Linked Data Patch Format (W3C WG Note)

Alexandre Bertails, Pierre-Antoine Champin, Andrei Sambra

Linked Data Patch Format

W3C Working Group Note 28 July 2015

This version:
http://www.w3.org/TR/2015/NOTE-ldpatch-20150728/
Latest published version:
http://www.w3.org/TR/ldpatch/
Latest editor's draft:
https://dvcs.w3.org/hg/ldpwg/raw-file/ldpatch/ldpatch.html
Test suite:
https://github.com/pchampin/ld-patch-testsuite
Implementation report:
https://dvcs.w3.org/hg/ldpwg/raw-file/tip/tests/ldpatch/reports/ldpatch.html
Previous version:
http://www.w3.org/TR/2015/CR-ldpatch-20150303/
Editors:
Alexandre Bertails, alexandre@bertails.org
Pierre-Antoine Champin, Université de Lyon, pchampin@liris.cnrs.fr
Andrei Sambra, MIT/W3C, andrei@w3.org

Abstract

Linked Data Patch Format (LD Patch) defines a language for expressing a sequence of operations for patching Linked Data resources; it is suitable for use with the HTTP PATCH method.

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/.

Although the Linked Data Platform (LDP) Working Group is currently favoring LD Patch, it seeks more input in deciding which format to promote for use in LDP PATCH [LDP] operations on LDP RDF Sources. Other viable candidates include:

- SPARQL 1.1 Update — already standardized, but quite complex for LDP scenarios
- SparqlPatch — restricted to a simple subset of SPARQL 1.1 Update
- TurtlePatch — uses an even simpler subset, but requires unusual handling of blank nodes
- RDF Patch — simple, but also requires unusual handling of blank nodes

At this point, the advantage leans towards LD Patch in terms of simplicity, ease of implementation, and run-time performance on anticipated data. We welcome data relevant to this decision.

This specification was previously published as a Candidate Recommendation (CR). Due to lack of sufficient implementations to meet the CR exit criteria within the time remaining under the current charter, the Working Group decided to take it off the W3C Recommendation track and publish it as a W3C Note for future reference. This document may be reused in part or in whole by another WG in the future, or not.

This document was published by the Linked Data Platform Working Group as a Working Group Note. If you wish to make comments regarding this document, please send them to public-ldp-comments@w3.org (subscribe, archives). All comments are welcome.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

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 1 August 2014 W3C Process Document.

Table of Contents

1. Introduction

This section is non-normative.

Linked Data “describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP, RDF and IRIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried.” (source Wikipedia).

This document defines the Linked Data Patch Format (LD Patch), a format for describing changes to apply to Linked Data. It is suitable for use with HTTP PATCH [RFC5789], a method to perform partial modifications to Web resources.

An instance of the LD Patch language (or LD Patch document) defines a list of operations to be performed against a Linked Data resource, namely the addition or removal of RDF [rdf11-

concepts] triples in the graph representing this resource.

The LD Patch format described in this document should be seen as a language for updating RDF Graphs in a resource-centric fashion. It is the intention to confine its expressive power to an RDF diff with partial support for blank nodes and `rdf:List` manipulations. For more powerful operations on RDF Graphs and Quad Stores, the LDP WG recommends the reader to consider SPARQL Update [sparql11-update].

2. Examples

This section is non-normative.

2.1 Full example

The following RDF Graph describes the relation between a person named Tim Berners-Lee (denoted by `<http://example.org/timbl#>`) and two events he attended.

EXAMPLE 1

```
@prefix schema: <http://schema.org/> .
@prefix profile: <http://ogp.me/ns/profile#> .
@prefix ex: <http://example.org/vocab#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<#> a schema:Person ;
  schema:alternateName "TimBL" ;
  profile:first_name "Tim" ;
  profile:last_name "Berners-Lee" ;
  schema:workLocation [ schema:name "W3C/MIT" ] ;
  schema:performerIn _:b1, _:b2 ;
  ex:preferredLanguages ( "en" "fr" ).

_:b1 schema:name "F2F5 - Linked Data Platform" ;
  schema:url <https://www.w3.org/2012/ldp/wiki/F2F5> .

_:b2 a schema:Event ;
  schema:name "TED 2009" ;
  schema:startDate "2009-02-04" ;
  schema:url <http://conferences.ted.com/TED2009/> .
```

The following is an example HTTP Patch request, conveying an LD Patch document:

EXAMPLE 2

```
PATCH /timbl HTTP/1.1
Host: example.org
Content-Length: 478
Content-Type: text/ldpatch
If-Match: "abc123"

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix profile: <http://ogp.me/ns/profile#> .
@prefix ex: <http://example.org/vocab#> .

Delete { <#> profile:first_name "Tim" } .
Add {
  <#> profile:first_name "Timothy" ;
    profile:image <https://example.org/timbl.jpg> .
} .

Bind ?workLocation <#> / schema:workLocation .
Cut ?workLocation .

UpdateList <#> ex:preferredLanguages 1..2 ( "fr-CH" ) .

Bind ?event <#> / schema:performerIn [ / schema:url = <https://www.w3.org/2012/ldp/wiki/F2F5> ]  .
Add { ?event rdf:type schema:Event } .

Bind ?ted <http://conferences.ted.com/TED2009/> / ^schema:url ! .
Delete { ?ted schema:startDate "2009-02-04" } .
Add {
  ?ted schema:location [
    schema:name "Long Beach, California" ;
```

```
      schema:geo [
        schema:latitude "33.7817" ;
        schema:longitude "-118.2054"
      ]
    ]
  } .
```

This example introduces most features of the LD Patch format: `@prefix` and prefixed names, the Add, Delete, Cut, and UpdateList operations, the node Binding mechanism, and blank node creation. The "text/ldpatch" media type is prospectively used to identify such LD Patch documents.

The following is the resulting (patched) document.

EXAMPLE 3

```
@prefix schema: <http://schema.org/> .
@prefix profile: <http://ogp.me/ns/profile#> .
@prefix ex: <http://example.org/vocab#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<#> a schema:Person ;
  schema:alternateName "TimBL" ;
  profile:first_name "Timothy" ;
  profile:last_name "Berners-Lee" ;
  profile:image <https://example.org/timbl.jpg> ;
  schema:performerIn _:b1, _:b2 ;
  ex:preferredLanguages ( "en" "fr-CH" ) .

_:b1 a schema:Event ;
  schema:name "F2F5 - Linked Data Platform" ;
  schema:url <https://www.w3.org/2012/ldp/wiki/F2F5> .

_:b2 a schema:Event ;
  schema:name "TED 2009" ;
  schema:url <http://conferences.ted.com/TED2009/> ;
  schema:location [
    schema:name "Long Beach, California";
    schema:geo [ schema:latitude "33.7817" ; schema:longitude "-118.2054" ]
  ] .
```

2.2 `rdf:List` manipulation examples

All the LD Patch examples in this section are applied against the following RDF graph (target IRI `http://example.org/timbl`):

EXAMPLE 4

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "ipsum" "dolor" "sit" "amet" ) .
```

How to replace elements

This example shows how to replace one element (here the second one) with a new one:

EXAMPLE 5

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> 1..2 ( "fr" ) .
```

Output graph:

EXAMPLE 6

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "fr" "dolor" "sit" "amet" ) .
```

How to insert new elements

This example shows how to insert new elements at a specific index (here `2`):

EXAMPLE 7

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> 2..2 ( "en" "fr" ) .
```

Output graph:

EXAMPLE 8

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "ipsum" "en" "fr" "dolor" "sit" "amet" ) .
```

How to append elements

This example shows how to append elements at the end of a collection:

EXAMPLE 9

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> .. ( "en" "fr" ) .
```

Output graph:

EXAMPLE 10

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "ipsum" "dolor" "sit" "amet" "en" "fr" ) .
```

How to replace all the elements after a given index

This example shows how to replace all the elements after the index 2 with the provided collection:

EXAMPLE 11

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> 2.. ( "en" "fr" ) .
```

Output graph:

EXAMPLE 12

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "ipsum" "en" "fr" ) .
```

How to replace the n last elements

This example shows how to replace the last 3 elements of the provided collection:

EXAMPLE 13

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> -3.. ( "en" "fr" ) .
```

Output graph:

EXAMPLE 14

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "ipsum" "en" "fr" ) .
```

How to remove elements

This example shows how to remove elements (here the second and the third) from a collection:

EXAMPLE 15

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> 1..3 ( ) .
```

Output graph:

EXAMPLE 16

```
<#> <http://example.org/vocab#preferredLanguages> ( "lorem" "sit" "amet" ) .
```

How to empty a collection

Finally, this example shows how to empty a collection:

EXAMPLE 17

```
UpdateList <#> <http://example.org/vocab#preferredLanguages> 0.. ( ) .
```

Output graph:

EXAMPLE 18

```
<#> <http://example.org/vocab#preferredLanguages> ( ) .
```

3. Conformance

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

The key words MUST and MUST NOT are to be interpreted as described in [RFC2119].

This specification defines conformance criteria for:

- LD Patch documents
- LD Patch parsers
- LD Patch processors
- LD Patch servers

A conforming **LD Patch document** is a Unicode string that conforms to the grammar defined in the Concrete Syntax section.

A conforming **LD Patch parser** is a system capable of parsing LD Patch documents. The resulting abstract concept is called a **Linked Data patch**, or simply **patch** when the context is unambiguous. Parsers should treat Literals as being composed of a lexical form and an optional language tag [BCP47] (as used by Turtle [Turtle]) or datatype IRI.

A conforming **LD Patch processor** is a system capable of executing a Linked Data patch against an RDF Graph and whose semantics follow the ones defined in the LD Patch Semantics section. It would either return a new graph or update the input graph in place.

