Standards Track Work Product

OASIS OPEN PROJECTS

OSLC Automation Version 2.1 Part 1: Specification

Project Specification Draft 01
21 January 2021

This stage:

https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/psd01/automation-spec.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/psd01/automation-spec.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/automation-spec.html (Authoritative)
https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/automation-spec.pdf

Latest version:

https://open-services.net/spec/auto/latest

Latest editor's draft:

https://open-services.net/spec/auto/latest-draft

Open Project:

OASIS Open Services for Lifecycle Collaboration (OSLC) OP

Project Chairs:

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

Editors:

Jim Amsden (jamsden@us.ibm.com), IBM
Fabio Ribeiro (fabio.ribeiro@koneksys.com), Koneksys

Additional components:

This specification is one component of a Work Product that also includes:

- *OSLC Automation Version 2.1 Part 1: Specification (this document).* automation-spec.html
- *OSLC Automation Version 2.1 Part 2: Vocabulary.* automation-vocab.html
- *OSLC Automation Version 2.1 Part 3: Constraints.* automation-shapes.html
- *OSLC Automation Version 2.1 Machine Readable Vocabulary Terms.* automation-vocab.ttl
- *OSLC Automation Version 2.1 Machine Readable Vocabulary Constraints.* automation-shapes.ttl

Related work:

This specification is related to:

- *Open Services for Lifecycle Collaboration Automation Specification Version 2.1.* http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/

RDF Namespaces:

http://open-services.net/ns/auto#

Abstract:

This specification defines the OSLC Automation domain, a RESTful web services interface for the management of OSLC resources. To support these scenarios, this specification defines a set of HTTP-based RESTful interfaces in terms of HTTP methods: GET, POST, PUT and DELETE, HTTP response codes, content type handling and resource formats.

Status:

This document was last revised or approved by the OASIS Open Services for Lifecycle Collaboration (OSLC) OP 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://github.com/oslc-op/oslc-specs.

Standards Track Work Product

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.

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:
[OSLC-AUTO-v2.1-Part1]
OSLC Automation Version 2.1 Part 1: Specification. Edited by Jim Amsden and Fabio Ribeiro. 21 January 2021. OASIS Project Specification Draft 01. https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/psd01/automation-spec.html. Latest stage: https://docs.oasis-open-projects.org/oslc-op/auto/v2.1/automation-spec.html.

Notices

Table of Contents

1. Introduction

This section is non-normative.

This section is non-normative, i.e. it does not affect compliance. This specification builds on [OSLCCore3] to define the resources and operations supported by an Open Services for Lifecycle Collaboration (OSLC) Automation provider. Automation resources define automation plans, automation requests and automation results of the software development, test, deployment, and operations lifecycle. They represent individual resources as well as their relationships to other automation resources and to other linked resources outside of the automation domain. The intent of this specification is to define the set of HTTP-based RESTful interfaces in terms of HTTP methods: GET, POST, PUT and DELETE, HTTP response codes, MIME type handling and resource formats. The capabilities of the interface definitions are driven by key integration scenarios and therefore don't represent a complete setup of operations on resources or resource types. The resource formats and operations may not match exactly the native models supported by automation service providers but are intended to be compatible with them. Automation, as referenced in this specification, refers to the use of IT systems such as servers, workstations and smart hand-held devices to improve efficiency and reduce the need for manual human interactions in the software development, test, deployment, and operations lifecycle. See the Automation Scenarios page for examples from the build, test, deployment, and operations disciplines.

1.1 Terminology

This section is non-normative.

Service Provider

an implementation of the OSLC Automation specification as a server. OSLC Automation clients consume these services.

Automation Resource

A resource managed by the Automation service provider. The types of resources defined by this specification are Automation Plan, Automation Request and Automation Result.

Automation Plan

Defines the unit of automation which is available for execution.

Automation Request

Defines the submission of the information required to execute an Automation Plan and indicates the desired execution state.

Automation Result

Defines intermediate and final execution status of an Automation Request, along with contributions to the result.

Automation Parameter Definition

Defines an individual input parameter of an Automation Plan. Parameter Definitions provide an indication of the type of the parameter and range of allowed values.

Automation Parameter Instance

Defines an individual input or output parameter instance for an Automation Request or Result.

1.2 References

1.2.1 Normative references

[OSLCActions]
Martin Pain; Samuel Padget. *OSLC Actions*. Draft. URL: https://raw.githack.com/oslc-op/oslc-specs/master/specs/core/actions.html

[OSLCAutomationSamples]
Reference not found.

[OSLCAutomationScenarios]
Reference not found.

[OSLCAutomationTemporaryDeploymentScenarios]
Reference not found.

[OSLCCore3]
Steve Speicher; Jim Amsden. *OSLC Core Overview v3.0*. Project Specification. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.html

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

[RFC8174]
B. Leiba. *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/rfc8174

1.3 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**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this specification are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Base Requirements

2.1 Specification Versioning

See [OSLCCore3] Specification Versioning section.

2.2 Namespaces

In addition to the namespace URIs and namespace prefixes defined in the [OSLCCore3], OSLC Automation defines the namespace URI of `http://open-services.net/ns/auto#` with a namespace prefix of `oslc_auto`. This namespace URI and prefix are used to designate the resources defined in this specification and their properties.

