Shape Trees Specification

Editor's Draft, 3 December 2021

▾ **More details about this document**

This version:
https://shapetrees.org/TR/specification/

Issue Tracking:
GitHub

Editors:
Eric Prud'hommeaux
Justin Bingham

Abstract

Semantic Web Applications interoperate by sharing semantics of terms and constellations of resource-oriented data structures. This specification defines shape trees, a mechanism for declaring and operating over constellations of resource-oriented data structures.

Status of this document

Table of Contents

§ 1. Introduction

This section is non-normative.

Realizing the value proposition of the Semantic Web lies in building useful and robust applications that can interoperate over linked data. Protocols such as [LDP] and [Solid] organize linked data graphs into resource hierarchies, providing a foundation upon which these robust and interoperable applications can be created.

Application interoperability depends on applications sharing semantics for relationships and data structures. Existing technologies fulfill portions of those dependencies:

- [RDF]'s foundation in unambiguous identifiers provides an infrastructure that allows for interoperability, but does not specifically encourage or enforce it.
- Shape languages (e.g. [ShEx] and [SHACL]) provide machine-readable, enforceable data structure definitions on single resources.

For applications that operate on more complex and interconnected resources, *Shape Trees* express the layout of those resources and associate them with their respective shapes.

Shape trees marry [RDF] vocabularies, shapes, and resources into "little trees" that provide machine to machine interoperability, combining them into concepts that humans can easily comprehend, such as medical records, notes, notebooks, calendars, and financial records.

This allows one to treat a set of related resources as a single grouping, and apply that to a range of operations including access control, data organization, data validation, and data migration.

While shape trees are intended to adapt to different technology platforms that support the notion of containers and resources, examples in this specification will reflect usage in an [LDP] environment.

Shape trees are defined as an RDF graph structure that expresses a set of expected behaviors by agents that work with them. This provides a sort of type-safety of resource hierarchies called ***shape tree consistency***. These semantics CAN be implemented by a server-side agent, or by a client-side agent that implements shape tree operations as primitive requests to a server.

Shape tree support by a server-side agent ensures shape tree consistency by managing all manipulations of data within a resource hierarchy (see managed resource).

§ 1.1. ShapeTree Support From Proxy or Client-side Library

If a server does not support shape trees, some shape tree consistency can be achieved by implementing shape tree support in the client, typically in a library than can enforce consistency for any clients using the library. Primitive operations by other clients not using the library may leave the resource hierarchy in an inconsistent state.

For client-side shape tree libraries that operate by intercepting HTTP operations, this specification serves as an API for those client interactions. (Additionally, if shape tree support is later added to the server, the client's execution of shape tree operations does not change.) In the remainder of this document, shape tree operations are described in terms of a client-side agent performing operations on a server-side agent with support for shape trees.

A proxy performing shape tree operations would be indistinguishable from server support except that clients performing primitive operations directly on the server (bypassing the proxy) may leave the server in an inconsistent state.

§ 2. Shape Tree

A ***shape tree*** is a machine-readable template describing the expected layout of a tree of resources in a container-based ecosystem. A shape tree expresses a tree hierarchy by containing other shape trees. The terms used to express a shape tree are described using an [RDF] vocabulary.

A ***managed instance*** is a resource assigned to and in conformance with one or more shape trees via a shape tree manager. The primary resource in a managed instance is called a ***managed resource***.

Every managed resource has an associated shape tree manager. A shape tree manager identifies the shape tree associated with a managed resource, and additional information needed to navigate nested hierarchies of managed resources. A resource becomes a managed resource when a shape tree manager is associated with it through the § 4.2 Plant Shape Tree operation.

The `st:expectsType` property of a shape tree specifies that the described managed resource be one of these three types:

`st:Resource`	Regular RDF resource that is not a container
`st:Container`	RDF resource that uses server-managed metadata to enumerate nested resources
`st:NonRDFResource`	Non-RDF resource such as binaries or images

The `st:shape` property specifies that the described managed resource conforms to the stated [ShEx] or [SHACL] shape.

Shape trees prescribe physical hierarchies and can reference other shape trees to

form virtual hierarchies.

For physical hierarchies, the `st:contains` property asserts that a managed resource is a container that explicitly contains another managed resource.

If shape tree `S1` `st:contains` shape tree `S2`, `S1` describes a container that contains another managed resource described by `S2`. For example, in [LDP], `S1` describes an [LDP] container which `ldp:contains` nested resources described by `S2`. Shape tree `S2` and the nested resources associated with it are considered to be ***in-scope*** of the containing shape tree `S1`.