A conforming **LD Patch server** is a system capable of processing an LD Patch document through an HTTP PATCH request as defined in LDP PATCH [LDP]. It MUST handle errors as defined in the Error Handling section.

The IRI that identifies the LD Patch format is: `http://www.w3.org/ns/formats/LD_Patch`.

4. LD Patch Semantics

An LD Patch document is applied to a Linked Data resource identified by an IRI (the ***target IRI***) and represented by an RDF graph (the ***target graph***). It is made of a prologue followed by a list of statements. The prologue declares a number of prefixes used to abbreviate IRIs as PrefixedNames. Then, each statement either binds a variable to a matching node from the target graph, or specifies a modification on the target graph.

4.1 *Nodes and triples Semantics*

LD Patch borrows much of its syntax and semantics from Turtle [Turtle] for describing nodes and triples. Especially, whenever production rules triples or collection are used, Turtle semantics must be applied to parse them as a set of triples that we call an ***argument graph***.

There are however a few points that need to be highlighted in the way LD Patch parses an argument graph compared to Turtle:

- The base IRI used to resolve relative IRIs is the target IRI.
- LD Patch allows variables in subject and object positions.
- The value of a variables is the last node to which it was bound (in case it appears in several Bind statements).
- ***The scope of blank node identifiers is the whole LD Patch document.*** That means that argument graphs across statements can share blank nodes.

As IRIs and RDF Literals have global scopes, such nodes in an argument graph represent the same resource as in the target graph. Blank nodes, on the other hand, pose a problem, as they have no global identifiers. Indeed, since the scope of blank node identifiers is limited to the LD Patch document in which they appear, any blank node identifier appearing in an LD Patch document is understood to denote a *fresh* blank node, distinct from any node initially present in the target graph. Therefore blank node identifiers in LD Patch cannot interfere with pre-existing blank nodes in the target graph.

However, LD Patch provides mechanisms to address those pre-existing blank nodes: binding a variable to a blank node reachable through a path expression, cutting a whole tree made of blank nodes, or using UpdateList to deal with those blank nodes that constitute RDF collections. There are cases where those mechanisms will not be able to unambiguously address a given blank node, but those cases are deemed pathological, and are out of the scope of this specification.

4.2 *Path Expression*

A Path expression can be used to locate RDF nodes within the target graph. A path expression consists of a series of one or more Steps (introduced by a "`/`") or Constraints, which are applied in order from left to right. The main goal is to allow addressing a blank node by “walking” the arcs of the graph from an previously identified node.

`/` behaves like a left-associated operator where the left operand is a node set, the right operand is a Step, and the result is a node set. A Constraint behaves like a predicate function whose implicit parameter is the node set on which it is applied. In the context of a Filter, this implicit node set becomes the left operand for `/`.

A ***Step*** can be of three kinds:

- A ***StepForward*** is defined by an IRI, and consists in following the corresponding outgoing arcs in the target graph.
- A ***StepBackward*** is defined by an IRI preceded by the caret ("`^`") sign, and consists in following the corresponding incoming arcs *in reverse* in the target graph.
- A ***StepAt*** is defined by an integer *n*, and consists in following *n* `rdf:rest` arcs and one `rdf:first` arc in order to reach the corresponding member of an RDF collection. It is equivalent to a sequence of *n*+1 StepForwards with the corresponding IRIs. A negative index *n* denotes the n-th element from the end of the list counting backwards.

A ***Constraint*** can be of two kinds:

- A ***Unicity constraint***, described by the *bang* ("`!`") character, checks that the current node set contains exactly one node.
- A ***Filter***, consisting of a Path expression between square brackets ("`[`", "`]`"), keeps only the nodes that “satisfy” the enclosed path, i.e. those from which the enclosed path reaches at least one node.
- Additionally, the path in a filter can specify an equality constraint with the use of the equal ("`=`") sign and a Value. In that case, only the nodes for which that particular value is reached through the enclosed path are kept.

The following path expression (taken from the Examples section) will look for all events matching the predicate `schema:performerIn`, keeping only the one matching the IRI `<https://www.w3.org/2012/ldp/wiki/F2F5>`.

EXAMPLE 19

```
/ schema:performerIn [ / schema:url = <https://www.w3.org/2012/ldp/wiki/F2F5> ]
```

4.3 Patch Operations

4.3.1 *Bind*

The Bind operation is used to bind an RDF Term to a variable. The process results in the variable being bound to exactly one node. After being bound, the variable can be used in the subsequent statements. Another Bind can override the value of a previously bound variable.

The Bind operation is defined by three components: Var, Value and Path, the last component being optional (can be considered equivalent to the empty path).

Var contains a unique name for the new variable. Variables are prefixed by the "`?`" character, which is not part of the variable name.

Value is the RDF Term that will be used as starting point when following the path expression.

Path is the expression that is used to identify the RDF Term to which the Variable will be bound. It is comprised of Step(s) and/or Constraint(s).