2.3 Resource Formats

In addition to the requirements for [OSLCCore3] Defined Resource Representations, this section outlines further refinements and restrictions.

See HTTP Method support table for further clarification on support for HTTP methods and media types for each OSLC Automation resource.

For HTTP GET requests on all OSLC Automation and OSLC Core defined resource types,

- Automation Providers **MUST** provide RDF/XML representations. The RDF/XML representation **SHOULD** follow the guidelines outlined in the [OSLCCore3] for RDF/XML. [auto-1]
- Automation Providers **MAY** provide XML and JSON representations. If provided, the XML and JSON representations **SHOULD** follow the guidelines outlined in the [OSLCCore3].
- Automation Consumers requesting RDF/XML **SHOULD** be prepared for any valid RDF/XML document. Automation Consumers requesting XML **SHOULD** be prepared for representations that follow the guidelines outlined in the [OSLCCore3].
- Automation Providers **SHOULD** support an [X]HTML representation and a user interface (UI) preview as defined by [OSLCCore3].

For HTTP PUT/POST request formats for Automation resources,

- Automation Providers **MUST** accept RDF/XML representations and **MAY** accept XML representations. Automation Providers accepting RDF/XML **SHOULD** be prepared for any valid RDF/XML document. If XML is accepted, Automation Providers **SHOULD** be prepared for representations that follow the guidelines outlined in the [OSLCCore3]. [auto-2]
- Automation Providers **MAY** accept XML and JSON representations. Automation Providers accepting XML or JSON **SHOULD** be prepared for representations that follow the guidelines outlined in the [OSLCCore3]. [auto-3]

For HTTP GET response formats for Query requests,

Automation Providers **MUST** provide RDF/XML and **MAY** provide JSON, XML, and Atom Syndication Format XML. [auto-4]

When Automation Consumers request:

- `application/rdf+xml` Automation Providers **MUST** respond with RDF/XML representation without restrictions. [auto-5]
- `application/xml` Automation Providers **SHOULD** respond with OSLC-defined abbreviated XML representation as defined in the [OSLCCore3].
- `application/atom+xml` Automation Providers **SHOULD** respond with Atom Syndication Format XML representation as defined in the [OSLCCore3].
- If supported, the Atom Syndication Format XML representation **SHOULD** use RDF/XML representation without restrictions for the atom:content entries representing the resource representations.

2.4 Authentication

See [OSLCCore3] Authentication section. OSLC Automation puts no additional constraints on authentication.

2.5 Error Responses

See [OSLCCore3] Error Responses section. OSLC Automation puts no additional constraints on error responses.

2.6 Pagination

OSLC Automation service providers **SHOULD** support pagination of query results and **MAY** support pagination of a single resource's properties as defined by the OSLC Core Specification.

2.7 Labels for Relationships

This section is non-normative.

Automation relationships to other resources are represented as properties whose values are the URI of the object or target resource. When an Automation relationship property is to be presented in a user interface, it may be helpful to provide an informative and useful textual label for that relationship instance. (This in addition to the relationship property URI and the object resource URI, which are also candidates for presentation to a user.) To this end, OSLC providers **MAY** support a `dcterms:title` link property in Automation resource representations, using the anchor approach outlined in the [OSLCCore3].

RDF/XML and XML example using reified statement:

Example 1

```
<rdf:RDF
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslc_auto="http://open-services.net/ns/auto#">
```



```
    <oslc_auto:AutomationResult rdf:about="http://example.com/results/4321">
        <oslc_auto:reportsOnAutomationPlan rdf:ID="link1"
            rdf:resource="http://example.com/plans/123" />
    </oslc_auto:AutomationResult>

    <rdf:Description rdf:about="#link1">
        <dcterms:title>Build Definition 123: Pet Shop App production build</dcterms:title>
    </rdf:Description>
</rdf:RDF>
```


3. Vocabulary Terms and Constraints

OSLC Automation Version 2.1 Part 2: Vocabulary > Defines the vocabulary terms for OSLC Automation resources. These terms and constraints are specified according to [OSLCCore3].

4. Automation Service Provider Capabilities

4.1 Asynchronous and Synchronous Automation Execution

An OSLC Automation service provider is generally assumed to implement automation requests asynchronously. In this model, a client creates an automation request and then later queries a collection of automation results for the particular result(s) related to its request. For generality, it is also assumed that results may be contributed asynchronously by a set of distributed processes, where each contributor adds its contribution(s) to the result via HTTP PUT. When a provider creates an automation request, it can also include an automation result, which might or might not yet be finished at the point in time when the provider responds to the create request. Providers can persist automation results for as long as they deem reasonable. Consumers are assumed to poll for updates to automation results until they have finished. Once a request has finished, the provider may remove it at any time. An automation result is "finished" when it has an `oslc_auto:state` predicate with an object of `oslc_auto:complete` or `oslc_auto:canceled` or an `oslc_auto:verdict` property with a value other than `oslc_auto:unavailable`.