Figure 1 *Shape tree validation of a physical hierarchy*

Managed Resource	Associated Shape Tree
/project-1/	ex:ProjectTree
-- /milestone-A/	ex:MilestoneTree
---- /task-43/	ex:TaskTree
---- /task-48/	ex:TaskTree
------ /attachment-aa89	st:NonRDFResourceTree
---- /task-61/	ex:TaskTree
---- /issue-22/	ex:IssueTree
------ /attachment-cd12	st:NonRDFResourceTree
------ /attachment-ef55	st:NonRDFResourceTree
---- /issue-31/	ex:IssueTree

```
PREFIX st: <http://www.w3.org/ns/shapetrees#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example.com/ns/ex#>

<> st:describedBy ex:project-en, ex:project-es, ex:project-nl, ex:project-ko .

<#ProjectTree>
  a st:ShapeTree ;
  st:expectsType st:Container ;
  st:shape ex:ProjectShape ;
  st:contains <#MilestoneTree> .

<#MilestoneTree>
  a st:ShapeTree ;
  st:expectsType st:Container ;
  st:shape ex:MilestoneShape ;
  st:contains <#TaskTree>, <#IssueTree> .

<#TaskTree>
  a st:ShapeTree ;
  st:expectsType st:Container ;
  st:shape ex:TaskShape ;
  st:contains st:NonRDFResourceTree .

<#IssueTree>
  a st:ShapeTree ;
  st:expectsType st:Container ;
  st:shape ex:IssueShape ;
  st:contains st:NonRDFResourceTree .
```

A virtual hierarchy is defined by shape tree references that link to other shape trees by the `st:references` property. A ***shape tree reference*** identifies the shape tree to be referenced via `st:referencesShapeTree`, and the shape path through which it is linked via either `st:viaShapePath`, or `st:viaPredicate`.

A ***shape path*** is a string that defines a traversal of a shape schema. [SHEXPATH]

Figure 2 *Shape tree validation of a virtual hierarchy*

Managed Resource	Associated Shape Tree
/project-1	ex:VirtualProjectTree
/milestone-A	ex:VirtualMilestoneTree
/task-43	ex:VirtualTaskTree
/task-48	ex:VirtualTaskTree
/attachment-aa89	st:NonRDFResourceTree
/task-61	ex:VirtualTaskTree
/issue-22	ex:VirtualIssueTree
/attachment-cd12	st:NonRDFResourceTree
/attachment-ef55	st:NonRDFResourceTree
/issue-31	ex:VirtualIssueTree

```
<#VirtualProjectTree>
  a st:ShapeTree ;
  st:expectsType st:Resource ;
  st:shape ex:ProjectShape ;
  st:references [
    st:referencesShapeTree <#VirtualMilestoneTree> ;
    st:viaShapePath "@ex:ProjectShape~ex:hasMilestone"
  ] .

<#VirtualMilestoneTree>
  a st:ShapeTree ;
  st:expectsType st:Resource ;
  st:shape ex:MilestoneShape ;
  st:references [
    st:referencesShapeTree <#VirtualTaskTree> ;
    st:viaPredicate ex:hasTask
  ] ,
  [
    st:referencesShapeTree <#VirtualIssueTree> ;
    st:viaPredicate ex:hasIssue
  ] .

<#VirtualTaskTree>
  a st:ShapeTree ;
  st:expectsType st:Resource ;
  st:shape ex:TaskShape ;
  st:references [
    st:referencesShapeTree st:NonRDFResourceTree ;
    st:viaShapePath "@ex:TaskShape~ex:hasAttachment"
  ] .

<#VirtualIssueTree>
  a st:ShapeTree ;
```

```
st:expectsType st:Container ;
st:shape ex:IssueShape ;
st:references [
  st:referencesShapeTree st:NonRDFResourceTree ;
  st:viaShapePath "@ex:IssueShape~ex:hasAttachment"
] .
```

Let ST be a shape tree. Let STI be a corresponding managed instance.

- An { ST st:expectsType T } triple identifies the resource type T of a corresponding managed resource R in STI where T MUST be one of st:Resource, st:Container, or st:NonRDFResource.
- An { ST rdfs:label L } triple indicates that there is at most one corresponding managed resource R in STI and it has the name L. L is a ***static resource***.
- An { ST st:shape SH } triple indicates that managed resource R has at most one node which conforms to shape SH.
- An { ST st:contains ST2 } triple identifies a nested shape tree ST2.
- An { ST st:references STR } triple indicates that the shape tree identified in shape tree reference STR, is referenced through the instance data of STI.
    - An { STR st:referencesShapeTree ST3 } triple indicates a shape tree referenced through the instance data of STI
    - An { STR st:viaShapePath PATH } triple identifies the shape path through which a managed instance of ST3 can be found via the instance data of STI
    - An { STR st:viaPredicate PRED } triple identifies the RDF predicate through which a managed instance of ST3 can be found via the instance data of STI

§ 2.1. Shape Tree Schema

Figure 3 *ShEx Schema for a Shape Tree*

```
<#ShapeTreeShape> {
  a [st:ShapeTree] ;
  (
    st:expectsType [st:Container] ;
    st:contains @<#LocalOrExternalShape> * ;
    |
    st:expectsType [st:Resource st:NonRDFResource]
  ) ;
  rdfs:label xsd:string ? ;
  st:references @<#ReferenceShape> * ;
  st:shape IRI ?
}

<#LocalOrExternalShape>
  @<#ShapeTreeShape> OR
  ((@<#ReservedShapeTree> OR IRI) AND {})

<#ReservedShapeTree> [
  st:ResourceTree
  st:ContainerTree
  st:NonRDFResourceTree
]
```

§ 3. Assigning Shape Trees

§ 3.1. Shape Tree Manager

A ***shape tree manager*** associates a managed resource with one or more shape trees. No more than one shape tree manager may be associated with a managed resource.