Following the example above, the Bind operation creates a new variable called `event`, starting from the RDF Term `<#>` and following the path expression `/ schema:performerIn [ / schema:url = <https://www.w3.org/2012/ldp/wiki/F2F5> ]` in order to identify the RDF Term to which this variable will be bound to – i.e. `_:b2` in the target graph.

EXAMPLE 20

```
Bind ?event <#> / schema:performerIn [ / schema:url = <https://www.w3.org/2012/ldp/wiki/F2F5> ] .
```

4.3.2 *Add*

The Add operation is used to append new RDF triples to the target graph.

It has a single argument: an argument graph *g*. All triples in *g* must be added to the *target graph*. If an argument graph contains one or more triples that already exist in the target graph, the Add operation does not fail.

EXAMPLE 21

```
Add {
    <#> profile:first_name "Timothy" ;
        profile:image <https://example.org/timbl.jpg> .
} .

Add { ?event rdf:type schema:Event } .
```

4.3.3 *AddNew*

The AddNew operation is used to append new RDF triples to the target graph. It behaves like Add but unlike its counterpart, AddNew fails when trying to add an already existing triple.

4.3.4 *Delete*

The Delete operation is used to remove RDF triples from the target graph.

It has a single argument: an argument graph *g*. All triples in *g* must be removed from the *target graph*. It does not fail if one of those triples did not exist in the target graph. Blank nodes

identifiers are allowed in Delete statements but they remain scoped to the LD Patch document, so they can only match a blank node previously added by the same LD Patch document.

EXAMPLE 22

```
Delete { <#> profile:first_name "Tim" } .

Delete { ?ted schema:startDate "2009-02-04" } .
```

4.3.5 *DeleteExisting*

The DeleteExisting operation is used to remove RDF triples from the target graph. It behaves like Delete but unlike its counterpart, DeleteExisting fails when trying to delete a non-existing triple.

4.3.6 *Cut*

The Cut operation is used to remove one or more triples connected to a specific blank node *b*. More precisely, it removes all the outgoing arcs for *b* from the target graph, and does the same recursively for all objects of those triples being blank nodes. Finally, it removes all incoming arcs of *b*.

EXAMPLE 23

```
Cut ?workLocation .
```

4.3.7 *UpdateList*

The UpdateList operation is used to update some members of an RDF collection. It works in a similar way to slicing in Python or similar languages: it replaces a slice of a list by another list.

The UpdateList operation is defined by four components: a variable or IRI, a predicate, a Slice expression, and an argument graph containing an RDF collection.

The ***Slice expression*** is composed of two optional 0-based indexes i_{min} and i_{max} separated by "`..`". A negative index denotes elements from the end of the list counting backwards, e.g. the last element of any non-empty list always has the index `-1`. An omitted value is interpreted as the length of the collection. The Slice expression will denote the slice of the list being preceded by i_{min} elements, and spanning over (i_{max} - i_{min}) elements.

For example, here are some Slice expressions for the list `( "lorem" "ipsum" "dolor" "sit" "amet" )`:

- `2..4` denotes the slice `( "dolor" "sit" )`, i.e. the elements between the indexes `2` and `4`
- `0..` denotes the slice `( "lorem" "ipsum" "dolor" "sit" "amet" )`, i.e. the whole list
- `3..` denotes the slice `( "sit" "amet" )`, i.e. all the elements after the index `3`
- `-2..` denotes the slice `( "sit" "amet" )`, i.e. the last 2 elements
- `2..2` denotes the empty slice located between `"ipsum"` and `"dolor"`
- `..` denotes the empty slice located at the end of the list

Appendix A contains a detailed algorithm for implementing the UpdateList logic using reified `rdf:List`.

4.3.8 *Error Handling*

LD Patch abides to the semantics of the HTTP PATCH method [RFC5789], in that the server "MUST apply the entire set of changes atomically and never provide (e.g., in response to a GET during this operation) a partially modified representation. If the entire patch document cannot be successfully applied (e.g., one of the instructions has failed), then the server MUST NOT apply any of the changes". In the case LD Patch operations fail to be applied, Error Handling, Section 2 of [RFC5789] specifies the error codes to be used.

Here are some additional error conditions more specific to LD Patch:

- If a Bind statement fails to match exactly one node, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If a Unicity constraint is violated, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If a Cut operation fails to remove any triple, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If a Cut operation is called on a variable not bound to a blank node, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If a DeleteExisting attempts to remove a non-existing triple, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If a AddNew attempts to add an already existing triple, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If the subject and predicate provided to an UpdateList do not have a unique object, or if this object is not a well-formed collection, then a HTTP 422 (Unprocessable Entity) error status code MUST be returned.
- If the indexes in a slice expression are in the wrong order (e.g. `2868..42`), then the parsing fails and a 400 (Bad Request) error status code MUST be returned.
- If an index in a slice expression is greater than the length of the `rdf:List`, then a 422 (Unprocessable Entity) error status code MUST be returned.
- If a prefix name (PNAME_NS) is used without being previously declared, then the parsing fails and a 400 (Bad Request) error status code MUST be returned.
- If a variable is used without being previously bound, then the parsing fails and a 400 (Bad Request) error status code MUST be returned.

