basically: only a single predicate to manage), but since it has separate predicates for assets and liabilities an ambiguity arises: it is unspecified whether a client's creation request (POST) should add a new `o:asset` or `o:liability` triple. Having separate `http://example.org/netWorth/nw1/assets/` and `http://example.org/netWorth/nw1/liabilities/` containers allows both assets and liabilities to be created and linked to the net-worth resource using the appropriate predicate. Similar ambiguities arise if the client wishes to list the members and/or contained resources.

Continuing in the multiple containers direction, we will now extend our net worth example to add a container for advisors (people) that have managed the assets and liabilities. We have decided to identify these advisors with URLs that contain a fragment (hash) to represent these real-world resources, not the documents that describe them.

Request to `http://example.org/netWorth/nw1/`:

EXAMPLE 13

```
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

EXAMPLE 14

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "e8d129f45ca31848fb56213844a32b49"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/>
   a o:NetWorth;
   o:netWorthOf <http://example.org/users/JohnZSmith>;
   o:advisor
         <advisors/bob#me>,     # URI of a person
         <advisors/marsha#me>.

<advisors/>
   a ldp:IndirectContainer;
   dcterms:title "The asset advisors of JohnZSmith";
   ldp:membershipResource <>;
   ldp:hasMemberRelation o:advisor;
   ldp:insertedContentRelation foaf:primaryTopic;
   ldp:contains
         <advisors/bob>,     # URI of a document a.k.a. an information resource
         <advisors/marsha>.  # describing a person
```

To handle this type of indirection, the triple with predicate of `ldp:insertedContentRelation` and object of `foaf:primaryTopic` informs clients that when POSTing to this container, they need to include a triple of the form `(<>, foaf:primaryTopic, topic-URI)` to inform the server which URI to use (`topic-URI`) in the new membership triple.

This type of container is referred to as a LDP Indirect Container. It is similar to an LDP Direct Container but it provides an indirection to add (via a create request) as a member any resource, including a URI representing a real-world object. Create requests to LDP Direct Containers can only add information resources [WEBARCH] - the documents they create - as members.

To add a another advisor, a client would POST something like this to the advisors container:

Request to `http://example.org/netWorth/nw1/advisors/`:

EXAMPLE 15

```
POST /netWorth/nw1/advisors/ HTTP/1.1
Host: example.org
```

```
Accept: text/turtle
Content-Type: text/turtle
Slug: george
Content-Length: 72

@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix o: <http://example.org/ontology#>.
<>
   a o:Advisor;
   foaf:primaryTopic <#me>.
   # plus any other properties that the domain says advisors have
```

Response:

EXAMPLE 16

```
HTTP/1.1 201 Created
Location: http://example.org/netWorth/nw1/advisors/george
Date: Thu, 12 Jun 2014 19:56:13 GMT
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

Assuming the server successfully processes this request and assigns the URI `<http://example.org/netWorth/nw1/advisors/george>` to the newly created advisor resource, at least two new triples would be added.

1. In the net worth resource, `<http://example.org/netWorth/nw1/> o:advisor <advisors/george#me>`
2. In the advisors container, `<http://example.org/netWorth/nw1/advisors/> ldp:contains <george>`

In summary, Fig. 3 Class relationship of types of Linked Data Platform Containers illustrates the LDP-defined container types: Basic, Direct, and Indirect, along with their class relationship to types of LDPRs.

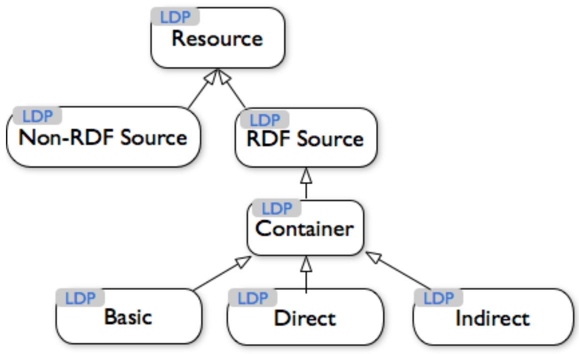

Fig. 3 Class relationship of types of Linked Data Platform Containers

The following table illustrates some differences between membership and containment triples. For details on the normative behavior, see appropriate sections below.

Completed Request	Effects	
	Membership	**Containment**
LDPR created in LDP-BC	New triple: (LDPC, ldp:contains, LDPR)	Same
LDPR created in LDP-DC	New triple links LDP-RS to created LDPR. LDP-RS URI may be same as LDP-DC	New triple: (LDPC, ldp:contains, LDPR)
LDPR created in LDP-IC	New triple links LDP-RS to content indicated URI	New triple: (LDPC, ldp:contains, LDPR)
LDPR is deleted	Membership triple may be removed	(LDPC, ldp:contains, LDPR) triple is removed
LDPC is deleted	Triples and member resources may be removed	Triples of form (LDPC, ldp:contains, LDPR) and contained LDPRs may be removed

5.1.1 Retrieving Only Minimal-Container Triples

The representation of a container that has many members will be large. There are several important cases where clients need to access only the subset of the container's properties that are unrelated to member resources and unrelated to contained documents, for example to determine the membership triple pattern and membership predicate of an LDP-DC. LDP calls these minimal-container triples, because they are what remains when the container has zero members and zero contained resources. Since retrieving the whole container representation to get this information may be onerous for clients and cause unnecessary overhead on servers, we define a way to retrieve only these property values (and more generally, a way to retrieve only the subset of properties used to address other major clusters of use cases). LDP adds parameters to the HTTP `Prefer` header's `return=representation` preference for this (see section 7.2 Preferences on the Prefer Request Header for details).

The example listed here only shows a simple case where few minimal-container triples are retrieved. In real world situations more complex cases are likely, such as those that add other predicates to containers, for example providing validation information and associating SPARQL endpoints. [sparql11-query]

Here is an example requesting the minimal-container triples of a container identified by the URL `http://example.org/container1/`.

Request to `http://example.org/container1/`:

EXAMPLE 17