A shape tree manager includes one or more shape tree assignments.

The server MUST advertise the URI of the shape tree manager for a given resource by responding to HTTP requests of that resource with an included `HTTP Link` header with a `rel` value of `http://www.w3.org/ns/shapetrees#managedBy` and the manager resource as the link target.

Conversely, the server MUST advertise the URI of the resource managed by a shape tree manager by responding to HTTP requests of the manager resource with an included `HTTP Link` header with a `rel` value of `http://www.w3.org/ns/shapetrees#manages` and the managed resource as the link target.

Figure 4 *Shape Tree Manager properties*

Property	Description
`st:hasAssignment`	Links a shape tree assignment of the shape tree manager

§ 3.2. Shape Tree Assignment

A ***shape tree assignment*** is used to associate a given shape tree with a managed resource.

Shape tree assignments identify key contextual points in a physical hierarchy. A ***root shape tree assignment*** is set by the Plant Operation, and any subsequent assignment applied within that managed hierarchy reference it via `st:hasRootAssignment`.

If there is more than one shape tree assignment in a shape tree manager, they must all apply to the same managed resource associated with the shape tree manager.

Figure 5 *Shape Tree Assignment properties*

Property	Description
`st:assigns`	Identifies the shape tree to be associated with the managed resource
`st:manages`	Identifies the managed resource associated with the shape tree assignment
`st:hasRootAssignment`	Identifies the root shape tree assignment
`st:focusNode`	Identifies the focus node for shape validation in the associated managed resource, and is only valid when the corresponding shape tree includes `st:shape`
`st:shape`	Identifies the shape to which `st:focusNode` must conform, and must be equivalent to `st:shape` in the corresponding shape tree

A root shape tree, and its corresponding managed resource can be planted within an existing managed hierarchy, alongside or within other root shape trees and managed resources.

Shape tree assignments in a given shape tree manager may have different focus nodes.

Figure 6 *Navigating a physical shape tree hierarchy*

Managed Resource	Shape Tree	Root Shape Tree	Root Managed Resource
/data/	ex:DataTree	ex:DataTree	/data/
-- /projects/	ex:DataCollectionTree ex:ProjectCollectionTree	ex:DataTree ex:ProjectCollectionTree	/data/ /data/projects/
---- /project-1/	ex:ProjectTree	ex:ProjectCollectionTree	/data/projects/
------ /milestone-A/	ex:MilestoneTree	ex:ProjectCollectionTree	/data/projects/
-------- /task-43/	ex:TaskTree	ex:ProjectCollectionTree	/data/projects/
-------- /task-48/	ex:TaskTree	ex:ProjectCollectionTree	/data/projects/
---------- /attachment-aa89	ex:AttachmentTree	ex:ProjectCollectionTree	/data/projects/
-------- /task-61/	ex:TaskTree	ex:ProjectCollectionTree	/data/projects/
-------- /issue-22/	ex:IssueTree	ex:ProjectCollectionTree	/data/projects/
---------- /attachment-cd12	ex:AttachmentTree	ex:ProjectCollectionTree	/data/projects/
---------- /attachment-ef55	ex:AttachmentTree	ex:ProjectCollectionTree	/data/projects/
-------- /issue-31/	ex:IssueTree	ex:ProjectCollectionTree	/data/projects/

***Figure 7** Shape Tree Manager for /data/projects/ with multiple shape tree assignments in a nested physical hierarchy*

```
PREFIX st: <http://www.w3.org/ns/shapetrees#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example/ns/ex#>

<>
  a st:Manager ;
  st:hasAssignment
    <#assignment1>,
    <#assignment2> .

<#assignment1>
  st:assigns ex:DataCollectionTree ;
  st:manages <https://storage.example/data/projects/>
  st:hasRootAssignment <https://storage.example/data/.shapetree#assignment1> ;
  st:focusNode <https://storage.example/data/projects/#collection> ;
  st:shape ex:DataCollectionShape .

<#assignment2>
  st:assigns ex:ProjectCollectionTree ;
  st:manages <https://storage.example/data/projects/>
  st:hasRootAssignment <https://storage.example/data/projects/.shapetree#assignment2> ;
  st:focusNode <https://storage.example/data/projects/#collection> ;
  st:shape ex:ProjectCollectionShape .
```

***Figure 8** Shape Tree Manager for /data/projects/project-1/milestone-A/task-48 with a single shape tree assignment in a nested physical hierarchy*

```
PREFIX st: <http://www.w3.org/ns/shapetrees#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example/ns/ex#>

<>
  a st:Manager ;
  st:hasAssignment <#assignment1> .

<#assignment1>
  st:assigns ex:TaskTree ;
  st:manages <https://storage.example/data/projects/project-1/milestone-A/task-48> ;
  st:hasRootAssignment <https://storage.example/data/projects/.shapetree#assignment2> ;
  st:focusNode <https://storage.example/data/projects/project-1/milestone-A/task-48#tas
  st:shape ex:ProjectCollectionShape .
```

