**OSLC Tracked Resource Specification**

**29 March 2017**

**1. Introduction**

*This section is non-normative.*

OSLC Tracked Resource Set (TRS) provides a general-purpose mechanism for making a large set of resource URIs discoverable and for reporting ongoing changes affecting the set. This allows tools to expose a live feed of linked lifecycle data via a Tracked Resource Set in a way that permits other tool to build and maintain live, searchable information based on that linked data.

A TRS Server maintains one or more Resource Sets. A Tracked Resource Set consists of a finite, enumerable set of Tracked Resources identified by a URI. A Tracked Resource Set Client can discover the Tracked Resources Sets provided by a TRS Server and use this information to track changes to Tracked Resources for its purposes. The TRS Server will have its own criteria for determining the exact set of Tracked Resources in its Resource Sets at any point in time. However, TRS Clients need not be aware of the TRS Server's criteria, and will instead discover a Resource Set’s members by interacting with the TRS Server using the Tracked Resource Set protocol.

A TRS Server specifies an HTTP(S) URI corresponding to its Resource Set. This is referred to as the Tracked Resource Set URI.

A Tracked Resource Set is a resource representing the state of the Resource Set characterized in terms of a Base and a Change Log. The Base provides a point-in-time enumeration of the Tracked Resource members of the Resource Set while the Change Log provides a time series of adjustments describing changes to members of the Resource Set. This information can be used by TRS Clients to see what resources are tracked and to address changes to those resources as needed by the consumer.

**1.1 Typographical Conventions and Use of RFC Terms**

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

The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY*, and *OPTIONAL* in this specification are to be interpreted as described in [[*RFC2119*](https://tools.oasis-open.org/version-control/browse/wsvn/oslc-core/trunk/specs/tracked-resource-set.html#bib-RFC2119)].

**1.2 Motivation**

*This section is non-normative.*

TRS Clients can use TRS Server Tracked Resource Sets to build and maintain their own replica of the provider's resources. They read the TRS and get the base tracked resource member URIs. They then may do a GET on each member URI and copy the required properties into their own persistent store. When the TRS Client polls the TRS Server, it gets the Change Log and iterates through that, adding new tracked resources, deleting resources or updating modified resources according to change events in the change log.

A single TRS Client could get information from many different TRS Servers and tracked resource sets in order to aggregate information into a repository for more efficient federated access to the resource data, or for access using a different query language or protocol.

For example, a reporting server could provide SPARQL query access to lifecycle data exposed through Tracked Resource Sets. The reporting server could support a SPARQL endpoint to query data provided by tools that support TRS. Lifecycle tools could make data available for indexing by using Tracked Resource Sets (TRS). Clients can create and run SPARQL queries on the RDF dataset that aggregates the RDF graphs of the tracked resources. These queries include data from across the lifecycle tools; they also include cross-tool links between the resources. The change log in the tracked resource set captures any changes that happen to tracked resources, and the changes are propagated to the reporting server's repository, keeping it up to date.

A link index server could use TRS to provide access to links provided by link owners so that clients can query incoming links. This can be used to provide access to incoming links eliminating the need to store "backlinks" which are not allowed when CCM is used.

**1.3 Terminology**

*This section is non-normative.*

Terminology is based on OSLC Core Overview [[*OSLCCore3*](https://tools.oasis-open.org/version-control/browse/wsvn/oslc-core/trunk/specs/tracked-resource-set.html#bib-OSLCCore3)], W3C Linked Data Platform [[*LDP*](https://tools.oasis-open.org/version-control/browse/wsvn/oslc-core/trunk/specs/tracked-resource-set.html#bib-LDP)], W3C's Architecture of the World Wide Web [[*WEBARCH*](https://tools.oasis-open.org/version-control/browse/wsvn/oslc-core/trunk/specs/tracked-resource-set.html#bib-WEBARCH)], and Hyper-text Transfer Protocol [[*HTTP11*](https://tools.oasis-open.org/version-control/browse/wsvn/oslc-core/trunk/specs/tracked-resource-set.html#bib-HTTP11)].

*Tracked Resource Set (TRS)*

Describes a resource that defines a set of Tracked Resources expressed as a Base and a Change Log.

*Tracked Resource*

A resource identified by URI that is a member of one or more Tracked Resource Sets.

*Base*

The portion of a Tracked Resource Set representation that lists the Tracked Resources at some specific point in time. Change Events in the Change Log are relative to the Base.

*Change Log*

The portion of a Tracked Resource Set representation detailing a series of Change Events on Tracked Resources.

*Change Event*

Describes the addition, removal, or state change of a Tracked Resource in a Tracked Resource Set.

*TRS Patch*

An extended Change Event in a Tracked Resource Set detailing a change to the resource’s RDF representation.

*TRS Client*

An application or application component that consumes TRS resources to discover a set of resources and track changes to them.

*TRS Server*

An application or application component that provides Tracked Resource Sets.

*Access Context*

A grouping of resources with similar security requirements.

*Access Context List*

A resource describing a list of Access Contexts.

**1.4 Basic Concepts**

*This section is non-normative.*

The TRS Server maintains one or more Tracked Resource Sets. The members of a Tracked Resource Set consist of a finite, enumerable set of Resource URIs. The TRS Server will have its own criteria for determining the exact set of Tracked Resources at any point in time. TRS Clients can discover a Tracked Resource Set’s members by interacting with the TRS Server using the Tracked Resource Set protocol.

A HTTP GET request sent to the Tracked Resource Set URI returns a representation of the state of the Tracked Resource Set characterized in terms of a Base and a Change Log. The Base provides an enumeration of the Tracked Resource members of the Tracked Resource Set, at some specific point in time. The Change Log provides a time series of adjustments describing changes to the Tracked Resources. When the Base is empty, the Change Log describes a history of how the Tracked Resource Set has grown and evolved since its inception. When the Change Log is empty, the Base is a simple enumeration of the Tracked Resources in the Tracked Resource Set. This hybrid base+delta form gives the TRS Server flexibility to structure the representation in ways that are most useful to its TRS Clients. A TRS Server may periodically provide a TRS with just a Base in order to reset the Change Log to avoid excessively large Change Logs.

The Base portion of a Tracked Resource Set representation is a Linked Data Platform (LDP) Container where each member references a Tracked Resource that was in the Tracked Resource Set at the time the Base was computed. The Change Log portion is represented as multiple same-subject and same-predicate triples, where the objects correspond to Change Events. The order information is indicated within the Change Event entry itself. There must not be a gap between the Base portion and the Change Log portion of a Tracked Resource Set representation. However, the Change Log portion may contain earlier Change Event entries that would be accounted for by the Base portion. A “cutoff” property of the Base identifies the point in the Change Log at which processing of Change Events can be cut off because older changes are already covered by the Base portion. TRS Clients use the Base to establish the resources to track, and the Change Log to address changes to those resources. TRS Clients are responsible for knowing what change events they have already processed in the Change Log, and should only process new change events.