```
GET /container1/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

Response:

EXAMPLE 18

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

<http://example.org/container1/>
   a ldp:DirectContainer;
   dcterms:title "A Linked Data Platform Container of Acme Resources";
   ldp:membershipResource <http://example.org/container1/>;
   ldp:hasMemberRelation ldp:member;
   ldp:insertedContentRelation ldp:MemberSubject;
   dcterms:publisher <http://acme.com/>.
```

LDP recommends using PATCH to update these properties, if necessary. It provides no facility for updating them via PUT without replacing the entire container's state.

5.2 Container

The following section contains normative clauses for Linked Data Platform Container.

5.2.1 General

The Linked Data Platform does not define how clients discover ***LDPCs***.

5.2.1.1 Each Linked Data Platform Container MUST also be a conforming Linked Data Platform RDF Source. LDP clients MAY infer the following triple: one whose subject is the LDPC, whose predicate is `rdf:type`, and whose object is `ldp:RDFSource`, but there is no requirement to materialize this triple in the LDPC representation.

5.2.1.2 The representation of a LDPC MAY have an `rdf:type` of `ldp:Container` for Linked Data Platform Container. Non-normative note: LDPCs might have additional types, like any LDP-RS.

5.2.1.3 LDPC representations SHOULD NOT use RDF container types `rdf:Bag`, `rdf:Seq` or `rdf:List`.

5.2.1.4 LDP servers exposing LDPCs MUST advertise their LDP support by exposing a HTTP `Link` header with a target URI matching the type of container (see below) the server supports, and a link relation type of `type` (that is, `rel="type"`) in all responses to requests made to the LDPC's HTTP `Request-URI`. LDP servers MAY provide additional HTTP `Link: rel="type"` headers. The notes on the corresponding LDPR constraint apply equally to LDPCs.

Valid container type URIs for `rel="type"` defined by this document are:

- `http://www.w3.org/ns/ldp#BasicContainer` - for LDP Basic Containers
- `http://www.w3.org/ns/ldp#DirectContainer` - for LDP Direct Containers
- `http://www.w3.org/ns/ldp#IndirectContainer` - for LDP Indirect Containers

5.2.1.5 LDP servers SHOULD respect all of a client's LDP-defined hints, for example which subsets of LDP-defined state the client is interested in processing, to influence the set of triples returned in representations of a LDPC, particularly for large LDPCs. See also [LDP-PAGING].

5.2.2 HTTP GET

Per section 4.2.2 HTTP GET the HTTP GET method is required and additional requirements can be found in section 5.2.1 General.

5.2.3 HTTP POST

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on creation or update must be advertised to clients.

5.2.3.1 LDP clients SHOULD create member resources by submitting a representation as the entity body of the HTTP **`POST`** to a known LDPC. If the resource was created successfully, LDP servers MUST respond with status code 201 (Created) and the **`Location`** header set to the new resource's URL. Clients shall not expect any representation in the response entity body on a 201 (Created) response.

5.2.3.2 When a successful HTTP **`POST`** request to a LDPC results in the creation of a LDPR, a containment triple MUST be added to the state of the LDPC whose subject is the LDPC URI, whose predicate is **`ldp:contains`** and whose object is the URI for the newly created document (LDPR). Other triples may be added as well. The newly created LDPR appears as a contained resource of the LDPC until the newly created document is deleted or removed by other methods.

5.2.3.3 LDP servers MAY accept an HTTP **`POST`** of non-RDF representations (LDP-NRs) for creation of any kind of resource, for example binary resources. See the Accept-Post section for details on how clients can discover whether a LDPC supports this behavior.

5.2.3.4 LDP servers that successfully create a resource from a RDF representation in the request entity body MUST honor the client's requested interaction model(s). If any requested interaction model cannot be honored, the server MUST fail the request.

- If the request header specifies a LDPR interaction model, then the server MUST handle subsequent requests to the newly created resource's URI as if it is a LDPR. When the server treats the resource as a LDPR, then clients can only depend upon behaviors defined by LDPRs, even if the content contains an **`rdf:type`** triple indicating a type of LDPC. That is, if the server's statement conflicts with the content's, the server's statement wins.
- If the request header specifies a LDPC interaction model, then the server MUST handle subsequent requests to the newly created resource's URI as if it is a LDPC.
- This specification does not constrain the server's behavior in other cases.

Clients use the same syntax, that is **`HTTP Link`** headers, to specify the desired interaction model when creating a resource as servers use to advertise it on responses.

Note: A consequence of this is that LDPCs can be used to create LDPCs, if the server supports doing so.

Non-normative note: LDP assumes that the interaction model of a resource is fixed when the resource is created, and this is reflected in the language of the bullets. If an implementation were to extend LDP by allowing the interaction model to vary after creation, that is viewed as a compatible extension to LDP and the statements above would constrain the default interaction model rather than saying that no other behavior is possible.

5.2.3.5 LDP servers that allow creation of LDP-RSs via POST MUST allow clients to create new members by enclosing a request entity body with a **`Content-Type`** request header whose value is **`text/turtle`** [turtle].

5.2.3.6 LDP servers SHOULD use the **`Content-Type`** request header to determine the request representation's format when the request has an entity body.

5.2.3.7 LDP servers creating a LDP-RS via POST MUST interpret the null relative URI for the subject of triples in the LDP-RS representation in the request entity body as identifying the entity in the request body. Commonly, that entity is the model for the "to be created" LDPR, so triples whose subject is the null relative URI result in triples in the created resource whose subject is the created resource.

5.2.3.8 LDP servers SHOULD assign the URI for the resource to be created using server application specific rules in the absence of a client hint.

5.2.3.9 LDP servers SHOULD allow clients to create new resources without requiring detailed knowledge of application-specific constraints. This is a consequence of the requirement to enable simple creation and modification of LDPRs. LDP servers expose these application-specific constraints as described in section 4.2.1 General.

5.2.3.10 LDP servers MAY allow clients to suggest the URI for a resource created through **`POST`**, using the HTTP **`Slug`** header as defined in [RFC5023]. LDP adds no new requirements to this usage, so its presence functions as a client hint to the server providing a desired string to be incorporated into the server's final choice of resource URI.

5.2.3.11 LDP servers that allow member creation via **`POST`** SHOULD NOT re-use URIs.