Note: 422 (Unprocessable Entity) is defined in 422 Unprocessable Entity, Section 11.2 of [RFC4918].

4.3.9 *Pathological Graph*

There exists a particular case which LD Patch is not able to address. Given an RDF graph *G*, a blank node *b* is said to be unambiguous in *G* if there exists a couple *(n, p)* where

- *n* is an IRI or a literal
- *p* is a Path Expression

such that applying *p* to {*n*} results in the singleton set {*b*}.

It is easy to see that only the unambiguous blank nodes of a graph can be handled in LD Patch.

Consider for example the following graph:

> EXAMPLE 24
>
> ```
> <#> foaf:name "Alice" ; foaf:knows _:b1, _:b2 .
> _:b1 a foaf:Person .
> _:b2 a foaf:Person ; schema:workLocation _:b3 .
> _:b3 schema:name "W3C/MIT" .
> ```

The blank nodes `_:b2` and `_:b3` are unambiguous as they can be reached unambiguoulsy from the literal `"W3C/MIT"`. The blank node `_:b1`, on the other hand, is ambigious as all path expressions that can match it would also match `_:b2`.

Another example is a graph containing only blank nodes. All its nodes are therefore ambiguous as they can not be reached from an IRI or a literal. Such a graph is not interesting in the context of Linked Data as it contains no IRI to link to or from it.

Therefore, ambiguous blank nodes are considered a pathological case in the context of Linked Data, and so the fact that they cannot be coped with in LD Patch is deemed acceptable. Furthermore, their presence in a graph does not prevent the other nodes of that graph to be handled by LD Patch. Most notably, all non-lean graphs [rdf11-mt] are also pathological.

5. LD Patch compared to Turtle and SPARQL

This section is non-normative.

The LD Patch syntax uses a Turtle [Turtle] style syntax for its triples production. This production differs from the Turtle language in that the subject and object production rules allow the use of variables.

LD Patch variables are restricted to the VAR1 production rule from SPARQL 1.1 [sparql11-query], only allowing a leading '?'.

Finally, the prefix directive is restricted to the prefixID production rule in Turtle [Turtle], only allowing @prefix.

6. Concrete Syntax

Production labels consisting of a number and a final 's', e.g. [135s], reference the production with that number in the SPARQL 1.1 Query Language grammar [sparql11-query]. Production labels consisting of a number and a final 't', e.g. [6t], reference the production with that number in the Turtle grammar [Turtle]. A production label containing an extra trailing '*' denotes a modified rule, e.g. [10t*] and [12t*].

[1]	ldpatch	::= prologue statement*
[2]	prologue	::= prefixID*
[3]	statement	::= bind | add | addNew | delete | deleteExisting | cut | updateList
[4]	bind	::= ("Bind" | "B") VAR1 value path? "."
[5]	add	::= ("Add" | "A") "{" graph "}" "."
[6]	addNew	::= ("AddNew" | "AN") "{" graph "}" "."
[7]	delete	::= ("Delete" | "D") "{" graph "}" "."
[8]	deleteExisting	::= ("DeleteExisting" | "DE") "{" graph "}" "."
[9]	cut	::= ("Cut" | "C") VAR1 "."
[10]	updateList	::= ("UpdateList" | "UL") varOrIRI predicate slice collection "."
[11]	varOrIRI	::= iri | VAR1
[12]	value	::= iri | literal | VAR1
[13]	path	::= ('/' step | constraint)*
[14]	step	::= '^' iri | iri | INDEX
[15]	constraint	::= '[' path ('=' value)? ']' | '!'
[16]	slice	::= INDEX? '..' INDEX?
[17]	INDEX	::= '-'? [0-9]+
[143s]	VAR1	::= '?' VARNAME
[166s]	VARNAME	::= (PN_CHARS_U | [0-9]) (PN_CHARS_U | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040])*
[4t]	prefixID	::= "@prefix" PNAME_NS IRIREF "."
[18]	graph	::= triples ('.' triples)* '.'?
[6t]	triples	::= subject predicateObjectList | blankNodePropertyList predicateObjectList?
[7t]	predicateObjectList	::= verb objectList (';' (verb objectList)?)*
[8t]	objectList	::= object (',' object)*
[9t]	verb	::= predicate | 'a'
[10t*]	subject	::= iri | BlankNode | collection | VAR1
[11t]	predicate	::= iri
[12t*]	object	::= iri | BlankNode | collection | blankNodePropertyList | literal | VAR1
[13t]	literal	::= RDFLiteral | NumericLiteral | BooleanLiteral
[14t]	blankNodePropertyList	::= '[' predicateObjectList ']'
[15t]	collection	::= '(' object* ')'
[16t]	NumericLiteral	::= INTEGER | DECIMAL | DOUBLE
[128s]	RDFLiteral	::= String (LANGTAG | '^^' iri)?