**2. Tracked Resource Set**

**2.1 General Rules**

An HTTP GET on a Tracked Resource Set URI returns a representation structured as follows.

**2.1.1 A TRS Server *MUST* provide HTTP(S) URIs corresponding to its Resource Sets. These are referred to as the Tracked Resource Set URIs.**

Example 1

# Resource: http://cm1.example.com/trackedResourceSet

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

<http://cm1.example.com/trackedResourceSet>

 a trs:TrackedResourceSet ;

 trs:base <http://cm1.example.com/baseResources/> ;

 trs:changeLog [

 a trs:ChangeLog ;

 trs:change ... .

 ] .

**2.1.2 A Tracked Resource Set *MUST* provide references to the Base and Change Log using the trs:base and trs:changeLog predicates respectively.**

**2.1.3 A typical TRS Client will periodically poll the Tracked Resource Set looking for recent Change Events. In order to cater to this usage, the Tracked Resource Set’s representation *MUST* contain the Change Log resource (i.e., via a Blank Node, or an inline named resource). Specifically the Tracked Resource Set representation *MUST* contain a triple {TRS-URI, rdf:type, trs:TrackedResourceSet} including the triples for the Change Events themselves enumerated in {TRS-URI, trs:change, ChangeEvent-URI} where the Change Events *MUST* be present in the Tracked Resource Set’s representation. The TRS Server *SHOULD* also support ETags, caching, and conditional GETs for Tracked Resource Set resources and relegate the Base to separate resources.**

**2.2 TRS Resource**

**2.2.1 A TRS Server *MAY* offer one or more Tracked Resource Sets.**

**2.2.2 Each Tracked Resource Set has a set of URIs to linked data Resources called Tracked Resources. The TRS Server decides which particular Tracked Resources are in a particular Tracked Resource Set at any moment. Both the Tracked Resource Sets and the linked data contents of each Tracked Resource *MAY* vary over time.**

**2.2.3 Tracked Resources *MUST* have a RDF representation, and *SHOULD* support GET requests specifying text/turtle as the acceptable media type and returning a Turtle serialization of RDF content in response. TRS Servers *MAY* support other RDF media types as well. The RDF content of a Tracked Resource is one RDF data graph representing one of the TRS Server’s linked data resources.**

**2.2.4 Tracked Resources *MAY* be Linked Data Platform RDF Sources (LDP-RS), and *MAY* support OSLC or LDP paging.**

**2.2.5 By retrieving a TRS Servers' Tracked Resource Set, an TRS Client can discover the URIs Tracked Resources. By retrieving the Tracked Resources, a TRS Client *MAY* discover the linked data representation of that resource.**

**2.3 Change Log**

A Change Log provides a set of changes, where the ordering of the changes is included with each Change Event. The following example illustrates the contents of a Change Log:

Example 2

# Resource: http://cm1.example.com/trackedResourceSet

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

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://cm1.example.com/trackedResourceSet>

 a trs:TrackedResourceSet ;

 trs:base <http://cm1.example.com/baseResources/> ;

 trs:changeLog [

 a trs:ChangeLog ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103> ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:32.000Z:102> ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> .

 ] .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103>

 a trs:Creation ;

 trs:changed <http://cm1.example.com/bugs/23> ;

 trs:order "103"^^xsd:integer .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:32.000Z:102>

 a trs:Modification ;

 trs:changed <http://cm1.example.com/bugs/22> ;

 trs:order "102"^^xsd:integer .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101>

 a trs:Deletion ;

 trs:changed <http://cm1.example.com/bugs/21> ;

 trs:order "101"^^xsd:integer .

As shown, a Change Log provides a set of Change Event entries in a multi-valued RDF property called trs:change.

**2.3.1 Change Events *MUST* have URIs (i.e., they cannot be Blank Nodes) to allow Clients to recognize entries they have seen before. The URI is only used to identify an event (i.e., it need not be HTTP GETable) and therefore *MAY* be a URN, as shown in the example.**

**2.3.2 Each Change Event has a sequence number, trs:order; sequence numbers are non-negative integer values that increase over time. A Change Event entry carries the URI of the changed Tracked Resource, trs:changed, and an indication, via rdf:type, of whether the Tracked Resource was added to the Tracked Resource Set, removed from the Tracked Resource Set, or changed state while a member of the Tracked Resource Set. The entry with the highest trs:order value (i.e., 103 in this example) is the most recent change. As changes continue to occur, a TRS Server *MUST* add new Change Events to the newest Change Log segment. The sequence number (i.e., trs:order) of newer entries *MUST* be greater than previous ones. The sequence numbers *MAY* be consecutive numbers but need not be.**

Note that the actual time of change is not included in a Change Event. Only a sequence number, representing the “sequence in time” of each change is provided. The URI of a Change Event *MUST* be guaranteed unique, even in the wake of a TRS Server rollback where sequence numbers get reused. A timestamp *MAY* be used to generate such a URI, as in the above example, although other ways of generating a unique URI are also possible.

**2.3.3 A Change Log represents a series of changes to its corresponding Tracked Resource Set over some period of time. The Change Log *MUST* contain Change Events for every Tracked Resource creation, deletion, and modification during that period. A TRS Server *MUST* report a Tracked Resource modification event if a GET on it would return a semantically different response from previously. For a resource with RDF content, a modification is anything that would affect the set of RDF triples in a significant way. A TRS Server *MAY* safely report a modification event even in cases where there would be no significant difference in response. Some cases of modifications that would be considered semantically different from previous or significant difference would be: inserted triple, removed triple, triple replaced (new object/literal, e.g. changing boolean literal “true” to “false”), replaced vocabulary term used (e.g. change from dcterms:title to rdfs:label).**

**2.3.4 The TRS Server *SHOULD NOT* report unnecessary Change Events although it might happen, for example, if changes occur while the Base is being computed. A TRS Client *SHOULD* ignore a creation event for a Tracked Resource that is already a member of the Tracked Resource Set, and *SHOULD* ignore a deletion or modification event for a Tracked Resource that is not a member of the Tracked Resource Set.**

**2.3.5 Change Log Format**

The Change Log in the previous example consisted of a single trs:ChangeLog resource. Typically, however, the Change Log will be very large, requiring the changes to be segmented into multiple smaller trs:ChangeLog resources:

Example 3

# Resource: http://cm1.example.com/trackedResourceSet

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

<http://cm1.example.com/trackedResourceSet>

 a trs:TrackedResourceSet ;

 trs:base <http://cm1.example.com/baseResources/> ;

 trs:changeLog [

 a trs:ChangeLog ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103> ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:32.000Z:102> ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> ;

 trs:previous <http://cm1.example.com/changeLog/1> .

] .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:33.000Z:103>

...

and then...

Example 4

# Resource: http://cm1.example.com/changeLog/1

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

<http://cm1.example.com/changeLog/1>

 a trs:ChangeLog ;

 trs:change <urn:urn-3:cm1.example.com:2010-10-27T17:39:30.000Z:100>, {more stuff} .