4.2 Automation Provider Sub-Domains

An instance of an OSLC Automation service provider might provide services for one or more particular automation sub-domains (e.g. test or build automation). Automation service providers **MAY** declare sub-domain information in the Service Provider document by specifying a sub-domain value in the `oslc:usage` attribute on the `oslc:Service` resource in the Service Provider document. Valid sub-domain values are:

- **http://open-services.net/ns/auto#Build**: Indicates that the related service provider or services provide build automation capabilities - the process of converting source code artifacts into software artifacts such as executables, libraries and documentation.
- **http://open-services.net/ns/auto#Test**: Indicates that the related service provider or services provide test automation capabilities - the process of executing tests on a system under test and comparing the results of the tests to pass/fail conditions.
- **http://open-services.net/ns/auto#Deploy**: Indicates that the related service provider or services provide deployment capabilities - the process of executing processes and procedures to ready systems and software for use.

An automation service provider which is a general-purpose automation provider, or a provider not wanting to provide a sub-domain should provide an `oslc:usage` value of **http://open-services.net/ns/auto**. If no `oslc:usage` attribute indicating a sub-domain is present, the default is assumed to be **http://open-services.net/ns/auto**.

4.2.1 Sub-domain Example

Example of a service provider document fragment with a 2 Services which are identified as related to the Test and Deploy sub-domains:

```
<oslc:serviceProvider>
     <oslc:ServiceProvider>
          <oslc:service>
               <oslc:Service>
                    <oslc:usage rdf:resource="http://open-services.net/ns/auto#Test">
                    <oslc:queryCapability>
                        ...
                    </oslc:queryCapability>
                    <oslc:creationFactory>
                        ...
                    </oslc:creationFactory>
                        ...
               </oslc:Service>
          </oslc:service>
          <oslc:service>
               <oslc:Service>
                    <oslc:usage rdf:resource="http://open-services.net/ns/auto#Deploy">
                    ...
               </oslc:Service>
          </oslc:service>
     </oslc:ServiceProvider>
</oslc:serviceProvider>
```

4.3 Resource Shapes

OSLC Automation service providers **MAY** support [OSLCCore3] Resource Shapes as defined in [OSLCCore3]

4.4 Service Provider Resource

OSLC Automation service providers **MUST** provide a [OSLCCore3] Service Provider Resource that can be retrieved at a implementation dependent URI. [auto-6]

OSLC Automation service providers **MAY** provide a [OSLCCore3] Service Provider Catalog Resource that can be retrieved at a implementation dependent URI.

It is **RECOMMENDED** that OSLC Automation service providers provide a `oslc:serviceProvider` property for their defined resources that will be the URI to a [OSLCCore3] Service Provider Resource.

4.5 Creation Factories

If an OSLC Automation service provider supports the creation of resources, there **MUST** be at least one [OSLCCore3] Creation Factories entry in the Services definition. [auto-7]

See HTTP Method support table for further clarification on support for HTTP methods and media types for each OSLC Automation resource.

4.6 Query Capabilities

OSLC Automation service providers **SHOULD** have at least one Query Capabilities entry in the its Services definition that allows a client to query !AutomationResults.

Note: OSLC Automation does not require providers to keep resources accessible forever. Clients should not expect automation results to be available for any particular length of

time once the request has finished. Some providers might respond to an AutomationRequest creation request with an AutomationRequest that is also an AutomationResult, and might make the result inaccessible immediately thereafter.

Note: If an OSLC Automation provider does expose a Query Capability that applies to AutomationResults, and if its AutomationRequest creation responses are not also AutomationResults, then its Query Capability is the only Automation-defined way for clients to find the corresponding AutomationResults.

The Query Capability **MUST** support these OSLC query parameters and **MAY** support others: [auto-8]

- `oslc:where`
- `oslc:select`

If shape information is NOT present with the Query Capability, service providers **SHOULD** use the default properties defined in [OSLCCore3] to contain the result.

4.6.1 Selective Property Values

OSLC Automation providers **SHOULD** support the `oslc.properties` syntax for selective property value retrieval when a resource is accessible via its resource URI.

4.7 Delegated UIs

OSLC Automation service providers support the selection and creation of Automation resources as defined by [OSLCCore3] Delegated UIs in OSLC Core.

The service providers supports requirements for delegated UIs is as follows:

Automation Resource	Selection	Creation
AutomationPlan	**SHOULD**	**MAY**
AutomationRequest	**MAY**	**SHOULD**
AutomationResult	**SHOULD**	**MAY**

This is new for 2.1: START

4.7.1 Immediate-Execution Creation Dialog

An "immediate-execution" creation dialog is one that creates an Automation Request and makes it eligible for execution as soon as it is created. This is the only form of creation dialog that was defined in OSLC Automation 2.0. OSLC Automation 2.1 defines the term "immediate-execution creation dialog" and the `oslc:usage` URI `http://open-services.net/ns/auto#ImmediateExecution` (`oslc_auto:ImmediateExecution`) to distinguish them from deferred-execution creation dialogs.