[133s]	BooleanLiteral	::=	'true' | 'false'
[17]	String	::=	STRING_LITERAL_QUOTE | STRING_LITERAL_SINGLE_QUOTE | STRING_LITERAL_LONG_SINGLE_QUOTE | STRING_LITERAL_LONG_QUOTE
[135s]	iri	::=	IRIREF | PrefixedName
[136s]	PrefixedName	::=	PNAME_LN | PNAME_NS
[137s]	BlankNode	::=	BLANK_NODE_LABEL | ANON
[18]	IRIREF	::=	'<' ([^#x00-#x20<>"{}|^`\] | UCHAR)* '>' /* #x00=NULL #01-#x1F=control codes #x20=space */
[139s]	PNAME_NS	::=	PN_PREFIX? ':'
[140s]	PNAME_LN	::=	PNAME_NS PN_LOCAL
[141s]	BLANK_NODE_LABEL	::=	'_:' (PN_CHARS_U | [0-9]) ((PN_CHARS | '.')* PN_CHARS)?
[144s]	LANGTAG	::=	'@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[19]	INTEGER	::=	[+-]? [0-9]+
[20]	DECIMAL	::=	[+-]? [0-9]* '.' [0-9]+
[21]	DOUBLE	::=	[+-]? ([0-9]+ '.' [0-9]* EXPONENT | '.' [0-9]+ EXPONENT | [0-9]+ EXPONENT)
[154s]	EXPONENT	::=	[eE] [+-]? [0-9]+
[22]	STRING_LITERAL_QUOTE	::=	'"' ([^#x22#x5C#xA#xD] | ECHAR | UCHAR)* '"' /* #x22=" #x5C=\ #xA=new line #xD=carriage return */
[23]	STRING_LITERAL_SINGLE_QUOTE	::=	"'" ([^#x27#x5C#xA#xD] | ECHAR | UCHAR)* "'" /* #x27=' #x5C=\ #xA=new line #xD=carriage return */
[24]	STRING_LITERAL_LONG_SINGLE_QUOTE	::=	"'''" (("'" | "''")? ([^'\] | ECHAR | UCHAR))* "'''"
[25]	STRING_LITERAL_LONG_QUOTE	::=	'"""' (('"' | '""')? ([^"\] | ECHAR | UCHAR))* '"""'
[26]	UCHAR	::=	'\\u' HEX HEX HEX HEX | '\\U' HEX HEX HEX HEX HEX HEX HEX HEX
[159s]	ECHAR	::=	'\' [tbnrf"'\]
[161s]	WS	::=	#x20 | #x9 | #xD | #xA
[162s]	ANON	::=	'[' WS* ']'
[163s]	PN_CHARS_BASE	::=	[A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x02FF] | [#x0370-#x037D] | [#x037F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
[164s]	PN_CHARS_U	::=	PN_CHARS_BASE | '_'
[166s]	PN_CHARS	::=	PN_CHARS_U | '-' | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040]
[167s]	PN_PREFIX	::=	PN_CHARS_BASE ((PN_CHARS | '.')* PN_CHARS)?
[168s]	PN_LOCAL	::=	(PN_CHARS_U | ':' | [0-9] | PLX) ((PN_CHARS | '.' | ':' | PLX)* (PN_CHARS | ':' | PLX))?
[169s]	PLX	::=	PERCENT | PN_LOCAL_ESC
[170s]	PERCENT	::=	'%' HEX HEX
[171s]	HEX	::=	[0-9] | [A-F] | [a-f]
[172s]	PN_LOCAL_ESC	::=	'\' ('_' | '~' | '.' | '-' | '!' | '$' | '&' | "'" | '(' | ')' | '*' | '+' | ',' | ';' | '=' | '/' | '?' | '#' | '@' | '%')

A. UpdateList Algorithm

This section is non-normative.

Below is an algorithm explaining how `UpdateList s p i_min..i_max collection` can be processed in the presence of a reified `rdf:List`, i.e. encoded with `rdf:first` and `rdf:rest`. Implementers may take advantage of a more native encoding for `rdf:List`.