<urn:urn-3:cm1.example.com:2010-10-27T17:39:30.000Z:100>

...

As shown, the trs:previous reference is used in this case to connect to the Change Log resource containing the next group of chronologically earlier Change Events. The most recent Change Events are included in the Tracked Resource Set itself. This allows a TRS Client to easily discover the most recent Change Event, and retrieve successively older Change Log resources until it encounters a Change Event that has already been processed (on a previous check). The protocol does not attach significance to where a TRS Server breaks the Change Log into separate parts, i.e., the number of entries in a trs:ChangeLog is entirely up to the Server.

**2.3.5.1 To allow TRS Clients to retrieve the Change Events in a Change Log segment using a single HTTP GET request, TRS Servers *MUST* include all of the triples corresponding to a Change Log segment in the same HTTP response (i.e., in the representation of either the Tracked Resource Set or a trs:previous Change Log). This includes triples whose subject is the Change Log, the trs:change entries, potentially a trs:previous entry and the Change Events themselves. Other than the Change Events, all of these *MAY* be represented using Blank Nodes.**

**2.3.6 Truncated Change Logs**

**Editor: This text might be read as suggesting that change log segments are the points at which trucation is permitted to take place. I don't think this is the case. (img)**

**A chain of Change Logs *MAY* continue all the way back to the inception of the Tracked Resource Set and contain Change Events for every change made since then. However, to avoid maintaining this ever growing list of Change Logs indefinitely, a TRS Server *MAY* truncate the log at a suitable point in the chain. This can be accomplished by deleting the oldest segments of the Change Log and/or by removing the triples that reference them. In any case, TRS Clients *MUST* be prepared to receive HTTP status code 404 (Not found) when navigating the “previous” reference from a final or stale Change Log segment.**

**2.3.6.1 To ensure that a new TRS Client can always get started, the Change Log *MUST* contain the base cutoff event of the corresponding Base, and all Change Events more recent than it. Thus the TRS Server is only allowed to truncate Change Events older than the base cutoff event. When the Base has no base cutoff event (i.e., the Base enumerates the Tracked Resource Set at the start of time), the Change Log *MUST* contain all Change Events back to the start of time; i.e., no truncation is allowed.**

**Editor: There may be fewer than 7 days of changes. the intent is "do not truncate events that occurred within 7 days of the date of the base set. (img)**

**To minimize the likelihood of Clients falling too far behind and losing information, it is STRONGLY *RECOMMENDED* that a Server retain a minimum of seven days worth of Change Events.**

**2.4 Base Resources**

**2.4.1 The Base of a Tracked Resource Set is a W3C Linked Data Platform (LDP) Container where each member references a Tracked Resource that was in the Tracked Resource Set at the time the Base was computed. HTTP GET on a Base URI returns an LDP Container with the following structure:**

Example 5

# Resource: http://cm1.example.com/baseResources/

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

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

<http://cm1.example.com/baseResources/>

 a ldp:DirectContainer;

 ldp:membershipResource <http://cm1.example.com/baseResources/>;

 ldp:hasMemberRelation ldp:member;

 trs:cutoffEvent <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> ;

 ldp:member <http://cm1.example.com/bugs/1> ;

 ldp:member <http://cm1.example.com/bugs/2> ;

 ldp:member <http://cm1.example.com/bugs/3> ;

 ...

 ldp:member <http://cm1.example.com/bugs/199> ;

 ldp:member <http://cm1.example.com/bugs/200> .

**2.4.2 Each Tracked Resource in the Tracked Resource Set *MUST* be referenced from the container using an LDP membership predicate.**

**2.4.3 Because of the highly dynamic nature of the Tracked Resource Set, a TRS Server may have difficulty enumerating the exact set of resources at a point in time. Because of that, the Base can be only an approximation of the Tracked Resource Set. A Base might omit a resource that ought to have been included or include a resource that ought to have been omitted. For each erroneously reported resource in the Base, the TRS Server *MUST* at some point include a corrective Change Event in the Change Log more recent that the base cutoff event. The corrective Change Event corrects the picture for that Tracked Resource, allowing the TRS Client to compute the correct set of member Tracked Resources. A corrective Change Event might not appear in the Change Log that was retrieved when the TRS Client dereferenced the Tracked Resource Set URI. The TRS Client might only see a corrective Change Event when it processes the Change Log resource obtained by dereferencing the Tracked Resource Set URI on later occasions.**

**2.4.4 A TRS Server *MUST* refer to a given resource using the exact same URI in the Base (membership triple) and every Change Event (trs:changed reference) for that resource.**

**2.4.5 The response representation of a Base *MUST* include a trs:cutoffEvent property, whose value is the URI of the most recent Change Event in the corresponding Change Log that is already reflected in the Base. This corresponds to the latest point in the Change Log from which a TRS Client can begin incremental monitoring/updating if it wants to remain synchronized with further changes to the Tracked Resource Set. As mentioned above, the cutoff Change Event *MUST* appear in the non-truncated portion of the Change Log. When the trs:cutoffEvent is rdf:nil, the Base enumerates the (possibly empty) Tracked Resource at the beginning of time.**

**2.4.6 Paged Base**

Note (Feature Unstable): The paging support is based on the W3C Linked Data Platform (Paging) Specification that has not stabilized.

**Editor: the wording here is not quite right. That the base *MAY* be broken should not imply that the Server "will" respond with a 30x. A server is free to redirect as it sees fit, for any reason, nor is it required to redirect in case its base is paged. (img)**

The Base *MAY* be broken into multiple pages in which case the Server will respond with a 30x redirect message, directing the Client to the first “single-page resource”. The representation of a single-page resource will contain a subset of the Base’s membership triples. In addition, it will contain response header indicating a reference to the next page.

Below is an example of server-initiated paging and response headers:

HTTP Request:

Example 6

GET /baseResources/ HTTP/1.1

Host: cm1.example.com

Accept: text/turtle

HTTP Response:

Example 7

HTTP/1.1 303 See Other

Location: http://cm1.example.com/baseResources/page1

Following the redirect (server-initiated paging):

HTTP Request:

Example 8

GET /baseResources/page1 HTTP/1.1

Host: cm1.example.com

Accept: text/turtle

HTTP Response:

Example 9

HTTP/1.1 200 OK

Content-Type: text/turtle

Date: Wed, 11 Jun 2014 12:55:05 GMT

ETag: 2014-06-10T14:05:44.18Z

Link: <http://cm1.example.com/baseResources/page1>; rel="first",

 <http://cm1.example.com/baseResources/page2>; rel="next",

 <http://www.w3.org/ns/ldp#Page>; rel="type"

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

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

<http://cm1.example.com/baseResources/>

 a ldp:DirectContainer;

 ldp:membershipResource <http://cm1.example.com/baseResources/>;

 ldp:hasMemberRelation ldp:member;

 trs:cutoffEvent <urn:urn-3:cm1.example.com:2010-10-27T17:39:31.000Z:101> ;

 ldp:member <http://cm1.example.com/bugs/1> ;

 ldp:member <http://cm1.example.com/bugs/2> ;

 ldp:member <http://cm1.example.com/bugs/3> .