§ 3.3. Shape Tree Manager Schema

***Figure 9** ShEx Schema for a Shape Tree Manager*

```
<#ManagerShape> {
  a [st:Manager] ;
  st:hasAssignment @<#AssignmentShape> +
}

<#AssignmentShape> {
  a [st:Assignment] ;
  st:assigns IRI ;
  st:manages IRI ;
  st:hasRootAssignment IRI ;
  ( st:focusNode IRI ;
    st:shape IRI ) ?
}
```

§ 4. Shape Tree Operations

Working with shape trees entails using several higher-level operations —each of which may represent one or more HTTP requests and/or pieces of processing logic.

In regular use, a client-side agent manipulates resources on a resource server running a server-side agent. That server-side agent applies logic for shape tree validation and navigation where applicable when processing requests from client-side agents.

The key operations used to manage shape trees are:

- Discover Shape Tree - determine the shape tree linked to some managed resource.
- Plant Shape Tree - declare that a resource will henceforth be managed by the provided shape tree.
- Unplant Shape Tree - assert that a managed resource should no longer be managed by the provided shape tree.
- Create Managed Instance - add a resource to a managed resource hierarchy.
- Update Managed Instance - modify a resource in a managed resource hierarchy.
- Delete Managed Instance - remove a resource in a managed resource hierarchy.

These operations make use of reusable, internal algorithms defined in Shape Tree Algorithms.

Shape tree logic can be applied by server-side agents implementing different

protocols such as [LDP] or [Solid]. The operations defined herein defer to the implementing protocol as to the appropriate status code and composition of HTTP responses.

> Note: Server-side processing of changes to shape tree managers can support the addition and removal of multiple shape tree assignments at once. For simplicity, this specification provides discreet client-side operations to plant or unplant one shape tree at a time. However, it would not be inappropriate to provide client implementations that support requests to plant and unplant multiple shape trees for a given resource in a single request.

§ 4.1. Discover Shape Tree

Description	
This operation is used by a client-side agent to discover any shape trees associated with a given resource. If `URI` is a managed resource, the associated Shape Tree Manager will be returned.	
Inputs	
`RESOURCEURI`	The URI of the resource to discover shape trees for
Outputs	
`MANAGER`	Shape tree manager associated with the managed resource

1. Perform an HTTP HEAD or GET on the provided **`RESOURCEURI`**.

```
HEAD https://storage.example/data/projects/
```

```
HTTP/1.1 200 OK
Link: <https://storage.example/meta/c560224b>; rel="http://www.w3.org/ns/shapetrees#man
Link: <http://www.w3.org/ns/ldp#Container>; rel="type"
...other HTTP response headers omitted...
```

2. Let **`MANAGERURI`** be the URI of a shape tree manager associated with **`RESOURCEURI`** with a Link relation type of **`http://www.w3.org/ns/shapetrees#managedBy`**.
3. If **`MANAGERURI`** is missing, the resource at **`RESOURCEURI`** is not a managed resource, and no shape tree manager will be returned.
4. Perform an HTTP GET on **`MANAGERURI`**

```
GET https://storage.example/data/projects/.shapetree


...HTTP response headers omitted...
PREFIX st: <http://www.w3.org/ns/shapetrees#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example/ns/ex#>

<>
  a st:Manager ;
  st:hasAssignment <#assignment1> .

<#assignment1>
  st:assigns ex:DataCollectionTree ;
  st:manages <https://storage.example/data/projects/>
  st:hasRootAssignment <https://storage.example/data/.shapetree#assignment1> ;
  st:focusNode <https://storage.example/data/projects/#collection> ;
  st:shape ex:DataCollectionShape .
```

5. If a corresponding resource at **`MANAGERURI`** is not found, it MUST be considered an unmanaged resource.

```
GET https://storage.example/data/projects/.shapetree


HTTP/1.1 404 Not Found
```

5. If **`MANAGERURI`** contains a valid shape tree manager, the resource at **`RESOURCEURI`** MUST be considered a managed resource.

§ 4.2. Plant Shape Tree

Description
This operation marks an existing resource as managed by one or more shape trees, by creating or updating an associated shape tree manager. If the resource is already managed, the associated shape tree manager will be updated with another shape tree assignment for the planted shape tree. If the resource is a container that already contains existing resources, this operation will perform a depth first traversal through the containment hierarchy, validating and assigning as it works its way back up to the target resource of this operation.