5.2.3.12 Upon successful creation of an LDP-NR (HTTP status code of 201-Created and URI indicated by **`Location`** response header), LDP servers MAY create an associated LDP-RS to contain data about the newly created LDP-NR. If a LDP server creates this associated LDP-RS, it MUST indicate its location in the response by adding a HTTP **`Link`** header with a context URI identifying the newly created LDP-NR (instead of the effective request URI), a link relation value of **`describedby`**, and a target URI identifying the associated LDP-RS resource [RFC5988].

5.2.3.13 LDP servers that support **`POST`** MUST include an **`Accept-Post`** response header on HTTP **`OPTIONS`** responses, listing **`POST`** request media type(s) supported by the server. LDP only specifies the use of **`POST`** for the purpose of creating new resources, but a server can accept **`POST`** requests with other semantics. While "POST to create" is a common interaction pattern, LDP clients are not guaranteed, even when making requests to a LDP server, that every successful **`POST`** request will result in the creation of a new resource; they must rely on out of band information for knowledge of which **`POST`** requests, if any, will have the "create new resource" semantics. This requirement on LDP servers is intentionally stronger than the one levied in the header registration; it is unrealistic to expect all existing resources that support **`POST`** to suddenly return a new header or for all new specifications constraining **`POST`** to be aware of its existence and require it, but it is a reasonable requirement for new specifications such as LDP.

5.2.3.14 LDP servers that allow creation of LDP-RSs via POST MUST allow clients to create new members by enclosing a request entity body with a **`Content-Type`** request header whose value is **`application/ld+json`** [JSON-LD].

5.2.4 HTTP PUT

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on creation or update must be advertised to clients.

5.2.4.1 LDP servers SHOULD NOT allow HTTP `PUT` to update a LDPC's containment triples; if the server receives such a request, it SHOULD respond with a 409 (Conflict) status code.

5.2.4.2 LDP servers that allow LDPR creation via `PUT` SHOULD NOT re-use URIs.

5.2.5 HTTP DELETE

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

5.2.5.1 When a contained LDPR is deleted, the LDPC server MUST also remove the corresponding containment triple, which has the effect of removing the deleted LDPR from the containing LDPC.

> Non-normative note: The LDP server might perform additional actions, as described in the normative references like [RFC7231]. For example, the server could remove membership triples referring to the deleted LDPR, perform additional cleanup tasks for resources it knows are no longer referenced or have not been accessed for some period of time, and so on.

5.2.5.2 When a contained LDPR is deleted, and the LDPC server created an associated LDP-RS (see the LDPC POST section), the LDPC server MUST also delete the associated LDP-RS it created.

5.2.6 HTTP HEAD

Note that certain LDP mechanisms rely on HTTP headers, and HTTP recommends that `HEAD` responses include the same headers as `GET` responses. LDP servers must also include HTTP headers on responses to `OPTIONS`, see section 4.2.8 HTTP OPTIONS. Thus, implementers supporting `HEAD` should also carefully read the section 5.2.2 HTTP GET and section 5.2.8 HTTP OPTIONS.

5.2.7 HTTP PATCH

Per [RFC5789], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on LDPR creation or update must be advertised to clients.

5.2.7.1 LDP servers are RECOMMENDED to support HTTP `PATCH` as the preferred method for updating a LDPC's minimal-container triples.

5.2.8 HTTP OPTIONS

This specification imposes the following new requirements on HTTP `OPTIONS` for LDPCs. Note that support for this method is required for LDPCs, since it is required for LDPRs and LDPCs adhere to LDP-RS requirements.

5.2.8.1 When responding to requests whose `request-URI` is a LDP-NR with an associated LDP-RS, a LDPC server MUST provide the same HTTP `Link` response header as is required in the create response.

5.3 Basic

The following section contains normative clauses for Linked Data Platform Basic Container.

5.3.1 General

5.3.1.1 Each LDP Basic Container MUST also be a conforming LDP Container in section 5.2 Container along with the following restrictions in this section. LDP clients MAY infer the following triple: whose subject is the LDP Basic Container, whose predicate is `rdf:type`, and whose object is `ldp:Container`, but there is no requirement to materialize this triple in the LDP-BC representation.

5.4 Direct

The following section contains normative clauses for Linked Data Platform Direct Container.

5.4.1 General

5.4.1.1 Each LDP Direct Container MUST also be a conforming LDP Container in section 5.2 Container along the following restrictions. LDP clients MAY infer the following triple: whose subject is the LDP Direct Container, whose predicate is `rdf:type`, and whose object is `ldp:Container`, but there is no requirement to materialize this triple in the LDP-DC representation.

5.4.1.2 LDP Direct Containers SHOULD use the `ldp:member` predicate as a LDPC's membership predicate if there is no obvious predicate from an application vocabulary to use. The state of a LDPC includes information about which resources are its members, in the form of membership triples that follow a consistent pattern. The LDPC's state contains enough information for clients to discern the membership predicate, the other consistent membership value used in the container's membership triples (*membership-constant-URI*), and the position (subject or object) where those URIs occurs in the membership triples. Member resources can be any kind of resource identified by a URI, LDPR or otherwise.

5.4.1.3 Each LDP Direct Container representation MUST contain exactly one triple whose subject is the LDPC URI, whose predicate is the **`ldp:membershipResource`**, and whose object is the LDPC's *membership-constant-URI*. Commonly the LDPC's URI is the *membership-constant-URI*, but LDP does not require this.

5.4.1.4 Each LDP Direct Container representation MUST contain exactly one triple whose subject is the LDPC URI, and whose predicate is either **`ldp:hasMemberRelation`** or **`ldp:isMemberOfRelation`**. The object of the triple is constrained by other sections, such as ldp:hasMemberRelation or ldp:isMemberOfRelation, based on the membership triple pattern used by the container.

5.4.1.4.1 LDP Direct Containers whose membership triple pattern is *(membership-constant-URI , membership-predicate , member-derived-URI)* MUST contain exactly one triple whose subject is the LDPC URI, whose predicate is **`ldp:hasMemberRelation`**, and whose object is the URI of *membership-predicate*.

5.4.1.4.2 LDP Direct Containers whose membership triple pattern is *(member-derived-URI , membership-predicate , membership-constant-URI)* MUST contain exactly one triple whose subject is the LDPC URI, whose predicate is **`ldp:isMemberOfRelation`**, and whose object is the URI of *membership-predicate*.

5.4.1.5 LDP Direct Containers MUST behave as if they have a *(LDPC URI, **`ldp:insertedContentRelation`** , **`ldp:MemberSubject`**)* triple, but LDP imposes no requirement to materialize such a triple in the LDP-DC representation. The value **`ldp:MemberSubject`** means that the *member-derived-URI* is the URI assigned by the server to a document it creates; for example, if the client POSTs content to a container that causes the container to create a new LDPR, **`ldp:MemberSubject`** says that the *member-derived-URI* is the URI assigned to the newly created LDPR.