The last page in the list is indicated by omitting the link relation for next, for example omitting Link: rel="next". The Tracked Resource Set protocol does not attach significance to the order in which a Server enumerates the resources in the Base or breaks the Base up into pages.

The first single-page resource of a Base *MUST* include a trs:cutoffEvent property.

**Editor: This discussion of parallelism is confusing to me (img). Since the base is singly-linked list of pages, these pages cannot be fetched in parallel, so perhaps the suggestion is that earlier pages can be fetched opportunisticaly? (img)**

**Editor: I do not see how this protocol allows a Client to determine the base. Since the base and the changelog are paired together (in the TRS resource), yet comprised of distinct pages, there is the possibility that a retreived base page will be inconsistent either with the TRS, the changelog, or one of the previously read base pages. How should a server avoid this, or, indicate to a client that this has happened, and that the client needs to restart? (img)**

When a Base is broken into pages, the Client will discover and retrieve Base page resources to determine the Resources in the Base. A Client *MUST* retrieve all the page resources of the Base to compute the complete set of resources in the Base. A Client *MAY* retrieve the Base page resources in any order, including retrieving some Base page resources in parallel. A Client retrieves the Base page resources at its own pace, and *MAY* retrieve any of the Base page resources more than once. If the Server allows the representation of Base page resources to vary over time, the Server *MUST* ensure that the set of Resources a Client would infer as members is necessarily an approximation of the Resource Set which, when corrected by Change Events after the Base’s cutoff event, yields the correct set of member Resources in the Resource Set.

**2.5 TRS Patch**

*This section is non-normative.*

**Editor: Why is this non-normative? (img)**

**Editor: need some words here motivating the need for TRS Patch. Version 432 motivated using Indexable provider, which i think we agreed was not appropriate. (img)**

For a Resource that changes frequently, a typical Client may retrieve the same Resource over and over again. When the representation of the Resource is large and the differences between adjacent representations can be described compactly, including additional information in the trs:Modification Change Event can allow the Client to determine the Resource’s current representation and thereby avoid having to retrieve the Resource.

Similiarly, in versioned worlds each change to a versioned resource may result in the creation of a new Resource representing an immutable version of the resource. The typical Client retrieves each such Resource as it is created. The state of the new Resource is often quite similar to the state of a Resource corresponding to a previous version. When the state of one Resource is similar to that of another Resource and the differences between the two can be described compactly, including additional information in the trs:Creation Change Event can allow the Client to determine the new Resource’s resultant state from the potentially-known state of a previously-retrieved Resource and thereby avoid having to retrieve the new Resource.

This section describes an extension to Change Events allowing them to carry detailed information about modifications to resources with an RDF representation.

**2.5.1 The trspatch:createdFrom property, when present, identifies the antecedent resource. If omitted, the antecedent resource is the resource referenced in the trs:changed property. The antecedent resource is the one that supplies the “before” state.**

**2.5.2 The trspatch:rdfPatch property, when present, describes a patch applied to the antecedent resource’s representation. The result of applying the patch describes the representation of the resource referenced in the trs:changed property. The trspatch:rdfPatch property is used with trs:Modification and trs:Creation Change Events; it is not meaningful for trs:Deletion Change Events. The value of the trspatch:rdfPatch property is an LD Patch. The trspatch:rdfPatch property is meaningful only for resources with RDF representations.**

**2.5.3 The trspatch:beforeETag property, when present, gives the initial HTTP entity tag of the antecedent resource. This is the entity-tag value that would be returned in the HTTP ETag response header if the antecedent resource is retrieved immediately before the change.**

**2.5.4 The trspatch:afterETag property, when present, gives the final HTTP entity tag of the resource referenced in the trs:changed property. For a trs:Modification (trs:Creation) Change Event, this is the entity tag of the resource immediately after it was modified (created, respectively). This is the entity-tag value that would be returned in the HTTP ETag response header if the resource is retrieved immediately after the change.**

Note that these properties are can be used with any resource having both an RDF representation and an entity tag. This includes all Linked Data Platform RDF Source (LDP-RS) resources, which have both.

Note also that the trspatch:beforeETag and trspatch:afterETag properties are meaningful for any kind of resource, not just ones with RDF representations.

**2.5.5 LD Patch**

*This section is non-normative.*

**Editor: the LD Patch spec has moved up to working group note 28 july 2015. (img)**

**Editor: the meaning of "temporarily" below is not clear. (img)**

The Linked Data Patch (LD Patch) specification is currently under development by the W3C LDP WG. Our intention is to adopt the syntax and semantics of LD patches from the LD Patch specification rather than specifying our own. However, the LD Patch effort is only just beginning, and the First Public Working Draft was published on 18 September 2014.

In an effort to insulate Servers from changes to the LD Patch specification while it is being refined, this document proposes that Servers temporarily limit themselves to generating LD patches in a limited subset which we call Core format. Core format is extremely simple (no prefixes, no variables, and no Binds) but perfectly adequate for describing patches to graphs not involving blank nodes. (Core format is based on an early (and unofficial) precursor called RDF Patch.)

A Core format patch consists of a sequence of rows. A row with ‘A’ (or ‘D’) in the first column describes the addition (deletion) of one RDF triple from the resource’s RDF data graph. The subject, predicate, and object of the triples are described in columns 2-4 in the form of absolute URI references enclosed between ‘<’ and ‘>’. Each row is delimited by a ‘.’ and may have white space between the various terms in a row, including newlines.

Example of a Core format patch that deletes one RDF triple (subject http://example.com/bob, predicate http://xmlns.com/foaf/0.1/knows, object https://example.com/alice) and adds another an RDF triple (subject https://example.com/fred, predicate http://http://xmlns.com/foaf/0.1/member, object http://example.com/old-timers):

Example 10

D <http://example.com/bob> <http://xmlns.com/foaf/0.1/knows> <http://example.com/alice> .

A <http://example.com/fred> <http://http://xmlns.com/foaf/0.1/member> <http://example.com/old-timers> .

**2.5.6 TRS Patch Example 1**

*This section is non-normative.*

Turtle representation for the resource https://a.example.com/config/a1 in state 1. Assume that when the resource is retrieved in this state, the entity tag 15687ds9gha6s7 is returned in the ETag response header:

Example 11

# The following is the representation of

# https://a.example.com/config/a1

# in the state with entity tag 15687ds9gha6s7

@prefix dcterms: <http://purl.org/dc/terms/>.

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

<https://a.example.com/config/a1>

 a ldp:BasicContainer;

 dcterms:title "Component configuration A1";

 ldp:member <https://a.example.com/version/s/143>;

 ldp:member <https://a.example.com/version/r/577>;

 ldp:member <https://a.example.com/version/t/033>.