§ 4.2.1. Client-side

Inputs	
`TR`	The URI of the resource to plant on
`TST`	A URI representing the shape tree to plant for `TR`
`FN`	An OPTIONAL URI representing the subject within `TR` used as the focus node for shape validation
Outputs	
`RESPONSE`	A standard HTTP response

1. Discover if **`TR`** is a managed resource.
    - Let **`MANAGERURI`** be the URI of the Shape Tree Manager associated with **`TR`**.
2. Perform an HTTP **`PUT`** or **`PATCH`** on **`MANAGERURI`** to create or update the Shape Tree Manager for **`TR`**
    - Add a new **`st:Assignment ASN`**
    - Let **`ASN assigns`** be **`TST`**
    - Let **`ASN manages`** be **`TR`**
    - Let **`ASN hasRootAssignment`** be **`ASN`**
    - If **`TST`** has an **`st:shape`**
        - Let **`ASN st:shape`** be the object value of **`TST st:shape`**
        - Let **`ASN st:focusNode`** be **`FN`**

```
PUT https://storage.example/data/projects/.shapetree
```

```
PREFIX st: <http://www.w3.org/ns/shapetrees#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://www.example/ns/ex#>

<>
  a st:Manager ;
  st:hasAssignment <#assignment1>, <#assignment2> .

<#assignment1>
  st:assigns ex:DataCollectionTree ;
  st:manages <https://storage.example/data/projects/>
  st:hasRootAssignment <https://storage.example/data/.shapetree#assignment1> ;
  st:focusNode <https://storage.example/data/projects/#collection> ;
  st:shape ex:DataCollectionShape .
```

```
<#assignment2>
  st:assigns ex:ProjectCollectionTree ;
  st:manages <https://storage.example/data/projects/>
  st:hasRootAssignment <https://storage.example/data/projects/.shapetree#assignment2> ;
  st:focusNode <https://storage.example/data/projects/#collection> ;
  st:shape ex:ProjectCollectionShape .


HTTP/1.1 204 No Content
```

§ 4.2.2. Server-side

Inputs	
REQ	An HTTP PUT or PATCH request on MANAGERURI from the previous sequence
Outputs	
RESPONSE	A standard HTTP response

> Note: The following sequence is invoked by the server in response to the **HTTP PUT** or **HTTP PATCH** on **MANAGERURI** by the client-side agent in the previous sequence.

1. Let **MANAGER** be the shape tree manager target of **REQ**
2. Let **R** be the primary resource directly associated with **MANAGER**
3. Let **UMR** be the updated shape tree manager in the body of **REQ**
4. Let **EMR** be the existing shape tree manager resource on the server
5. Let **ADDED** be the set of shape tree assignments that have been added to **EMR** by **UMR**
6. Let **REMOVED** be the set of shape tree assignments that have been removed from **EMR** by **UMR**
7. If **REMOVED** is not empty, the server must unplant the shape tree assignments that have been removed.
8. For each shape tree assignment **ASN** in **ADDED**
    1. Call § 5.1 Assign Shape Tree to Resource with inputs: **UMR**, **ASN**, **ASN**, **R**, **NULL**

§ 4.3. Unplant Shape Tree

Description
This operation unassigns a planted shape tree from a given managed resource. If the managed resource is a managed container, it will also unassign contained resources. This operation will fail immediately if the shape tree to unplant is not the root shape tree assignment.

§ 4.3.1. Client-side

Inputs	
`TR`	The URI of the target managed resource to unplant
`TST`	A URI representing the target shape tree to unplant for `TR`
Outputs	
`RESPONSE`	A standard HTTP response

1. Let **`MANAGER`** be the Shape Tree Manager returned from § 4.1 Discover Shape Tree with inputs: **`TR`**
2. Return failure if **`MANAGER`** has zero or more than one shape tree assignments where **`st:assigns`** is **`TST`**
3. If **`MANAGER`** has a single shape tree assignment
    1. Perform an HTTP **`DELETE`** on **`MANAGER`** to fully remove the Shape Tree Manager for **`TR`**

```
DELETE https://storage.example/data/projects/.shapetree
```

```
HTTP/1.1 204 No Content
```

1. If **`MANAGER`** has more than one shape tree assignment
    1. Let **`ASN`** be the shape tree assignment where **`st:assigns`** is **`TST`**
    2. Perform an HTTP **`PUT`** or **`PATCH`** on **`MANAGER`** to remove **`ASN`**

§ 4.3.2. Server-side

Inputs	
`REQ`	An HTTP PUT, PATCH, or DELETE request on the shape tree manager `MANAGER` from the previous sequence.
Outputs	
`RESPONSE`	A standard HTTP response

> Note: The following sequence is invoked by the server in response to the **`HTTP PUT`**, **`HTTP PATCH`**, or **`HTTP DELETE`** on **`MANAGERURI`** by the client-side agent in the previous sequence.

1. Let **`MANAGER`** be the shape tree manager target of **`REQ`**
2. Let **`R`** be the primary resource directly associated with **`MANAGER`**
3. Let **`UMR`** be the updated shape tree manager in the body of **`REQ`**
4. Let **`EMR`** be the existing shape tree manager resource on the server
5. Let **`ADDED`** be the set of shape tree assignments that have been added to **`EMR`** by **`UMR`**
6. Let **`REMOVED`** be the set of shape tree assignments that have been removed from **`EMR`** by **`UMR`**
7. For each shape tree assignment **`ASN`** in **`REMOVED`**
8. Call § 5.2 Unassign Shape Tree from Resource with inputs: **`ASN`**, **`R`**
9. If **`ADDED`** is not empty, the server must plant the shape tree assignments that have been added.