OSLC Automation 2.1 consumers **MUST** interpret an `oslc_auto:AutomationRequest` creation dialog that has neither `oslc:usage oslc_auto:ImmediateExecution` set nor `oslc:usage oslc_auto:DeferredExecution` set as being an "immediate-execution creation dialog". This is to maintain compatibility with OSLC Automation 2.0 providers. [auto-9]

If an OSLC Automation provider offers both immediate-execution and deferred-execution creation dialogs, it **MUST** provide `oslc_auto:ImmediateExecution` or `oslc_auto:DeferredExecution` as a `oslc:usage` value (respectively) on the `oslc:Dialog` creation dialog resources. Such a provider **SHOULD** also set `oslc:default` as a `oslc:usage` value on the immediate-execution dialogs, to guide OSLC Automation 2.0 consumers to use those dialogs and not the deferred-execution ones. [auto-10]

4.7.2 Deferred-Execution Creation Dialog

A Deferred-Execution Creation Dialog is a [OSLCCore3] resource creation delegated user interface dialog that creates an Automation Request but **does not** make it eligible for execution. A deferred-execution creation dialog **MUST** comply with all Core requirements on [OSLCCore3] resource creation delegated user interface dialogs. One important consequence of this is that all facilities available on resource creation delegated user interface dialogs, for example [OSLCCore3] pre-filling, apply equally to deferred-execution creation dialogs. [auto-11]

> Non-normative note: The Automation 2.0 specification only provided a standard way to create Automation Requests that are eligible for execution once they are created; Automation 2.1 adds deferred-execution creation dialogs to allow creation without execution eligibility in a standard way. This meets [OSLCAutomationScenarios] template scenarios, while retaining compatibility with Automation 2.0 clients by keeping the behavior of `oslc:creationDialog` resources unchanged from 2.0.

This specification defines the `oslc:usage` URI `http://open-services.net/ns/auto#DeferredExecution` (`oslc_auto:DeferredExecution`) to allow clients to discover deferred-execution creation dialogs that an Automation provider supplies, amongst any other dialogs in their `oslc:Service` resources, as shown in [OSLCAutomationSamples] immediate and deferred dialog provider. The corresponding resource shape is shown in an earlier section.

One anticipated usage of deferred execution dialogs is to create `AutomationRequest`s for later scheduling: a template `AutomationRequest` is created (but never actually executed) during a configuration phase, a copy is saved by the client, and then the copy is used at future point(s) in time as pre-fill input to a standard 2.0 Automation Request creation factory or dialog. "Template" in this context refers to the client's usage of the `AutomationRequest` resource, rather than implying anything about its content (see also the section in [OSLCActions]). Clients typically store templates as opaque representations; this specification does nothing to force or discourage any particular behavior.

Any `AutomationRequest` created by a deferred-execution creation dialog is **especially likely** to be short-lived (cleaned up by the server shortly after creation); while this can be true of resources in general, for historical reasons (the 2.0 creation factory behavior described above) it is particularly important in this case as a common usage pattern. As a consequence, the consumer **SHOULD** get its representation immediately after creating it.

> Non-normative note: we suggest that providers allow these resources to exist for at least 15 minutes, but the actual value used is implementation-dependent.

4.7.2.1 Executing a previously constructed Automation Request

When a deferred-execution creation dialog creates an Automation Request, it is not queued for execution unless the client takes some explicit further action; it is the responsibility of the consumer to decide when (if ever) it is ready to be executed. OSLC defines options to initiate execution that include the following:

- Provide it as input to a standard (immediately execution-eligible) Automation [OSLCCore3] creation factory.
- Provide it as input to a standard (immediately execution-eligible) Automation creation dialog that supports pre-fill.

Note: assuming that the request is successful, it is important to recognize that the cases above all result in the creation of a **new** Automation Request, with a **different URI** than anything provided as an input. The provider may provide other ways, in addition to or in place of these, for the consumer to use when it is ready to have the Automation Request executed. OSLC currently has no scenarios requiring the definition of a way to change the state to make **the same (input)** request eligible for execution.

OSLC defines options for locating those immediate-execution resources, for example creation factories and delegated creation dialogs, that include the following:

- Consumers can examine an OSLC Service Provider document's `oslc:Service` resources. In many scenarios, Automation clients will only need to implement the Creation Factory interaction pattern to initiate execution, although other possibilities exist.
- Consumers can use `oslc:binding` properties on the `oslc_auto:DeferredExecution` dialog resource to simplify the process of locating appropriate immediate-execution resources. Those consumers choose at least one Actions specification profile, and implement the interaction patterns described in that profile. They are only able to consume deferred-execution dialogs whose bindings use an interaction pattern that the consumer implements.

The Automation provider **MUST** describe how to immediately execute an Automation Request created by a deferred-execution dialog using one or more `oslc:binding` properties on the `oslc_auto:DeferredExecution` dialog resource. If the deferred-execution dialog is discoverable from a Service in a [OSLCCore3] Service Provider, then the provider **MUST** supply at least one immediate-execution binding whose target uses the Automation creation factory interaction pattern. If multiple `oslc:binding` properties are present, they **MUST** be equivalent alternatives to each other, as defined by Core Actions. [auto-12]

When the second class of consumer from the list above is ready to execute an Automation Request acting as a template, it uses one of the `oslc:binding` properties on the deferred-execution dialog to immediately execute the action (often, by creating a new Automation Request with a different URI). The consumer does this by following the selected [OSLCActions] binding's instructions; its interaction pattern might be defined by this specification, or might be defined by another specification. A consumer chooses which `oslc:binding` value to use based on which interaction patterns it understands. If there are no `oslc:binding` values whose interaction patterns are understood by the consumer then the Automation Request acting as a template cannot be used by this consumer and the consumer **SHOULD** indicate this to the user instead of allowing them to use the deferred-execution dialog. A [OSLCAutomationSamples] deferred execution binding1 full example is available in the companion Samples document.