5.4.2 HTTP POST

5.4.2.1 When a successful HTTP **`POST`** request to a LDPC results in the creation of a LDPR, the LDPC MUST update its membership triples to reflect that addition, and the resulting membership triple MUST be consistent with any LDP-defined predicates it exposes. A LDP Direct Container's membership triples MAY also be modified via through other means.

5.4.3 HTTP DELETE

5.4.3.1 When a LDPR identified by the object of a membership triple which was originally created by the LDP-DC is deleted, the LDPC server MUST also remove the corresponding membership triple.

5.5 Indirect

The following section contains normative clauses for Linked Data Platform Indirect Container.

5.5.1 General

5.5.1.1 Each LDP Indirect Container MUST also be a conforming LDP Direct Container as described in section 5.4 Direct, along with the following restrictions. LDP clients MAY infer the following triple: one whose subject is LDP Indirect Container, whose predicate is **`rdf:type`**, and whose object is **`ldp:Container`**, but there is no requirement to materialize this triple in the LDP-IC representation.

5.5.1.2 LDP Indirect Containers MUST contain exactly one triple whose subject is the LDPC URI, whose predicate is **`ldp:insertedContentRelation`**, and whose object *ICR* describes how the *member-derived-URI* in the container's membership triples is chosen. The *member-derived-URI* is taken from some triple *(S, P, O)* in the document supplied by the client as input to the create request; if *ICR*'s value is *P*, then the *member-derived-URI* is *O*. LDP does not define the behavior when more than one triple containing the predicate *P* is present in the client's input. For example, if the client POSTs RDF content to a container that causes the container to create a new LDP-RS, and that content contains the triple *(<> , foaf:primaryTopic , bob#me)* **`foaf:primaryTopic`** says that the *member-derived-URI* is *bob#me*. One consequence of this definition is that indirect container member creation is only well-defined by LDP when the document supplied by the client as input to the create request has an RDF media type.

5.5.2 HTTP POST

5.5.2.1 LDPCs whose **`ldp:insertedContentRelation`** triple has an object **other than** **`ldp:MemberSubject`** and that create new resources MUST add a triple to the container whose subject is the container's URI, whose predicate is **`ldp:contains`**, and whose object is the newly created resource's URI (which will be different from the *member-derived URI* in this case). This **`ldp:contains`** triple can be the only link from the container to the newly created resource in certain cases.

6. Notable information from normative references

This section is non-normative.

While readers, and especially implementers, of LDP are assumed to understand the information in its normative references, the working group has found that certain points are particularly important to understand. For those thoroughly familiar with the referenced specifications, these points might seem obvious, yet experience has shown that few non-experts find all of them obvious. This section enumerates these topics; it is simply re-stating (non-normatively) information locatable via the normative references.

6.1 Architecture of the World Wide Web

This section is non-normative.

Reference: [WEBARCH]

6.1.1 LDPC membership is not exclusive; this means that the same resource (LDPR or not) can be a member of more than one LDPC.

6.1.2 LDP servers should not re-use URIs, regardless of the mechanism by which members are created (`POST`, `PUT`, etc.). Certain specific cases exist where a LDPC server might delete a resource and then later re-use the URI when it identifies the same resource, but only when consistent with Web architecture. While it is difficult to provide absolute implementation guarantees of non-reuse in all failure scenarios, re-using URIs creates ambiguities for clients that are best avoided.

6.2 HTTP 1.1

This section is non-normative.

Reference: [RFC7230], [RFC7231], [RFC7232]

6.2.1 LDP servers can support representations beyond those necessary to conform to this specification. These could be other RDF formats, like N3 or NTriples, but non-RDF formats like HTML [HTML401] and JSON [RFC4627] would likely be common. HTTP content negotiation ([RFC7231] Section 3.4 - Content Negotiation) is used to select the format.

6.2.2 LDPRs can be created, updated and deleted using methods not defined in this document, for example through application-specific means, SPARQL UPDATE, etc. [SPARQL-UPDATE], as long as those methods do not conflict with this specification's normative requirements.

6.2.3 LDP servers remove the resource identified by the `Request-URI` in response to a successful HTTP `DELETE` request. After such a request, a subsequent HTTP `GET` on the same `Request-URI` usually results in a 404 (Not found) or 410 (Gone) status code, although HTTP allows others.

6.2.4 LDP servers can alter the state of other resources as a result of any HTTP request, especially when non-safe methods are used ([RFC7231] section 4.2.1). For example, it is acceptable for the server to remove triples from other resources whose subject or object is the deleted resource as the result of a successful HTTP `DELETE` request. It is also acceptable and common for LDP servers to not do this – the server's behavior can vary, so LDP clients cannot depend on it.

6.2.5 LDP servers can implement HTTP `PATCH` to allow modifications, especially partial replacement, of their resources. No minimal set of patch document formats is mandated by this document or by the definition of `PATCH` [RFC5789].

6.2.6 When the `Content-Type` request header is absent from a request, LDP servers might infer the content type by inspecting the entity body contents ([RFC7231] section 3.1.1.5).

6.3 RDF

This section is non-normative.

Reference: [rdf11-concepts]

6.3.1 The state of a LDPR can have triples with any subject(s). The URL used to retrieve the representation of a LDPR need not be the subject of any of its triples.

6.3.2 The representation of a LDPC can include an arbitrary number of additional triples whose subjects are the members of the container, or that are from the representations of the members (if they have RDF representations). This allows an LDP server to provide clients with information about the members without the client having to do a `GET` on each member individually.

6.3.3 The state of a LDPR can have more than one triple with an `rdf:type` predicate.

7. HTTP Header Definitions

7.1 The Accept-Post Response Header

NOTE

The LDP Working Group proposes incorporation of the features described in this section.

The `Accept-Post` header has applicability beyond LDP as outlined in this IETF draft [Accept-Post].

This specification introduces a new HTTP response header `Accept-Post` used to specify the document formats accepted by the server on HTTP `POST` requests. It is modelled after the `Accept-Patch` header defined in [RFC5789].

7.1.1 The syntax for `Accept-Post`, using the ABNF syntax defined in Section 1.2 of [RFC7231], is:

```
Accept-Post = "Accept-Post" ":" # media-range
```

The `Accept-Post` header specifies a comma-separated list of media ranges (with optional parameters) as defined by [RFC7231], Section 5.3.2. The `Accept-Post` header, in effect, uses the same syntax as the HTTP `Accept` header minus the optional `accept-`

`params` BNF production, since the latter does not apply to `Accept-Post`.

7.1.2 The `Accept-Post` HTTP header SHOULD appear in the `OPTIONS` response for any resource that supports the use of the `POST` method. The presence of the `Accept-Post` header in response to any method is an implicit indication that `POST` is allowed on the resource identified by the `Request-URI`. The presence of a specific document format in this header indicates that that specific format is allowed on `POST` requests to the resource identified by the `Request-URI`.

7.1.3 IANA Registration Template

The Accept-Post response header must be added to the permanent registry (see [RFC3864]).

Header field name: Accept-Post

Applicable Protocol: HTTP

Author/Change controller: W3C

Specification document: this specification

7.2 Preferences on the Prefer Request Header

7.2.1 Summary

This specification introduces new parameters on the HTTP `Prefer` request header's `return=representation` preference [RFC7240], used optionally by clients to supply a hint to help the server form a response that is most appropriate to the client's needs. The LDP-defined parameters suggest the portion(s) of a resource's state that the client application is interested in and, if received, is likely to be processed. LDP Containers with large numbers of associated documents and/or members will have large representations, and many client applications may be interested in processing only a subset of the LDPC's information (for example, only membership triples or only containment triples), resulting in a potentially large savings in server, client, and network processing.

Non-normative note: LDP server implementers should carefully consider the effects of these preferences on caching, as described in section 2 of [RFC7240].

Non-normative note: [RFC7240] recommends that server implementers include a `Preference-Applied` response header when the client cannot otherwise determine the server's behavior with respect to honoring hints from the response content. Examples illustrate some cases where the header is unnecessary.

7.2.2 Specification

7.2.2.1 The `include` hint defines a subset of a LDPR's content that a client would like included in a representation. The syntax for the `include` parameter of the HTTP `Prefer` request header's `return=representation` preference [RFC7240] is:

```
include-parameter = "include" *WSP "=" *WSP ldp-uri-list
```

Where `WSP` is whitespace [RFC5234], and `ldp-uri-list` is a double-quoted blank-delimited unordered set of URIs whose ABNF is given below. The generic preference BNF [RFC7240] allows either a quoted string or a token as the value of a preference parameter; LDP assigns a meaning to the value only when it is a quoted string of the form:

```
ldp-uri-list = DQUOTE *WSP URI *[ 1*WSP URI ] *WSP DQUOTE
```

where `DQUOTE` is a double quote [RFC5234], and `URI` is an absolute URI with an optional fragment component [RFC3986].

7.2.2.2 The `omit` hint defines a subset of a LDPR's content that a client would like omitted from a representation. The syntax for the `omit` parameter of the HTTP `Prefer` request header's `return=representation` preference [RFC7240] is:

```
omit-parameter = "omit" *WSP "=" *WSP ldp-uri-list
```

Where `WSP` and `ldp-uri-list` are defined as above for include.

7.2.2.3 When LDP servers receive a request with conflicting hints, this specification imposes no requirements on their behavior. They are free to reject the request, process it applying some subset of the hints, or anything else appropriate to the server. [RFC7240] suggests treating similar requests as though none of the conflicting preferences were specified.

7.2.2.4 This specification defines the following URIs for clients to use with `include` and `omit` parameters. It assigns no meaning to other URIs, although other specifications MAY do so.

Containment triples	`http://www.w3.org/ns/ldp#PreferContainment`
Membership triples	`http://www.w3.org/ns/ldp#PreferMembership`
Minimal-container triples	`http://www.w3.org/ns/ldp#PreferMinimalContainer` or the equivalent but deprecated term `http://www.w3.org/ns/ldp#PreferEmptyContainer`

Non-normative note: all currently defined URIs are only coherent for LDP-RSs, and in fact only for LDPCs, however in the future it is possible that additional URIs with other scopes of applicability could be defined.

7.2.3 Examples

This section is non-normative.

If we assume a container like the one below:

EXAMPLE 19

```
# The following is the representation of
#   http://example.org/netWorth/nw1/assets/

# @base <http://example.org/netWorth/nw1/assets/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<>
   a ldp:DirectContainer;
   dcterms:title "The assets of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:asset;
   ldp:insertedContentRelation ldp:MemberSubject.

<http://example.org/netWorth/nw1/>
   a o:NetWorth;
   o:asset <a1>, <a3>, <a2>.