Turtle representation for the same resource https://a.example.com/config/a1 in state 2. Assume that when the resource is retrieved in this state, the entity tag 285d4h2ffgddd9 is returned in the ETag response header:

Example 12

# The following is the representation of

# https://a.example.com/config/a1

# in the state with entity tag 285d4h2ffgddd9

@prefix dcterms: <http://purl.org/dc/terms/>.

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

<https://a.example.com/config/a1>

 a ldp:BasicContainer;

 dcterms:title "Component configuration A1";

 ldp:member <https://a.example.com/version/s/143>;

 ldp:member <https://a.example.com/version/r/578>;

 ldp:member <https://a.example.com/version/t/033>.

Turtle representation for a Change Event describing resource https://a.example.com/config/a1 changing from state 1 to state 2:

Example 13

# The following is the representation of a change event

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

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

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

<urn:urn-3:a.example.com:2014-04-28T17:39:32.000Z:102>

 a trs:Modification;

 trs:changed <https://a.example.com/config/a1>;

 trs:order "102"^^xsd:integer;

 trspatch:beforeEtag "15687ds9gha6s7";

 trspatch:afterEtag "285d4h2ffgddd9";

 trspatch:rdfPatch

 """

 D <https://a.example.com/config/a1> <http://www.w3.org/ns/ldp#member> <https://a.example.com/version/r/577> .

 A <https://a.example.com/config/a1> <http://www.w3.org/ns/ldp#member> <https://a.example.com/version/r/578> .

 """.

**2.5.7 TRS Patch Example 2**

*This section is non-normative.*

Turtle representation for the resource https://a.example.com/sw-movie/versions/1. Assume that when the resource is retrieved in this state, the entity tag 783xhaty95 is returned in the ETag response header:

Example 14

# The following is the representation of

# https://a.example.com/sw-movie/versions/1

# in the state with entity tag 783xhaty95

@prefix dcterms: <http://purl.org/dc/terms/>.

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

<https://a.example.com/sw-movie/versions/1>

 dcterms:isVersionOf <https://a.example.com/sw-movie> .

<https://a.example.com/sw-movie>

 a ldp:Resource;

 dcterms:title "Star Wars".

Turtle representation for the resource https://a.example.com/sw-movie/versions/2. Assume that when the resource is retrieved in this state, the entity tag 212gyysxx8 is returned in the ETag response header:

Example 15

# The following is the representation of

# https://a.example.com/sw-movie/versions/2

# in the state with entity tag 212gyysxx8

@prefix dcterms: <http://purl.org/dc/terms/>.

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

<https://a.example.com/sw-movie/versions/2>

 dcterms:isVersionOf <https://a.example.com/sw-movie> .

<https://a.example.com/sw-movie>

 a ldp:Resource;

 dcterms:title "Star Wars: Episode IV - A New Hope".

Turtle representation for a Change Event describing the creation of the resource https://a.example.com/sw-movie/versions/2. The TRS patch describes the state of this new resource in terms of the state of resource https://a.example.com/sw-movie/versions/1:

Example 16

# The following is the representation of a change event

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

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

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

<urn:urn-3:a.example.com:2014-11-20T13:08:00.000Z:102>

 a trs:Creation;

 trs:changed <https://a.example.com/sw-movie/version/2>;

 trs:order "192"^^xsd:integer;

 trspatch:createdFrom <https://a.example.com/sw-movie/version/1>;

 trspatch:beforeEtag "783xhaty95";

 trspatch:afterEtag "212gyysxx8";

 trspatch:rdfPatch

 """

 D <https://a.example.com/sw-movie/versions/1> <http://purl.org/dc/terms/isVersionOf> <https://a.example.com/sw-movie> .

 A <https://a.example.com/sw-movie/versions/2> <http://purl.org/dc/terms/isVersionOf> <https://a.example.com/sw-movie> .

 D <https://a.example.com/sw-movie> <http://purl.org/dc/terms/title> \"Star Wars\".

 A <https://a.example.com/sw-movie> <http://purl.org/dc/terms/title> \"Star Wars: Episode IV - A New Hope\".

 """.

id='accessContext' class='informative'>

**3. Access Context**

A Client that provides services based on resources fetched from a Tracked Resource Set Server, may want to control access to those Resources. It is simple enough for a Client to allow some users to access these copies, while denying access to other users.

In order to make it feasible for Clients to offer access control that reflects the access control on the Tracked Resource Set Server, a Server can define one or more Access Contexts and associate each of its Resources with an Access Context. When configuring a Client to work with a particular Server, the administrator can query the Server for a list of relevant Access Contexts. This allows the administrator to configure access control at the level of Access Contexts within a Tracked Resource Set.

For its part, the Server associates each of its Resources with an Access Context, asserted in the representation of each Resource. This lets the Client connect access control rules expressed in terms of Access Contexts with the resource representations copied from the Server. Adding a resource to an Access Context, or removing one from it, changes the RDF representation of the resource. Like other changes affecting the RDF representation of the resource, this change is reported as a Change Event in the Server's Tracked Resource Set. This supports Clients working with Servers whose resources' Access Contexts vary over time.

**Editor: related to https://issues.oasis-open.org/browse/OSLCCORE-82. (img)**

This set of Access Contexts within a Server can also change over time. Adding a new Access Context Client will generally require an administrator to reconfigure the Client against that Server.

**3.1 Associations between Resources and Access Contexts**

The RDF acc:accessContext property is used to indicate that a resource belongs to an Access Context. The resource is the subject; the Access Context is the object.

* Property Name: accessContext
* Description: Access Context of the resource
* Property URI: http://open-services.net/ns/core/acc#accessContext

For example, the RDF statement (in Turtle):

Example 17

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

<https://a.example.com/defect/2314> acc:accessContext <https://a.example.com/acclist#alpha> .

declares the resource https://a.example.com/defect/2314 to be in the Access Context https://a.example.com/acclist#alpha.

**3.1.1 A linked data resource that is deemed (by the Server) to be in an Access Context *MUST* use the acc:accessContext predicate in its RDF representation to assert a relation between the linked data resource (subject) and an Access Context.**

For example, the above RDF statement embedded in the representation of resource https://a.example.com/defect/2314 asserts that this resource is in Access Context https://a.example.com/acclist#alpha. The RDF representation of a linked data resource in several Access Contexts will have multiple such RDF statements; for a linked data resource not in any Access Context, there will be none.

**3.2 Access Context List Resource**

If an Server uses Access Contexts within its resources, the Server *MUST* provide an Access Context List resource. If a Server has more than one Tracked Resource Set, it *MUST* designate an Access Context List resource for each Tracked Resource Set; several Tracked Resource Sets *MAY* share the same Access Context List resource.

The Access Context List resource is intended to be accessed by administrator for the purpose of configuring a TRS Client that is working with linked data obtained from that Server's Tracked Resource Set. The representation of the Access Context resource is itself linked data.