4.8 Execution environments

An AutomationPlan can use the `oslc_auto:usesExecutionEnvironment` predicate to link to a resource representing the environment(s) which that Automation Plan can be executed in. The execution environment resource could represent a grouping of environmental details such as operating system, database, browser, compiler, etc. The type of that resource, and the predicates to use on it, are not defined by this specification.

If more than one execution environment is specified on the Automation Plan, the consumer has to specify the desired execution environment as part of the Automation Request which it is constructing for the Automation Plan's execution. The execution environment is provided as an InputParameter to the Automation Request.

The consumer is expected to find a parameter definition from the Automation Plan with its `oslc:propertyDefinition` property set to `http://open-services.net/ns/auto#executionEnvironment`, and to create an InputParameter on the Automation Request for that parameter definition, specifying the execution environment to use (choosing out of those specified on the Automation Plan). If that parameter definition's `oslc:occurs` property is exactly-one or one-or-more, then the consumer **MUST** specify an execution environment, Otherwise, the consumer **MAY** specify an execution environment. [auto-13]

This is new for 2.1: END

4.9 State and Verdict properties

OSLC Automation service providers can identify the state and verdict using references to property values in the OSLC Automation vocabulary or to property values that are not in the Automation vocabulary (i.e. in the service provider's own vocabulary). It is expected that the state and verdict values will be URI references to property values, but inline resources defining the state and verdict property values are also valid. Automation service providers **MUST** use at least one verdict (Automation Results) and state (Automation Requests and Results) defined in the OSLC automation vocabulary in addition to any states or verdicts not in the Automation vocabulary. [auto-14]

The additional property values for `oslc_auto:state` are:

- `http://open-services.net/ns/auto#new` - used to indicate an automation request or result has just been created in the service provider and has not yet been acted upon.
- `http://open-services.net/ns/auto#queued` - primarily used to indicate an automation request or result is queued for additional actions by the service provider
- `http://open-services.net/ns/auto#inProgress` - used to indicate an automation request or result is active in the service provider.
- `http://open-services.net/ns/auto#canceling` - used to indicate the service provider is in the process of canceling an automation request or result.
- `http://open-services.net/ns/auto#canceled` - used to indicate that an automation request or result has been canceled.
- `http://open-services.net/ns/auto#complete` - used to indicate that an automation request or result is complete.

The additional property values for `oslc_auto:verdict` are:

- `http://open-services.net/ns/auto#unavailable` - used to indicate an automation result is in a state where a final verdict such as `oslc_auto:passed` or `oslc_auto:failed` is not yet available. Usually used when the result is in a state other than `oslc_auto:complete`.
- `http://open-services.net/ns/auto#passed` - used to indicate an automation result represents a successful execution.
- `http://open-services.net/ns/auto#warning` - used to indicate an automation result represents an execution which encountered conditions which prevented successful execution but did not result in a failed execution.
- `http://open-services.net/ns/auto#failed` - used to indicate an automation result represents a failed execution.
- `http://open-services.net/ns/auto#error` - used to indicate an automation result has completed but did not run successfully due to some error. This could be a timeout, automation coding error, network problem or other error which prevented the automation from running successfully to a passed, warning or failed verdict as described above.

This is new for 2.1: START

5. OSLC Actions and Automation

This specification defines extensions to the OSLC Actions 2.0 specification. Actions provide “a means of advertising actions (or operations) that can be performed on (or in the context of) a specific resource”. This relates to Automation in two ways: firstly, Automation Requests can be used as an *interaction pattern* by which actions can be executed, and secondly, Actions can provide a way to aid management and the lifecycle of automation resources.

The Actions specification reuses Automation resources to define an Automation Request interaction pattern, which can be used to execute actions. Actions also defines a *specification profile* that implementations can use, which provides interoperability based on providers and consumers both using a common interaction pattern. This specification extends the Actions specification by defining interaction patterns which are useful in the management of automation resources.

See also: Deciding how to use Actions and Automation together

5.1 Discovering actions and choosing bindings

5.1.1 Discovering executable actions and choosing bindings

For information on how to discover currently-available actions on resources and how to choose which binding to use for execution, see the [OSLCActions].

5.1.2 Discovering actions that will be executable after an Automation Request completes

One Automation use of Core’s actions is to advertise actions that become available after an Automation Request completes: for example, tearing down a deployed system, promoting or deleting a build. If the execution of the Automation Request resulted in a new resource being created (e.g. a resource representing the deployed system, or a resource representing the build) then it is expected that newly created resource would be linked to as an `oslc_auto:contribution` on the Automation Result, and any action in the context of that new resource would be linked to as an `oslc:action` on that resource. However, consumers may not know which contributions to check for action, so any actions that would make sense to follow up the execution of an Automation Request - whether immediately or at a later time - **SHOULD** be advertised on the Automation Result in addition to (or instead of) on a contribution.