§ 4.4. Create Managed Instance

Description
This operation creates a managed instance within a managed container. > Note: This operation can be performed as a standard HTTP operation with no knowledge of shape trees. However, server-side processing is more efficient when the target shape tree and focus node for validation can be provided by the client-side agent.

§ 4.4.1. Client-side

Inputs	
`TR`	The URI of the target resource
`TST`	An OPTIONAL URI representing the target shape tree associated with the created resource
`FN`	An OPTIONAL URI representing the target subject within `TR` used for shape validation
Outputs	
`RESPONSE`	A standard HTTP response

1. Perform an HTTP **`POST`**, **`PUT`**, or **`PATCH`** in or on **`TR`** to create the managed instance including:
    - An HTTP **`Link`** header with the relation of **`http://www.w3.org/ns/shapetrees#TargetShapeTree`** if **`SHAPETREEURI`** is provided
    - An HTTP **`Link`** header with the relation of **`http://www.w3.org/ns/shapetrees#FocusNode`** if **`FNURI`** is provided

§ 4.4.2. Server-side

Inputs	
`REQ`	A HTTP `POST`, `PUT`, or `PATCH` request in or on the target resurce `TR` from the previous sequence
Outputs	
`RESPONSE`	A standard HTTP response

1. Let **`TR`** be the proposed resource from **`REQ`**
2. Let **`TST`** be the value of an optionally provided HTTP **`Link`** header with the relation of **`http://www.w3.org/ns/shapetrees#TargetShapeTree`**
3. Let **`FN`** be the value of an optionally provided HTTP **`Link`** header with the relation of **`http://www.w3.org/ns/shapetrees#FocusNode`**
4. Let **`PC`** be the parent container for **`TR`**

5. Let **`CASN`** be the shape tree assignment whose shape tree **`CASNST`** manages the allowed members of **`PC`** via **`CASNST st:contains`**

    1. If **`CASN`** is not found, the request can be passed through, as the created resource **`TR`** will not be a managed resource

6. Call § 5.3 Validate Contained Resource with inputs: **`CASNST`**, **`TR`**, **`TST`**, **`FN`**

    1. Let **`AVR`** be the positive validation result returned

7. Create **`TR`**

8. Call § 5.1 Assign Shape Tree to Resource with inputs: **`NULL`**, **`CASN`** **`st:hasRootAssignment`**, **`CASN`**, **`TR`**, **`AVR`**

§ 4.5. Update Managed Instance

Description
This operation updates an existing managed resource. Note: This operation can be performed as a standard HTTP operation with no knowledge of shape trees. However, server-side processing is more efficient when the target shape tree and focus node for validation can be provided by the client-side agent.

§ 4.5.1. Client-side

Inputs	
TR	The URI of the target resource
Outputs	
RESPONSE	A standard HTTP response

1. Perform an HTTP **`PUT`** or **`PATCH`** on an existing resource **`TR`** to update the managed instance

§ 4.5.2. Server-side

Inputs	
`REQ`	A HTTP `PUT` or `PATCH` request on the managed resource from the previous sequence
Outputs	
`RESPONSE`	A standard HTTP response

1. Let **`TR`** be the target resource of **`REQ`**
2. Let **`UR`** be the updated version of **`TR`** in the body of **`REQ`**
3. Let **`MR`** be the shape tree manager associated with **`TR`**
4. If **`MR`** exists
    1. For each shape tree assignment **`ASN`** in **`MR`**
        1. Call § 5.4 Validate Resource with inputs: **`ASN st:assigns`**, **`UR`**, **`ASN st:focusNode`**
5. Update resource **`TR`** with **`UR`**

§ 4.6. Delete Managed Instance

Description
This operation deletes a managed resource for an existing managed instance. Note: This operation should be performed as a standard HTTP operation with no knowledge of shape trees. It is included here for completeness

§ 4.6.1. Client-side

Inputs	
`TR`	The URI of the target resource
Outputs	
`RESPONSE`	A standard HTTP response

1. Perform an HTTP **`DELETE`** on an existing resource **`TR`** to delete the managed instance

4.6.2. Server-side

Inputs	
`REQ`	An HTTP `DELETE` request on the managed resource from the previous sequence
Outputs	
`RESPONSE`	A standard HTTP response

1. Let **`TR`** be the target resource of **`REQ`**
2. Delete resource **`TR`**
    - The Shape Tree Manager associated with **`TR`** MUST be removed with **`TR`**

5. Shape Tree Algorithms

The following algorithms define a library of functions referenced in the above operations.