**3.2.1 The Server *MUST* support the use of the HTTP GET method for the Access Context List resource. The Server *SHOULD* require the use of TLS when making requests to the Access Context List resource. The Server *SHOULD* require authentication for the Access Context List resource, and *SHOULD* allow access only to users with administrative privileges. The Server's response *MUST* support the JSON-LD media type (application/ld+json), and *MAY* support other linked data representations. The response *SHOULD* include an ETag header.**

A client uses an HTTP GET request to retrieve a representation of the Access Context List resource, specifying JSON-LD as an acceptable format. For example:

Example 18

GET https://a.example.com/acclist HTTP/1.1

Accept: application/ld+json

Authorization: Basic [missing - admin user credentials]

**3.2.2 The response *MUST* be a JSON-LD format string with a node for the Access Context List along with a node for each Access Context. The response *SHOULD* use the simple @graph form with a default graph as shown in the example below. The response *SHOULD* use the @context value shown below (i.e., as a boilerplate header), and *SHOULD NOT* use other advanced JSON-LD features, since these can make the response more difficult to understand for human readers who only know JSON, and more difficult to processed programmatically by scripts without the benefit of a full JSON-LD library. The node’s type property gives the type of the node - either acc:AccessContextList or acc:AccessContext; the node’s id property gives the Access Context URI; the title and description properties give the title and description, respectively.**

Example of a response:

Example 19

HTTP/1.1 200 OK

Content-Type: application/ld+json;charset=UTF-8

ETag: 68djsgg82

{

 "@context": {

 "acc": "http://open-services.net/ns/core/acc#",

 "id": "@id",

 "type": "@type",

 "title": "http://purl.org/dc/terms/title",

 "description": "http://purl.org/dc/terms/description"

 },

 "@graph": [{

 "id": "https://a.example.com/acclist",

 "type": "acc:AccessContextList"

 }, {

 "id": "https://a.example.com/acclist#alpha",

 "type": "acc:AccessContext",

 "title": "Alpha",

 "description": "Resources for Alpha project"

 }, {

 "id": "https://a.example.com/acclist#beta",

 "type": "acc:AccessContext",

 "title": "Beta",

 "description": "Resources for Beta project"

 }]

}

**3.2.3 The response *MAY* include other properties. A client *MUST* ignore any properties that it does not understand.**

**3.2.4 URI Stability**

*This section is non-normative.*

Access Context List and Access Context resources should have stable URIs. When Access Context URIs are based on an Access Context List URI with the addition of local id in the fragment (e.g., the Access Context URI https://a.example.com/acclist#alpha is based on the Access Context List URI https://a.example.com/acclist), the Server should ensure that each Access Context has a stable local id that is unique within the Access Context List.

**4. Discovery**

**Editor: Documentation is not part of the spec is it? This ought to be non-normative? (img)**

The documentation for an TRS Server *MUST* document its Tracked Resource Sets, including the URI of each of the Server's Tracked Resource Set resources and designated Access Context List resources.

In order to help an administrator of an TRS Client in configuring its access to a Server's Tracked Resources, a Server *MAY* also make its Tracked Resource Setss discoverable. Discoverability is a convenience; an administrator can configure a Client with a particular Tracked Resource Set knowing just the URIs of the Server's Tracked Resource Set and designated Access Context List resource. An administrator can retrieve the Access Context List resource to discover the titles and URIs of the Access Contexts being used with that Server.

The RDF trs:trackedResourceSet property can be used to declare the whereabouts of a Tracked Resource Set resource. The Tracked Resource Set resource is the object.

This allows the existence and location of a Server's Tracked Resource Set resource to be declared with an RDF statement like the following (rendered here in Turtle):

Example 20

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

<> trs:trackedResourceSet <https://a.example.com/trs1> .

The RDF acc:accessContextList property declares the whereabouts of an Access Context List resource. The Access Context List resource is the object.

* Property Name: accessContextList
* Description: URI of Access Context List resource
* Property URI: http://open-services.net/ns/core/acc#accessContextList

This allows the existence and location of an Access Context List resource to be declared with an RDF statement like the following (rendered here in Turtle):

Example 21

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

<> acc:accessContextList <https://a.example.com/acclist> .

Where such RDF statements might be found is outside the scope of this specification.

Applications *MAY* provide multiple Tracked Resource Sets.

**5. Resource Constraints**

This document applies the following constraints to the Tracked Resource Set vocabulary terms.

**5.1 TrackedResourceSet**

**5.2 Base**

**5.3 ChangeLog**

**5.4 CreationEvent**

**5.5 ModificationEvent**

**5.6 DeletionEvent**

**5.7 Resource**

**5.8 AccessContext**

**5.9 AccessContextList**

**Appendix A. General Guidance**

*This section is non-normative.*

The following sections provide some general guidance on how to servers provide and clients can consume Tracked Resource Sets.

**A.1 Building a Local Replica**

*This section is non-normative.*

This section describes one (relatively straightforward) way that a Client can use the Tracked Resource Set protocol to build and maintain its own local replica of a Server’s Resource Set.

**A.1.1 Initialization procedure**

*This section is non-normative.*

A Client wishing to determine the complete collection of Resources in a Server’s Resource Set, so that it can build its local replica of the Resource Set, proceeds as follows:

1. Send a GET request to the Tracked Resource Set URI to retrieve the Tracked Resource Set representation to learn the URI of the Base.
2. Use GET to retrieve successive pages of the Base, adding each of the member Resources to the Client’s local replica of the Resource Set.
3. Invoke the Incremental Update procedure (below). The sync point event is the trs:cutoffEvent property (on the first page of the Base). A clever Client might run this step in parallel with the previous one in an effort to prevent the case where the Client can’t catch up to the current state of the Resource Set using the Change Log (after initial processing) because initial processing takes too long.

The overall work to build the local replica of the Resource Set is linear in the size of the Base plus the number of Change Events that occurred after the base cutoff event. The Server can help Clients building new local replicas of its Resource Set by providing as recent a Base as possible, because that means the Client will have to process fewer Change Events. It is entirely up to the Server how often it computes a new Base. It is also up to the Server how it computes the members of a Base, whether by enumerating its Resource Set directly (e.g., by querying an underlying database), or perhaps by coalescing its internal change log entries into a previous base.

**A.1.2 Incremental update procedure**

*This section is non-normative.*

Suppose now that a Client has a local replica of the Server’s Resource Set that is accurate as of a particular sync point event known to the Client. A Client wishing to update its local replica of the Server’s Resource Set acts as follows:

1. Send a GET request to the Tracked Resource Set URI to retrieve the Tracked Resource Set representation to learn its current Change Log.
2. Search through the chain of Change Logs from newest to oldest to find the sync point event. The incremental update fails if the Client is unable to locate the sync point (i.e., it gets to the end of the log).
3. Process all Change Events after the sync point event, from oldest to newest, making corresponding changes to the Client’s local replica of the Resource Set. Record the latest event processed as the new sync point event. A clever Client might record (some number of) recently processed events for possible future undo in the event of a server rollback.