<a1>
   a o:Stock;
   o:marketValue 100.00 .
<a2>
   a o:Cash;
   o:marketValue 50.00 .
<a3>
   a o:RealEstateHolding;
   o:marketValue 300000 .
```

Clients interested only in information about the container (for example, which membership predicate it uses) might use this hint on a `GET` request: `Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMinimalContainer"`

A server that honors this hint would return a following response containing the HTTP header `Preference-Applied: return=representation` and this representation:

Request to `http://example.org/netWorth/nw1/assets/`:

EXAMPLE 20

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

Response:

EXAMPLE 21

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/assets/>
   a ldp:DirectContainer;
   dcterms:title "The assets of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:asset;
   ldp:insertedContentRelation ldp:MemberSubject.
```

Clients interested only in information about the container (same as before) might use this hint instead: `Prefer: return=representation; omit="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferContainment"`. Note: **Treating the two as equivalent is not recommended.** While today this `omit` parameter value is equivalent to the preceding `include` parameter value, they may not be equivalent in the future due to the definition of minimal-container triples. Clients should preferentially use the `include` parameter, as it more precisely communicates their needs.

A **LDP 1.0** server that honors this hint would return the following response. Servers implementing later versions of LDP might return substantively different responses.

Request to `http://example.org/netWorth/nw1/assets/`:

EXAMPLE 22

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; omit="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferContainment"
```

Response:

EXAMPLE 23

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/assets/>
   a ldp:DirectContainer;
   dcterms:title "The assets of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:asset;
   ldp:insertedContentRelation ldp:MemberSubject.
```

Clients interested only in information about the container (for example, which membership predicate it uses) and its membership might use this hint on a `GET` request: `Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferMinimalContainer"`.

A server that honors this hint would return (at least) the following response, and perhaps only this (it might well omit containment triples if they are not specifically requested). In cases like this example, where a client can detect from the content that its hints were honored (the presence of the predicates `dcterms:title` and `o:asset` demonstrate this in the representation below), there is no need for the server to include a `Preference-Applied` response header in many common cases like a `200 (OK)` response. In other cases, like status code `303`, the header would still be required for the client to know that the `303` response entity is a representation of the resource identified by the `Location` URI instead of a short hypertext note (one with a hyperlink to the same URI reference provided in the `Location` header field [RFC7231]).

Request to `http://example.org/netWorth/nw1/assets/`:

EXAMPLE 24

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferMinimalContaine
```

Response:

EXAMPLE 25

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/assets/>
   a ldp:DirectContainer;
   dcterms:title "The assets of JohnZSmith";
   ldp:membershipResource <http://example.org/netWorth/nw1/>;
   ldp:hasMemberRelation o:asset;
   ldp:insertedContentRelation ldp:MemberSubject.

<http://example.org/netWorth/nw1/>
   a o:NetWorth;
   o:asset <a1>, <a3>, <a2>.
```

8. Link Relations

The intent is that these link relations will be registered with IANA per [RFC5988] section 6.2.1.

8.1 describedby

The contents of this section were originally taken from [POWDER] appendix D, and then modified to comply with the current registration template. The pre-LDP IANA link relation registry entry for `describedby` refers to a different section of [POWDER] that was substantively updated in an erratum, and that section was not actually the normative definition of the link relation. Since we expect no update to [POWDER] that incorporates the erratum or fixes the registry link, this superseding registration approach is being taken.

The following Link Relationship will be submitted to IANA for review, approval, and inclusion in the IANA Link Relations registry.

Relation Name:
: `describedby`

Description:
: The relationship `A describedby B` asserts that resource B provides a description of resource A. There are no constraints on the format or representation of either A or B, neither are there any further constraints on either resource.

Reference:
: The W3C Linked Data Platform specification.

Notes:
: Descriptions of resources may be socially sensitive, may require processing to be understood and may or may not not be accurate. Consumers of descriptive resources should be aware of the source and chain of custody of the data. Security considerations for URIs (Section 7 of RFC 3986) and IRIs (Section 8 of RFC 3987) apply to the extent that describing resources may affect consumers' decisions about how or whether to retrieve those resources.

9. Security Considerations

This section is non-normative.

As with any protocol that is implemented leveraging HTTP, implementations should take advantage of the many security-related facilities associated with it and are not required to carry out LDP operations that may be in contradistinction to a particular security policy in place. For example, when faced with an unauthenticated request to replace system critical RDF statements in a graph through the PUT method, applications may consider responding with the 401 status code (Unauthorized), indicating that the appropriate authorization is required. In cases where the provided authentication fails to meet the requirements of a particular access control policy, the 403 status code (Forbidden) can be sent back to the client to indicate this failure to meet the access control policy.

A. Acknowledgements

This section is non-normative.