5.1. Assign Shape Tree to Resource

Description	
Assigns the target shape tree `TST` to the target resource resource `TR`.	
Inputs	
`RTMR`	The root shape tree manager of the planted hierarchy
`RTASN`	The root shape tree assignment of the planted hierarchy
`PASN`	The parent shape tree assignment of the primary resource `R`
`R`	The primary resource for assignment
`AVR`	An optional validation result indicating that that `R` has already passed validation in advance and does not need this algorithm to perform validation again.
Outputs	
`RESPONSE`	A regular HTTP Response

1. Let **`ATPLANTROOT`** be true if **`RTASN st:manages`** is **`R`**
2. If **`AVR`** is provided
    1. Let **`RST`** be the shape tree that **`R`** has been validated to conform to
    2. Let **`RFN`** be the focus node for shape validation by **`RST st:shape`**
3. If **`ATPLANTROOT`**
    1. Let **`RST`** be **`RTASN st:assigns`**
    2. If **`AVR`** was not provided call § 5.4 Validate Resource with inputs: **`RST`**, **`R`**, **`NULL`**
    3. Let **`RFN`** be the matching focus node provided in the validation result
4. If not **`ATPLANTROOT`** and **`AVR`** was not provided
    1. Call § 5.3 Validate Contained Resource with inputs: **`PASN st:assigns`**, **`R`**, **`RST`**, **`RFN`**
    2. Let **`RST`** be the matching shape tree provided in the validation result
    3. Let **`RFN`** be the matching focus node provided in the validation result
5. Let **`RMR`** be the shape tree manager associated with **`R`**
6. Let **`RASN`** be a new shape tree assignment created for **`R`** with the following properties:
    - Let **`st:assigns`** be **`RST`**

- Let **st:manages** be **R**
- Let **st:hasRootAssignment** be **RASN**
- Let **st:focusNode** be **RFN**
- Let **st:shape** be **RST st:shape**

7. If **R** is a non-empty container, let **CONTAINED** be the set of contained resources sorted by type, containers first.
8. For each contained resource **CR** in **CONTAINED**, starting with containers
    1. Call § 5.1 Assign Shape Tree to Resource with inputs: **RTMR**, **RTASN**, **RASN**, **R**, **NULL**
9. Create or Update the Shape Tree Manager **RMR**

§ 5.2. Unassign Shape Tree from Resource

Description	
Unassigns a shape tree managing resource `R` by removing the shape tree assignment assigned to `R` for that shape tree.	
Inputs	
`RTASN`	The root shape tree assignment at the top of the planted hierarchy.
`R`	The resource to unplant in the plant hierarchy of `RTASN`
Outputs	
`RESPONSE`	A regular HTTP response

1. Let **RMR** be the shape tree manager associated with **R**
2. Let **RASN** be the shape tree assignment to remove for **R** where **RTASN** is equivalent to **RASN st:hasRootAssignment**
3. Let **RASNST** be the shape tree **RASN st:assigns**
4. If **R** is a non-empty container, let **CONTAINED** be the set of contained resources sorted by type, containers first.
5. For each contained resource **CR** in **CONTAINED**, starting with containers
    1. Call § 5.2 Unassign Shape Tree from Resource with inputs: **RTASN**, **CR**
6. Update or delete the Shape Tree Manager **RMR**

§ 5.3. Validate Contained Resource

Description	
This algorithm is responsible for determining which shape tree within a set of shape trees mentioned in `st:contains` is applicable for a given proposed resource.	
Inputs	
`ST`	The validating shape tree
`R`	The resource to be evaluated against the permitted set of contained shape trees in `ST st:contains`
`TST`	An OPTIONAL URI that provides the algorithm with a target shape tree that `R` is expected to conform to. No other shape trees from `ST st:contains` are considered when provided.
`FN`	An OPTIONAL URI representing the target subject node within `R` used for shape validation.
Outputs	
`VR`	A validation result containing: • Valid (true or false) • Validating shape tree `ST` • Matching shape tree in `ST st:contains` • Matching shape (if applicable) • Matching focus node (if applicable)

1. If **`TST st:contains`** is empty, return a true validation result
2. If **`TST`** is provided but does not exist in **`ST st:contains`** return a false validation result
3. If **`TST`** is provided call § 5.4 Validate Resource with inputs: **`TST`**, **`R`**, **`FN`**
4. If **`TST`** is not provided then for each shape tree **`CST`** linked via **`ST st:contains`**
    1. Call § 5.4 Validate Resource with inputs: **`CST`**, **`R`**, **`FN`**

§ 5.4. Validate Resource

Description	
This algorithm is responsible for determining whether a given resource conforms with a shape tree	
Inputs	
`ST`	The shape tree that `R` will be evaluated against
`R`	The resource to evaluate for conformance to `ST`
`FN`	An OPTIONAL focus node to use for shape validation when `ST st:shape` is set
Outputs	
`VR`	A validation result containing: • Valid (true or false) • Validating shape tree `ST` • Matching shape (if applicable) • Matching focus node (if applicable)

1. Return a failing validation result if **`ST st:expectedType`** is set and is not the resource type of **`R`**
2. Return a failing validation result if **`ST rdfs:label`** is set and is not equal to the resource name of **`R`**
3. Return a failing validation result if **`ST st:shape`** is set and shape validation of the body content of **`R`** fails
4. Return a positive validation result

§ 6. Describing Shape Trees

§ 6.1. Shape Tree Description

While the RDF structure of shape trees enable machine readability, additional context is needed to make it human-friendly. A ***Shape Tree Description*** provides a human-readable information about a given shape tree.