When the procedure succeeds, the Client will have updated its own local replica of the Server’s Resource Set to be an accurate reflection of the set of resources as described by the retrieved representation of the Tracked Resource Set. Of course, the Server’s actual Resource Set may have undergone additional changes since then. While the Client may never catch up to the Server, it can at least keep its local replica of the Resource Set almost up to date. By choosing the interval at which it polls for updates, a Client controls how long the two are allowed to drift apart. The overall work to maintain the local replica of the Resource Set is linear in the length of the Change Event stream. In the (hopefully rare) situation that the Client fails to find its sync point event, one of two things is likely to have happened on the Server: either the Server has truncated its Change Log, or the Server has been rolled back to an earlier state.

If the Client had been retaining a local record of previously processed events, the Client may be able to detect a Server rollback if it notices the successor event of some previously processed event has been removed or changed to one with a different identifier than before. In this case, the Client can undo changes to its local replica back to that sync point, and then pick up processing from there.

Once the Incremental Update procedure fails, it is unlikely to succeed in the future. The Client has reached an impasse. The Client’s only way forward is to discard its local replica and start over.

**A.2 General Guidance for TRS Servers**

*This section is non-normative.*

There are a number of possible ways that a lifecycle tool could go about exposing its linked lifecycle data. Here is some general guidance:

* **Editor: These seem unwarranted reactions to a percieved usability problem. (img)**

A TRS Server should restrict itself to a small number of Tracked Resource Sets. When configuring a TRS Client, an administrator will typically have to select Tracked Resource Sets one at a time.

* A TRS Server should restrict itself to a static set of Tracked Resource Sets. When a Server Tracked Resource Set gets created dynamically, the administrator would be required to update the configurations of affected Clients.
* A Server’s Tracked Resource Sets should contain pairwise-disjoint sets of Tracked Resources. That is, a resource should not appear as a Resource in more than one Tracked Resource Set. A Server should document any overlap between its Tracked Resource Sets. (Some Clients are unable to work with overlapping Tracked Resource Sets.)
* A Server’s Resources should be linked data resources under the control of the Server itself, rather than linked data resources of some other lifecycle tool. In other words, a lifecycle tool should expose its own resources, not those of others.
* The RDF content of a Server's Resources should be statements about linked data resources under the control of the Server itself, rather than statements about linked data resources of some other lifecycle tool. In other words, the subjects of a lifecycle tool claims should be its own resources, as opposed to resources of some other lifecycle tool.
* A Tracked Resource may be one of the Server's regular linked lifecycle data resources, or it may be a resource containing an RDF data graph used specifically for exposing some linked data in a Tracked Resource Set.
* A Server should expose all of its linked lifecycle data via Tracked Resources in one of its Tracked Resource Sets. Any information that is held back will be unavailable to Clients.
* A Server’s combined RDF dataset should not repeat the same RDF statements.
* A Serv’s combined RDF dataset should not contain contradictory RDF statements.
* It is recommended that a Server report changes to its linked lifecycle data (including resource creations, deletions, and modifications) within 1 second of the changes being committed. Changes are reported via the Server's Tracked Resource Set Change Log. This helps ensure that Clients are able to obtain a live feed of changes in nearly real-time.
* It is recommended that a Server’s Tracked Resource Set include a Base not older than 7 days. This helps ensure that Clients are able to initially determine the resources in the set without having to process Change Events older than 7 days.
* It is recommended that a Server’s Tracked Resource Set Change Log retain Change Events for at least 7 days. This helps ensure that there are sufficient Change Events to allow a Client to catch up after a lengthy downtime or network outage.

**A.3 General Guidance for TRS Clients**

*This section is non-normative.*

A TRS Client does is akin to what a Web crawler does, and most of the same considerations apply.

A Client retrieves the TRS, Change Logs, and Base Resources, as well as some or all the Tracked Resources contained in the TRS. Except for the TRS Resource URI itself, the Client is blindly retrieving a succession of URIs that the Server includes in the Tracked Resource Set. An insufficiently wary Client can come to grief when it interacts with an imperfect or untrustworthy Server.

Most of the risks are always present: networks connecting Client to Server may experience delays and outages; and Server implementations may be imperfect (bugs in code, database corruptions). Moreover, when the Server is untrusted - when there is a concern the Server could attempt something nefarious - the Client needs to take extra steps to prevent itself from being misused or abused.

Here are risks and general guidance for Clients:

* The size and rate of change of a Server's resource set reflects the amount of linked data that a Server has to make available and how often that data changes. These vary considerably and are often difficult to estimate in advance. A Client that is maintaining a copy of a Server's Tracked Resources should establish reasonable limits and monitor the size and rate of change so that a misbehaving Server does not cause the Client to spend an unreasonable amount of effort (network communication and storage) in doing so. This should include caps on the number and representation sizes of resources (including TRS Change Logs and TRS Base pages, for example).
* Since the Client is retrieving resources of various kinds over the network from the Server, which takes time and network connections, the Client should do so in a way that does not prevent it from doing other useful work while waiting for responses. The Client should be tolerant of failures due to network failures and outages, and carry over important work until the blockage has been removed. On the other hand, the Client should not allow its queue of work to grow without bound since that may make it difficult for the Client to clear the backlog. The Client should also be polite to the Server, and avoid making multiple requests per second and/or downloading large files that might make it hard for the Server to keep up with its normal workload.
* A Server’s Resources should be linked data resources under the control of the Server itself, rather than linked data resources of some other lifecycle tool. A Client that does not trust a Server in this regard should mitigate the risk by keeping a server whitelist for each Server and refusing to retrieve the Resources of a given Server when the server is not on the whitelist. Without something like this, a Client can be tricked into retrieving and indexing resources that it should not, such as other resources located on a different server that the Client also happens to have access to.
* The RDF content of a Tracked Resource should be statements about linked data resources under the control of the Server itself, rather than statements about linked data resources of some other lifecycle tool. In other words, the subjects of a lifecycle tool claims should be its own resources, as opposed to resources of some other lifecycle tool. A Client that does not trust a Server in this regard should mitigate the risk by keeping a content whitelist for each Server and rejecting the RDF content of Tracked Resources of a given Server when the URIs of subjects are not all on the content whitelist. Without something like this, a Server can affect a Client with arbitrary claims which may interfere with or contradict authoritative claims made by the legitimate owner.

**A.4 Access Context Guidance**

*This section is non-normative.*

There are several things to consider when deciding how a lifecycle tool can make use of Access Contexts. Before suggesting possible designs, here are some characteristics that will help ensure a lifecycle tool will be useful to administrators tasked with configuring access to the Tracked Resources that have been retreived by a TRS Client:

* **Optional**. A Server should only use Access Contexts if there are reasons why an administrator might want to impose differential access in Clients.
* **Understandable**. An administrator should be able to intuit from the Access Context name and description what kinds of resources are in it.
* **Useful collections**. An Access Context should contain resources that can be treated similarly. Same security classification. An Access Context should contain resources with the same security classification.
* **Reasonable number**. The list of Access Contexts should not be so long as to overwhelm the administrator.
* **Stable**. The set of Access Contexts should be more or less static. Changes to the Access Context list will generally require the administrator to update configurations of Clients.
* **Centralized**. A Server should host a single Access Context List resource enumerating the Access Contexts used in any of its Tracked Resources, unless there are reasons to do otherwise.