- **Let** s_{pre} be s, p_{pre} be p and o_{pre} the object of the triple (s, p, ?) from the target graph.
- **Repeat** i_{min} times:
  - **Set** s_{pre} to o_{pre}, p_{pre} to `rdf:rest` and o_{pre} to the object of the triple (o_{pre}, `rdf:rest`, ?) from the target graph.
- **Let** s_{post} be s_{pre}, p_{post} be p_{pre} and o_{post} be o_{pre}.
- **Repeat** (i_{max}-i_{min}) times:
  - Remove from the target graph the arcs (s_{post}, p_{post}, o_{post}).
  - **Let** *elt* be the object of the triple (o_{post}, `rdf:first`, ?). Remove from the target graph the arc (o_{post}, `rdf:first`, *elt*). **If** *elt* is a blank node, **Then** apply the Cut operation on *elt*.
  - **Set** s_{post} to o_{post}, p_{post} to `rdf:rest` and o_{post} to the object of the triple (o_{post}, `rdf:rest`, ?) from the target graph.
- Remove from the target graph the arcs (s_{pre}, p_{pre}, o_{pre}) and (s_{post}, p_{post}, o_{post}). (NB: in some situations, they may be the same arc, or have already been removed by a previous step)
- **If** *col* is the empty collection
  - **Then**
    - Add in the target graph the arc (s_{pre}, p_{pre}, o_{post}).
  - **Else**
    - Add all the arcs resulting from the parsing of *col* to the target graph, let *fst* be the first node of the corresponding new collection, and *lst* the last node of that collection (excluding `rdf:nil`).
    - Remove from the target graph the arc (*lst*, `rdf:rest`, `rdf:nil`).
    - Add in the target graph the arcs (s_{pre}, p_{pre}, *fst*) and (*lst*, `rdf:rest`, o_{post}).

Here is an illustration of the previous algorithm.

Consider the graph represented in Fig. 1 Graph with a collection. The result of applying the operation `UpdateList :s :p 2..4 ("foo" "bar" "baz") .` on the collection in that graph can be seen in Fig. 2 Applying `UpdateList`.

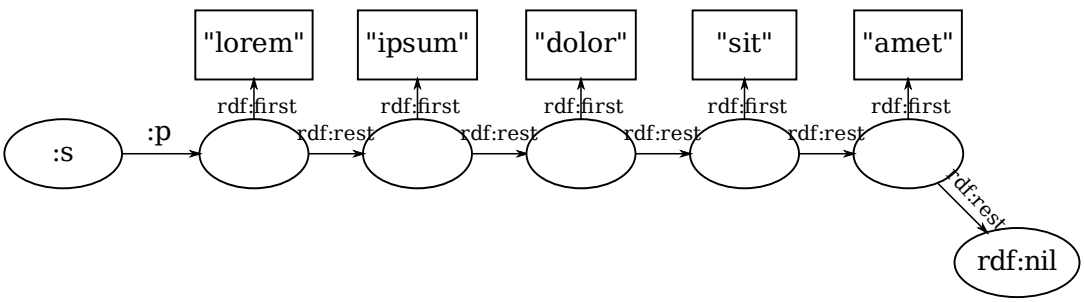

Fig. 1 Graph with a collection

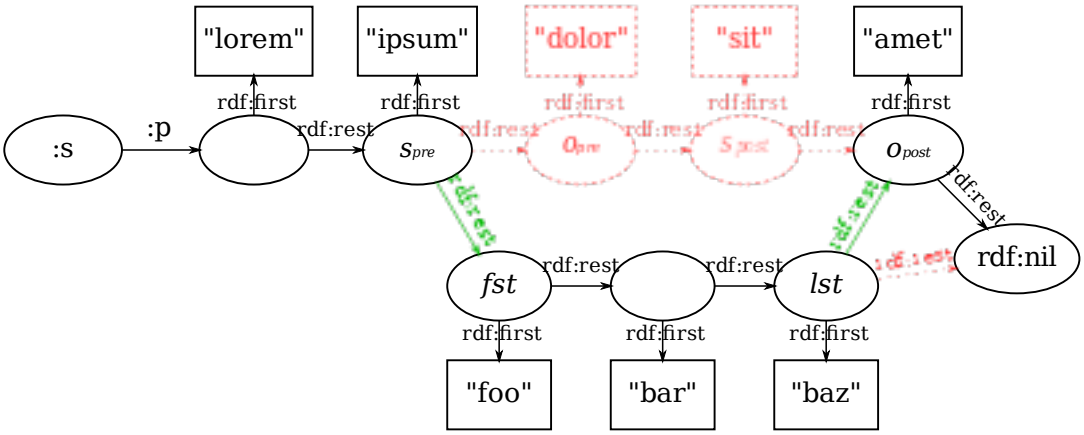

Fig. 2 Applying `UpdateList`

B. Internet Media Type, File Extension and Macintosh File Type

Contact:
Andrei Vlad Sambra
see also:
How to Register a Media Type for a W3C Specification
Internet Media Type registration, consistency of use
TAG Finding 3 June 2002 (Revised 4 September 2002)

The Internet Media Type / MIME Type for LD Patch is "text/ldpatch".

Pending discussion/registration with IETF.

It is recommended that LD Patch files have the extension ".ldp" (all lowercase) on all platforms.

Possible namespace conflict for .ldp!

It is recommended that LD Patch files stored on Macintosh HFS file systems be given a file type of "TEXT".