5.1.2.1 Future actions

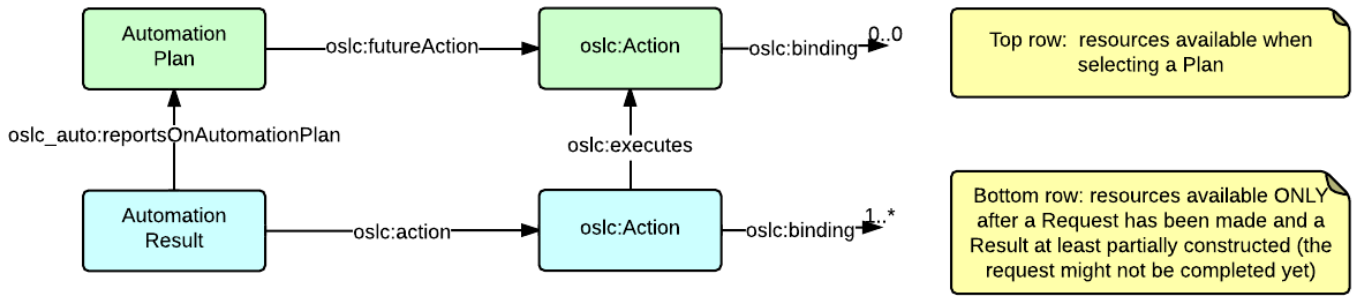
It is sometimes useful to know what actions will be available before an Automation Request is created (for example, for scheduling automated processes that will execute in their entirety without user intervention). Users might prefer such Automation Plans over otherwise equivalent ones that lack the ability to automate cleanup after themselves, so in fact it can be useful to know about future actions when selecting a Plan, before any Automation Request has even been created. Note: [OSLCCore3] mentions other uses of future actions.

To enable cases like these, providers **MAY** link to *future actions* using the `oslc:futureAction` predicate. When an Automation provider provides future action link(s) on an Automation Plan, they **SHOULD** link to resources of type `oslc:Action` which describe an action that may be executed after an execution of the Automation Plan has completed. As such, these `oslc:Action` resources **SHOULD NOT** contain any bindings that can be executed immediately.

> Non-normative note: Bindings using the “deferred execution dialog interaction pattern” may be present, but this specification does not define how to use them for future actions. It would not make sense to specify a deferred execution dialog execution binding for a future action, because its mandatory immediate-execution binding cannot become available until after an Automation Request has been created.

These future action resources describe what kind of actions are available on the Automation Result, so consumers can present these to users in preparation for when the execution has completed, and so the `oslc:Action` resources **SHOULD** include all the properties needed to render a display of the action. These `oslc:Action` resources **SHOULD NOT** be anonymous (RDF blank) nodes, so they can be linked to by the executable actions on the results using the predicate below.

The execution of these future actions requires an immediately executable action on an Automation Result. When an Automation Plan containing future actions is executed, each action applicable to the generated result **SHOULD** have an equivalent immediately executable action, linked to using the `oslc:action` predicate, from the Automation Result. Each of these actions **SHOULD** use the `oslc:executes` predicate to link to the future action on the Plan that it relates to. This allows consumers to map a user selection of a future action on the plan to an executable action on the result. Each future action **SHOULD** have *at most one* executable action linking to it from each Result. (Note: If a Plan’s future action `PFA` specifies a binding using the deferred execution dialog interaction pattern, then the corresponding Result’s action bindings linking back to `PFA` might be intended as immediate-execution bindings for the deferred execution dialog (see below), but this specification does not require that usage).



See the [OSLCAutomationTemporaryDeploymentScenarios] for a worked example of future actions.

5.2 Deciding how to use Actions and Automation together

This section is non-normative.

When implementing a provider of Automation Plans, you can decide whether to expose those plans through Actions or not. This section addresses that decision.

There are two main issues that come into play: discovery and execution. In the Automation 2.0 specification, which predated the OSLC Actions specification, Automation Plans

were discovered through query capabilities or selection dialogs on a service provider. This was the only way to discover them. Actions provide an additional option for discovery, in the context of any given resource. That is, if a given Automation Plan “acts on” another resource, it makes sense for that resource to point to that Automation Plan, including information on what executing that plan will achieve. (Plans discovered via Actions can still be made discoverable through the normal means as well, for consumers who don’t want to browse other resources, but instead just want to directly list or select an Automation Plan).

Automation Plans have a well-defined means of requesting execution. Automation Plans are one option for how providers can allow their actions to be executed. However, unlike plain Automation Plans discovered from a query capability or selection dialog, actions allow providers to specify other means of execution in addition to or instead of Automation Plans (while still supporting predictable iteroperability through being implemented against “specification profiles”). See the information on “interaction patterns” and “specification profiles” in the OSLC Actions 2.0 specification for more information.

	Discovery	**Execution**
Automation Plans only	Query capabilities/Selection dialogs	Creation of Automation Request
Actions	On other resources (which will be the context of the execution)	Creation of Automation Request
	(Actions’ Automation Plans can also be made available through query capabilities or selection dialogs as with other plans)	(Actions can also provide other non-Automation Plan bindings that the consumer can choose as an alternative)

6. OSLC Actions Extensions

6.1 Teardown action type

This specification defines the RDF class `oslc_auto:TeardownAction`, as an `rdfs:subClassOf oslc:Action`, with the meaning that any action of this type **MUST** have the semantics of tearing down some deployed resource. It is likely that such a deployed resource was deployed using an OSLC Automation deployment plan, but this **MAY** not be the case. That is, a tear-down action typically has the opposite semantics from a `oslc_auto:Deploy` sub-domain Automation Plan or Request, even if the service provider offers no equivalents in its Automation Plan collection. [auto-15]

6.2 Automation Request interaction pattern

This interaction pattern is defined by the [OSLCActions] (for reuse by other domain specifications).