The following recipes suggest some of the designs that are possible.

**Recipe 1**: Your tool has top-level objects called workspaces. New workspaces are created infrequently, and only by administrators. Each linked data resource is associated with a single workspace. Teams of users work in the context of a single workspace. All the resources in a workspace have the same security classification.

Your tool should treat each workspace as a separate Tracked Resource Set, and not use Access Contexts.

An administrator can always control access to the linked data in a Client on an TRS by TRS basis, and grant users access to linked data from some workspaces but not others.

**Recipe 2**: Your tool has top-level objects called projects. New projects are created infrequently, and only by administrators. Each linked data resource is associated with a single project. Teams of users work in the context of a set of projects. All the resources in a project have the same security classification.

Your tool should treat all projects as part of a single Resource Set, and automatically create Access Contexts in 1-1 correspondence with projects, taking on the name and description of the project.

An administrator can control access to the linked data in an Client on a project by project basis, and grant users access to linked data from some projects but not others.

**Recipe 3**: Your tool has resources that can be tagged as containing confidential customer information. Teams of users work in the context of your tool. In the customer’s organization, only some employees are allowed access to confidential customer information.

Your tool should have a single Tracked Resource Set, and automatically create an Access Context named “Confidential Customer Data” and assigns all tagged resources to this Access Context. Other resources are left “loose”; i.e., not included in any Access Context.

An administrator for a Client can control access to the confidential customer information separately from the regular linked data.

**Recipe 4**: Your tool has many resources. Teams of users work in the context of your tool. The customer’s organization has strict policies on what information can be shown to which employees.

Your tool should have a single Tracked Resource Set. Your tool should let an administrator define a set of custom Access Contexts. Your tool should let users (or possibly just administrators) associate resources with these Access Contexts.

An administrator can control access to the linked data in a Client based on these custom Access Contexts.

**A.5 TRS Patch Guidance**

*This section is non-normative.*

The following sections provide general guidlines on using the TRS Patch capability.

**A.5.1 TRS Patch Guidance for Servers**

*This section is non-normative.*

When the state of a Tracked Resource changes, the Server adds a trs:Modification Change Event to a Change Log. The Change Event describes a transition between two definite representations states of the Tracked Resource. In principle, the entity tags of the two states, and the LD patch between the two RDF representations, are all well-defined. This much is true whether or not the Server chooses to embed those pieces of information in the Change Event.

The decision as to whether to provide an LD Patch for a trs:Modification Change Event should be made on a case-by-case basis. Just because one Change Event for a resource includes an LD Patch, that does not mean that all Change Events for the same resource should also include an LD Patch.

Server developers should remember that a Client wishing to discover the current state of a resource can always do so using HTTP GET to retrieve the resource. Including an LD Patch in a Change Event is an optional embellishment that allows some Client under the right circumstances to determine the new current state of a resource instead of re-retrieving the resource. It is up to the Server to decide whether including an LD patch is likely to be worthwhile.

However, whenever a trs:Modification Change Event includes a trspatch:rdfPatch, it should also include accurate trspatch:beforeETag and trspatch:afterETag properties. Without all 3 pieces of information, a Client is unlikely to be able to do better than re-retrieving the resource to discover its updated state.

When the RDF representation of the resource contains a large number of RDF triples and the number of rows in the LD Patch is small, including the LD patch in the Change Event is recommended, and may improve overall system performance by allowing Clients to avoid having to re-retrieve the resource to discover its updated state. Similiarly, whenever a trs:Creation Change Event includes a trspatch:rdfPatch, it should also include a trspatch:createdFrom along with accurate trspatch:beforeETag and trspatch:afterETag properties.

Conversely, when the number of affected RDF triples is large, the size of the LD Patch becomes significant. Including the LD Patch in the Change Event is not recommended because it bloats the size of Change Events in the Change Log, which may negatively impact performance. Omitting the LD patch from the Change Event is likely to give better overall performance.

**A.5.2 TRS Patch Guidance for Clients**

*This section is non-normative.*

A typical Client is tracking the state of some or all Tracked Resources in a Resource Set. When the Client first discovers the Resource, whether through a trs:Creation Change Event in the Change Log or an entry in the Base, the Client uses HTTP GET to retrieve the current state of the Resource and gets back its RDF representation. When the response includes an entity tag for the resource in its current state, as it will when the Index Resource is a LDP-RS, the Client remembers both the RDF representation and entity tag as the state of that Index Resource.

When the Client processes a trs:Modification Change Event for the Resource in the Change Log, it learns that the Resource has changed state. This means that the Client’s remembered RDF representation and entity tag for the Resource are no longer accurate, which cues the Client to discard the remembered RDF representation and re-retrieve the Resource. However, when the Change Event includes a TRS Patch, the Client may have a second option. When the trspatch:beforeETag value matches the Client’s remembered entity tag, the Client can apply the trspatch:rdfPatch to its remembered RDF representation to compute a replacement RDF representation, which can be remembered along with the trspatch:afterETag value as the entity tag. When this happens, the Client can process the trs:Modification Change Event for the Resource without a network request. It is clearly advantageous for a Client to behave this way whenever possible. On the other hand, if the trspatch:beforeETag value does not match the Client’s remembered entity tag, the Client cannot apply the trspatch:rdfPatch, and should treat the Change Event as if the TRS Patch were absent.

Similarly, when the Client processes a trs:Creation Change Event for the Resource in the Change Log of the Tracked Resource Set, the Client learns of the existence of a new Resource. This cues the Client to retrieve the new Resource. However, when the Change Event includes a TRS Patch, the Client may have a second option. When the Clienthas previously retrieved and remembered the resource identified by trspatch:createdFrom in the state with entity tag matching trspatch:beforeETag, the Client can apply the trspatch:rdfPatch to the Client’s remembered RDF representation to compute an RDF representation of the new Resource, which can be remembered along with the trspatch:afterETag value as the entity tag. When this happens, the Client can process the trs:Modification Change Event for the Resource without having to retrieve the new Resource. It is clearly advantageous for a Clientto behave this way whenever possible. On the other hand, if the trspatch:beforeETag value does not match the Client’s remembered entity tag, the Client cannot apply the trspatch:rdfPatch and should treat the Change Event as if the TRS Patch were absent.

Risk-wise, TRS Patches provide a way for a Server to tamper with the RDF representations of another server’s resources in a Client without the other server’s involvement. The mitigations covered in General Guidance for Clients, above, will address this risk as well. The Clients’s server whitelist for an untrusted Tracked Resource Set should be used to vet trspatch:createdFrom URIs, and its content whitelist should be used to vet subjects in the results of applying TRS patches.