Type name:
text
Subtype name:
ldpatch
Required parameters:
None
Optional parameters:
charset — this parameter is required when transferring non-ASCII data. If present, the value of charset is always UTF-8 [UTF-8].
Encoding considerations:
The syntax of LD Patch is expressed over code points in Unicode [UNICODE]. The encoding is always UTF-8. Unicode code points may also be expressed using an `\uXXXX` (U+0000 to U+FFFF) or `\UXXXXXXXX` syntax (for U+10000 onwards) where `X` is a hexadecimal digit `[0-9A-Fa-f]`.
Security considerations:
Because of it's relation with Turtle, the same security considerations can be applied here. Applications may evaluate given data to infer more assertions or to dereference IRIs, invoking the security considerations of the scheme for that IRI. Note in particular, the privacy issues in [RFC3023] section 10 for HTTP IRIs. Data obtained from an inaccurate or malicious data source may lead to inaccurate or misleading conclusions, as well as the dereferencing of unintended IRIs. Care must be taken to align the trust in consulted resources with the sensitivity of the intended use of the data; inferences of potential medical treatments would likely require different trust than inferences for trip planning. Application rendering strings retrieved from untrusted LD Patch sources must ensure that malignant strings may not be used to mislead the reader. The security considerations in

the media type registration for XML ([RFC3023] section 10) provide additional guidance around the expression of arbitrary data and markup. LD Patch uses IRIs as term identifiers. Applications interpreting data expressed in LD Patch should address the security issues of Internationalized Resource Identifiers (IRIs) [RFC3987] Section 8, as well as Uniform Resource Identifier (URI): Generic Syntax [RFC3986] Section 7. Multiple IRIs may have the same appearance. Characters in different scripts may look similar (a Cyrillic "o" may appear similar to a Latin "o"). A character followed by combining characters may have the same visual representation as another character (LATIN SMALL LETTER E followed by COMBINING ACUTE ACCENT has the same visual representation as LATIN SMALL LETTER E WITH ACUTE). Any person or application that is writing or interpreting LD Patch data must take care to use the IRI that matches the intended semantics, and avoid IRIs that may look similar. Further information about matching of similar characters can be found in Unicode Security Considerations [UNICODE-SECURITY] and Internationalized Resource Identifiers (IRIs) [RFC3987] Section 8.

Interoperability considerations:
There are no known interoperability issues.

Published specification:
This specification.

Applications which use this media type:
No widely deployed applications are known to use this media type. It may be used by some web services and clients consuming their data.

Additional information:

Magic number(s):
LD Patch documents may have the string '@prefix' (case sensitive) near the beginning of the document.

File extension(s):
".ldp"

Macintosh file type code(s):
"TEXT"

Person & email address to contact for further information:
Andrei Vlad Sambra <andrei@w3.org>

Intended usage:
COMMON

Restrictions on usage:
None

Author/Change controller:
The LD Patch specification is the product of the LDP WG. The W3C reserves change control over this specifications.

C. Acknowledgements

This section is non-normative.

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

Andy Seaborne, Arnaud Le Hors, Ashok Malhotra, Eric Prud'hommeaux, Henry Story, John Arwe, Sandro Hawke, Steve Speicher, Tim Berners-Lee

D. Change Log

This section is non-normative.

D.1 Changes since March 2015 Candidate Recommendation

- Closed ACTION-156 by changing reference to "bcp47, as used by turtle”, based on i18n-issue-410.
- Fixed example
- Fixed typo resolution re: ISSUE-102
- Added Conformance section

D.2 Changes since September 2014 First Public Working Draft

- Negative indexes in Path and Slice (ISSUE-102 and ISSUE-104)
- New section comparing LD Patch with Turtle and SPARQL 1.1
- Add and Delete now takes Turtle as argument (ISSUE-101)
- New operations AddNew and DeleteExisting (ISSUE-103)
- The leading slash for Path expression is required (ISSUE-100)
- New dedicated Example section
- New section for Media Type registration
- New section about Error Handling, with specific behaviours
- Improvements and fixes in the grammar
- Removed the abstract model section, leading to a lighter operational semantics

E. References

E.1 Normative references

[BCP47]
A. Phillips; M. Davis. *Tags for Identifying Languages*. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

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

[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

[RFC3023]
M. Murata; S. St. Laurent; D. Kohn. *XML Media Types*. January 2001. Proposed Standard. URL: https://tools.ietf.org/html/rfc3023

[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

[RFC4918]
L. Dusseault, Ed.. *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. June 2007. Proposed Standard. URL: https://tools.ietf.org/html/rfc4918

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

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

[UNICODE]
The Unicode Standard. URL: http://www.unicode.org/versions/latest/

[UNICODE-SECURITY]
Mark Davis; Michel Suignard. *Unicode Security Considerations*. URL: http://www.unicode.org/reports/tr36/

[UTF-8]
F. Yergeau. *UTF-8, a transformation format of ISO 10646*. November 2003. Internet Standard. URL: https://tools.ietf.org/html/rfc3629

[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/

[rdf11-mt]
Patrick Hayes; Peter Patel-Schneider. *RDF 1.1 Semantics*. 25 February 2014. W3C Recommendation. URL: http://www.w3.org/TR/rdf11-mt/

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

E.2 Informative references

[sparql11-update]

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