Shape tree descriptions are organized into shape tree description sets.

Figure 10 *Shape Tree Description properties*

Property	Description
st:describes	Identifies the shape tree being described
st:inDescriptionSet	Identifies the shape tree description set that the description belongs to
skos:prefLabel	Provides a human readable name for the shape tree
skos:definition	Provides a more in-depth, human readable description of the shape tree
st:describesInstance	Identifies a predicate whose object value can be used to describe a managed instance of the described shape tree. The object value must be a literal.
skos:narrower	Identify a poly-hierarchy link between two shape tree descriptions in a shape tree description set
skos:broader	Identify a poly-hierarchy link between two shape tree descriptions in a shape tree description set

Figure 11 *ShEx Schema for Shape Tree Description*

```
<#DescriptionShape> {
  a [st:Description] ;
  st:inDescriptionSet @<#DescriptionSetShape> ;
  st:describes IRI ;
  st:describesInstance IRI ? ;
  skos:prefLabel xsd:string ;
  skos:definition xsd:string ? ;
  skos:narrower IRI ? ;
  skos:broader IRI? ;
}
```

§ 6.2. Shape Tree Description Set

Shape tree descriptions are organized into a ***Shape Tree Description Set***.

An RDF resource containing one or more shape trees can be OPTIONALLY linked to a shape tree description set to describe the contained shape trees in human-

readable terms.

Each shape tree description set is made up of any number of shape tree descriptions.

SKOS constructs such as `skos:narrower` and `skos:broader` MAY be used to group or organize related shape trees in a given shape tree description set

Figure 12 *Shape Tree Description Set properties*

Property	Description
`st:usesLanguage`	Identifies the `xsd:language` utilized in the contained shape tree descriptions

Figure 13 *ShEx Schema for Shape Tree Description Set*

```
<#DescriptionSetShape> {
  a [st:DescriptionSet] ;
  st:usesLanguage xsd:language
}
```

§ 7. Definitions

Two terms are imported from [RDF]:

- ***triple*** -- an RDF triple
- ***RDF graph*** -- an RDF graph as defined in [RDF].

The following terms are used throughout this specification:

- ***Client-side Agent*** -- A software component interacting with a server. A client-side agent will typically rely on the server for shape tree evaluation and/or validation, but may choose to apply it locally as well.
- ***Container*** -- the generalized notion of a collection of resources; implementations of shape trees MAY use a container implementation such as [LDP] (ldp:Container, ldp:BasicContainer, etc.)
- ***Ecosystem*** -- a software environment with resources organized in some hierarchical grouping that rely on shape tree concepts to better organize and validate structures of data

- ***Focus Node*** -- a node in an RDF graph. In the context of shape trees, one usage is directing shape validation to the appropriate node in an RDF graph.
- ***Managed Container*** -- any managed resource that is a container. A Managed Container MAY be an Instance Root or hierarchically nested within the resource hierarchy.
- ***Non-RDF Source*** -- the generalized notion of document not containing linked-data triples; this may include plain text or binary data.
- ***Resource*** -- the generalized notion of document containing linked-data; implementations of shape trees may use a resource implementation such as [LDP] (ldp:Resource, etc.)
- ***Server-side Agent*** -- A server-side software component. Server-side agents that support shape trees are responsible for data validation and maintaining shape tree managers.
- ***Shape*** -- a schema definition allowing validation of an RDF subject. Example specifications supporting the notion of shapes include [ShEx] and [SHACL].
- ***SKOS Graph*** -- an RDF graph conforming to [skos-reference] data model. For purposes of shape trees a SKOS Graph is used to describe a shape tree in human-readable terms.
- ***Unmanaged Resource*** -- any resource which does not have an associated shape tree.

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

EXAMPLE 1

This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

§ Conformant Algorithms

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant. Implementers are encouraged to optimize.

§ Index

§ Terms defined by this specification

§ References

§ Normative References

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://datatracker.ietf.org/doc/html/rfc2119

§ Informative References

[LDP]
Steve Speicher; John Arwe; Ashok Malhotra. *Linked Data Platform 1.0*. URL: https://www.w3.org/TR/ldp/

[RDF]
Richard Cyganiak; David Wood; Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. URL: https://www.w3.org/TR/rdf11-concepts

[SHACL]
Holger Knublauch; Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. 20 July 2017. REC. URL: https://www.w3.org/TR/shacl/

[ShEx]
Eric Prud'hommeaux; et al. *Shape Expressions Language 2.1*. URL: http://shex.io/shex-semantics/index.html

[SHEXPATH]
Eric Prud'hommeaux. *Shape Expressions ShExPath Language*. URL: https://shexspec.github.io/spec/ShExPath

[SKOS-REFERENCE]
Alistair Miles; Sean Bechhofer. *SKOS Simple Knowledge Organization System Reference*. 18 August 2009. REC. URL: https://www.w3.org/TR/skos-reference/

[Solid]
Sarven Capasdisli; et al. *Solid Protocol*. URL: https://solidproject.org/TR/protocol