6.3 Automation Creation Factory interaction pattern

This section defines how to use an [OSLCCore3] Creation Factory **that creates OSLC Automation Requests eligible for immediate execution** as an [OSLCActions] interaction pattern.

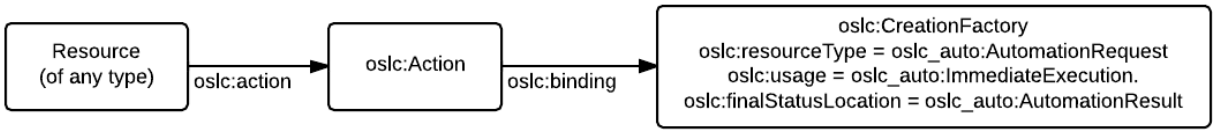
6.3.1 Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one `rdf:type` property UST have the value `oslc:CreationFactory`.
- at least one `oslc:resourceType` property **MUST** have the value `oslc_auto:AutomationRequest`.
- at least one `oslc:usage` property **MUST** have the value `oslc_auto:ImmediateExecution`.
- the `oslc:finalStatusLocation` property **MUST** have the value `oslc_auto:AutomationResult`.

  [auto-16]

A binding is deemed to use this pattern if it meets these restrictions.



6.3.2 Additional provider constraints

The binding **MUST** adhere to the requirements on Creation Factories as defined by the [OSLCCore3]. [auto-17]

> Non-normative note: it is useful to specify `oslc:usage oslc:default` on bindings where there are multiple bindings that use the Creation Factory interaction pattern, especially where the non-default binding does not behave as consumers might expect (for example, if it creates Automation Requests that are not by default eligible for execution) to point consumers to the best one to use when they have no other means to distinguish them.

6.3.3 Execution

To execute an action using this interaction pattern, a consumer follows the instructions for [OSLCCore3] Creating an OSLC Defined Resource in the OSLC Core 2.0 specification.

This interaction pattern does not define how the consumer forms the representation that is sent to the creation factory in the create request. If a consumer does not know how to construct or locate such a representation, then it **MUST NOT** execute a binding using this interaction pattern. The deferred execution dialog interaction pattern defines one way to construct such a representation. [auto-18]

The client's desired result is to successfully complete executing the Automation Request, not just to create it. The status of this desired result is determined using the corresponding Automation Result's state and verdict properties, as would be the case with any other Automation Request, not by using the HTTP status codes. Automation permits both single-message and multiple-message interactions, but the client **MUST** use the state and verdict for determining the [OSLCCore3] status of the client's goal when the HTTP status codes indicate that the creation was successful. [auto-19]

6.4 Deferred execution dialog interaction pattern

This section defines the Deferred-Execution Creation Dialog interaction pattern as an [OSLCActions] interaction pattern designed to address scheduling scenarios. This interaction pattern consists of the following stages:

1. **Configuration**: The consumer displays a deferred-execution creation dialog to a user for them to configure an action. An arbitrary time delay occurs. This accommodates use cases like calendar-schedule execution and manual approval cycles.
2. **Execution**: One or more executions of the configured action. Each execution uses a new resource with a separate lifecycle from the previously configured action, and might either require a user (to supply final configuration values) or might be fully automated.

In this interaction pattern, the consumer is in charge of when the action is executed. (If the provider needs to be in charge of when the action becomes executable, the standard "delegated UI dialog for immediate execution" interaction pattern can be used, with provider exercising whatever degree of control it needs to; for example, creating it immediately and internally holding it, or deferring its creation internally.)

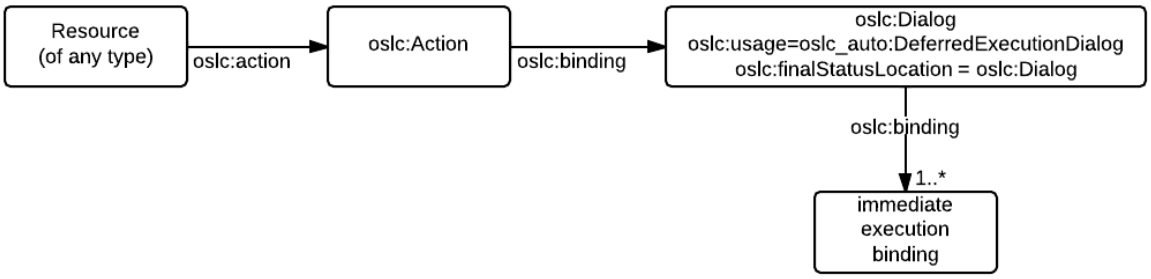
6.4.1 Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one `rdf:type` property **MUST** have the value `oslc:Dialog`.
- at least one `oslc:usage` property **MUST** have the value `oslc_auto:DeferredExecution`.
- the `oslc:finalStatusLocation` property **MUST** have the value `oslc:Dialog`.

[auto-20]

A binding is deemed to use this pattern if it meets these restrictions.



6.4.2 Additional provider constraints

In addition, Action bindings using this interaction pattern:

- **MUST** adhere to the requirements on Deferred-Execution Creation Dialog as defined by this specification.
- **MUST** have at least one `oslc:binding` property on the deferred-execution creation dialog resource, as required by that section. Each of these properties binds the deferred-execution dialog to one or more immediate-execution bindings, which are used in the Execution stage. Once these immediate-execution bindings are executed (in the Execution stage) they immediately execute the action. Hence, they are called immediate-execution bindings. These immediate-execution bindings accept a copy of the configuration previously created by the deferred-execution dialog, to execute the action in the way the user configured but without the user being present at the time at which it is executed. If an immediate-execution binding uses more than one interaction pattern, then at least one of them **MUST** be in the list of permitted immediate-execution bindings below.