The following people have been instrumental in providing thoughts, feedback, reviews, content, criticism and input in the creation of this specification:

> Arnaud Le Hors (chair), Alexandre Bertails, Andrei Sambra, Andy Seaborne, Antonis Loizou, Ashok Malhotra, Bart van Leeuwen, Cody Burleson, David Wood, Eric Prud'hommeaux, Erik Wilde, Henry Story, John Arwe, Kevin Page, Kingsley Idehen, Mark Baker, Martin P. Nally, Miel Vander Sande, Miguel Esteban Gutiérrez, Nandana Mihindukulasooriya, Olivier Berger, Pierre-Antoine Champin, Raúl García Castro, Reza B'Far, Richard Cyganiak, Rob Sanderson, Roger Menday, Ruben Verborgh, Sandro Hawke, Serena Villata, Sergio Fernandez, Steve Battle, Steve Speicher, Ted Thibodeau, Tim Berners-Lee, Yves Lafon

B. Change History

This section is non-normative.

The change history is up to the editors to insert a brief summary of changes, ordered by most recent changes first and with heading from which public draft it has been changed from.

Summary

Summary of notable changes from the Proposed Recommendation.

- Minor rewording about provided authentication in Security Considerations
- Added sentence before example 3 in the introduction to Linked Data Platform Containers

C. References

C.1 Normative references

[DC-TERMS]
: Dublin Core Metadata Initiative. *Dublin Core Metadata Initiative Terms, version 1.1.* 11 October 2010. DCMI Recommendation. URL: http://dublincore.org/documents/2010/10/11/dcmi-terms/.

[JSON-LD]
: Manu Sporny; Gregg Kellogg; Markus Lanthaler. *JSON-LD 1.0*. 16 January 2014. W3C Recommendation. URL: http://www.w3.org/TR/json-ld/

[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

[RFC3864]
: G. Klyne; M. Nottingham; J. Mogul. *Registration Procedures for Message Header Fields*. September 2004. Best Current Practice. URL: https://tools.ietf.org/html/rfc3864

[RFC3986]
: T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. January 2005. Internet Standard. URL: https://tools.ietf.org/html/rfc3986

[RFC3987]
: M. Duerst; M. Suignard. *Internationalized Resource Identifiers (IRIs)*. January 2005. Proposed Standard. URL:

https://tools.ietf.org/html/rfc3987

[RFC5023]
J. Gregorio, Ed.; B. de hOra, Ed.. *The Atom Publishing Protocol*. October 2007. Proposed Standard. URL: https://tools.ietf.org/html/rfc5023

[RFC5234]
D. Crocker, Ed.; P. Overell. *Augmented BNF for Syntax Specifications: ABNF*. January 2008. Internet Standard. URL: https://tools.ietf.org/html/rfc5234

[RFC5789]
L. Dusseault; J. Snell. *PATCH Method for HTTP*. March 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5789

[RFC5988]
M. Nottingham. *Web Linking*. October 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5988

[RFC6585]
M. Nottingham; R. Fielding. *Additional HTTP Status Codes*. April 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6585

[RFC7230]
R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7230

[RFC7231]
R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7231

[RFC7232]
R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7232

[RFC7240]
J. Snell. *Prefer Header for HTTP*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7240

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

[rdf-schema]
Dan Brickley; Ramanathan Guha. *RDF Schema 1.1*. 25 February 2014. W3C Recommendation. URL: http://www.w3.org/TR/rdf-schema/

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

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

C.2 Informative references

[Accept-Post]
J. Arwe; S. Speicher; E. Wilde. *The Accept-Post HTTP Header*. Internet Draft. URL: http://tools.ietf.org/html/draft-wilde-accept-post

[HTML401]
Dave Raggett; Arnaud Le Hors; Ian Jacobs. *HTML 4.01 Specification*. 24 December 1999. W3C Recommendation. URL: http://www.w3.org/TR/html401

[LDP-PAGING]
S. Speicher; J. Arwe; A. Malhotra. *Linked Data Platform Paging*. Candidate Recommendation. URL: http://www.w3.org/TR/ldp-paging/

[LDP-Tests]
R. Garcia-Castro; F. Serena. *Linked Data Platform 1.0 Test Cases*. Editor's Draft of Working Group Note. URL: https://dvcs.w3.org/hg/ldpwg/raw-file/default/tests/ldp-testsuite.html

[LDP-UCR]
Steve Battle; Steve Speicher. *Linked Data Platform Use Cases and Requirements*. 13 March 2014. W3C Note. URL: http://www.w3.org/TR/ldp-ucr/

[LINKED-DATA]
Tim Berners-Lee. *Linked Data Design Issues*. 27 July 2006. W3C-Internal Document. URL: http://www.w3.org/DesignIssues/LinkedData.html

[POWDER]
Phil Archer; Kevin Smith; Andrea Perego. *Protocol for Web Description Resources (POWDER): Description Resources*. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-powder-dr-20090901/

[RFC4627]
D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. July 2006. Informational. URL: https://tools.ietf.org/html/rfc4627

[SPARQL-UPDATE]
Paul Gearon; Alexandre Passant; Axel Polleres. *SPARQL 1.1 Update*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-update/

[sparql11-query]
Steven Harris; Andy Seaborne. *SPARQL 1.1 Query Language*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-query/