OASIS OPEN PROJECTS

OSLC Tracked Resource Set Guidance Version 1.0

Project Note 01
16 December 2021

This stage:

https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/pn01/trs-guidance.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/pn01/trs-guidance.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/trs-guidance.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/trs-guidance.pdf

Latest editor's draft:

https://oslc-op.github.io/oslc-specs/notes/trs-guidance/trs-guidance.html

Open Project:

OASIS Open Services for Lifecycle Integration (OSLC) Open Project

Project Chairs:

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

Editor:

Nick Crossley (nick_crossley@us.ibm.com), IBM

Related work:

This specification is related to:

- *OSLC Tracked Resource Set Version 3.0. Part 1: Specification*. Edited by Nick Crossley. Latest Stage: https://docs.oasis-open-projects.org/oslc-op/trs/v3.0/tracked-resource-set.html

Abstract:

This document contains material for a TRS tutorial - it is fragmented, contains repetition, and is not yet ready for publication.

Several current editors and project members feel many of the opinions expressed in the Guidance sections are excessively strict or judgemental, and not appropriate for general guidance.

Status:

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

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

Citation format:

When referencing this specification the following citation format should be used:
[TRS-Guidance-v1.0]
OSLC Tracked Resource Set Guidance Version 1.0. Edited by Nick Crossley. 16 December 2021. OASIS Project Note 01. https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/pn01/trs-guidance.html. Latest stage: https://docs.oasis-open-projects.org/oslc-op/trs-guidance/v1.0/trs-guidance.html.

Notices

Table of Contents

1. Timescales and event frequency

In order to provide adequate response to client requests, a TRS server needs to allow those clients sufficient time to read the base, the change log, and process the set of tracked resources. However, the data volumes and timescales involved in TRS processing are likely to vary between servers for different applications. A server representing Amazon transactions might have many events per second, while a server representing exhibits at a museum might have a few events per month. The cost of processing a single event is also likely to vary between applications; reading a new or modified resource with 5 RDF properties will take less time than reading one with 5,000 properties.

For these reasons, the TRS specification does not impose specific constraints over the length of time for which a TRS base must remain readable, nor what the degree of overlap should be between a base and a corresponding change log. A server implementing TRS must consider, and should document, the quality of service it will provide in terms of the size of pages in the base or change log, how long base pages are kept, how long change events are kept, and the minimum period for which change events behind the latest base cutoff are kept.

2. Motivation and Use Cases for Patch Events

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.

Similarly, 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.

3. General Guidance

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

3.1 Building a Local Replica

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.

3.1.1 Initialization procedure

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.

3.1.2 Incremental update procedure

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.

3.2 General Guidance for TRS Servers

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

- 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.

3.3 General Guidance for TRS Clients

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.

TRS Clients are responsible for knowing what change events they have already processed in the Change Log, and should only process new change events.

While servers preserve the cutoff event in the truncated change log, the same is not true of any earlier events that a client might have already processed. For this reason, a client should not assume that it will find a change event that it has previously processed: that change event might have been truncated.

If an insufficiently wary client reads the tracked resources themselves, some risks are 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.

3.4 Access Context Guidance

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.

3.5 TRS Patch Guidance

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

3.5.1 TRS Patch Guidance for Servers

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.

3.5.2 TRS Patch Guidance for Clients

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.

Appendix A. Acknowledgements

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

Project Governing Board:

James Amsden, IBM (co-chair)
Andrii Berezovskyi, KTH (co-chair)
Axel Reichwein, Koneksys

Techical Steering Committee:

James Amsden, IBM
Andrii Berezovskyi, KTH
Axel Reichwein, Koneksys

Additional Participants:

Nick Crossley
David Honey