[auto-21]

6.4.3 Execution

To execute an action binding using this interaction pattern, a consumer does the following:

1. **Configuation stage**
    1. The consumer follows the requirements in the [OSLCCore3] Delegated UI specification to display the deferred-execution creation dialog (recall that deferred-execution creation dialogs are also standard creation dialogs). The dialog will either return a URI or an error code, which gives the client the status of this phase of its goal.
    2. If the dialog returns a URI, then the consumer performs an HTTP GET request on that URI immediately and stores the result representation to be used at the later time to execute the action.
2. **Execution stage**
    1. If and when the consumer comes to execute the action at a later time, then the consumer selects an interaction pattern and follows its instructions, but with the changes described under immediate-execution bindings below. The client determines the status of this phase of its goal using the selected interaction pattern.
        1. The consumer considers for selection interaction patterns it supports amongst the immediate-execution bindings for the deferred-execution creation dialog (see Additional provider constraints above).
        2. The consumer ignores any interaction pattern prohibited by its defining document from use as immediate-execution bindings for deferred execution dialogs, like those prohibited here.
        3. A single immediate-execution binding might match the recognition rule for multiple interaction patterns; only explicitly prohibited interaction patterns are disqualified from consideration via the preceding step. For example, if three patterns are considered and one is prohibited, the consumer is still free to select either of the remaining two, even if all three exist on a single binding.
        4. The consumer is free to either ignore or retain for consideration interaction patterns whose defining document is silent on their use in this context. This is likely a decision made when the consumer code is written, although it can be made at run time as well.
        5. Interaction patterns defined elsewhere will help consumers by explicitly stating as part of their definition if and how they can be used as immediate-execution bindings for deferred execution dialogs. Consumers might avoid using interaction patterns that fail to do so, because of a reduced likelihood for interoperability.

6.4.4 Immediate-execution bindings

The delegated UI dialog for later execution interaction pattern involves two bindings: one at configuration time that creates the configuration for use at the later time, and a second binding that once executed (with the configuration returned from the first binding) triggers the action immediately. Hence, these second bindings are called “immediate-execution bindings”.

Immediate-execution bindings **MAY** use any of the following interaction patterns for the execution of this interaction pattern. In each case, the input representation **MUST** be replaced by the representation saved during the configuration stage, regardless of whether it is used as a request body, dialog pre-fill, or other purpose by the patterns listed below. [auto-22]

- [OSLCActions] HTTP request with Resource Shape to describe the request body
- [OSLCActions] HTTP request with fixed body
- [OSLCActions] Automation Request
- [OSLCActions] Delegated UI dialog for immediate execution
- Automation Creation Factory

Consumers **MUST NOT** use these interaction patterns on immediate-execution bindings, even if the binding meets the pattern's recognition rule: [auto-23]

- [OSLCActions] HTTP request with empty body cannot be used because there is no conceptual input representation slot.
- Deferred execution dialog interaction pattern cannot be used because execution is deferred.

Other specifications that define new interaction patterns **MAY** state whether or not those interaction patterns can be used as immediate-execution bindings, and if they are allowed, then how to use the template to execute them.

This is new for 2.1: END

7. Automation Service Provider HTTP method support

For V2 of the OSLC Automation specification, support for all HTTP methods in the compliance table is not required for all Automation resources. The following table summarizes the requirements for each resource type, HTTP method and for each media type.

Resource	RDF/XML	XML	JSON	OSLC Compact	HTML	Unspecified
Automation Plan						
GET	MUST	MAY	SHOULD	SHOULD	SHOULD	N/A
PUT	MAY	MAY	MAY	N/A	N/A	N/A
POST	MAY	MAY	MAY	N/A	N/A	N/A
DELETE	N/A	N/A	N/A	N/A	N/A	MAY
Automation Request						
GET	MUST	MAY	SHOULD	SHOULD	SHOULD	N/A
PUT	MAY	MAY	MAY	N/A	N/A	N/A
POST	MUST	MAY	SHOULD	N/A	N/A	N/A
DELETE	N/A	N/A	N/A	N/A	N/A	MAY
Automation Result						
GET	MUST	MAY	MAY	SHOULD	SHOULD	N/A
PUT	SHOULD	MAY	SHOULD	N/A	N/A	N/A
POST	MAY	MAY	MAY	N/A	N/A	N/A
DELETE	N/A	N/A	N/A	N/A	N/A	MAY
Parameter Definition						
GET	MAY	MAY	MAY	MAY	MAY	N/A
PUT	MAY	MAY	MAY	N/A	N/A	N/A
POST	MAY	MAY	MAY	N/A	N/A	N/A
DELETE	N/A	N/A	N/A	N/A	N/A	MAY
Parameter Instance						
GET	MAY	MAY	MAY	MAY	MAY	N/A
PUT	MAY	MAY	MAY	N/A	N/A	N/A
POST	MAY	MAY	MAY	N/A	N/A	N/A
DELETE	N/A	N/A	N/A	N/A	N/A	MAY

OSLC Automation service providers SHOULD support deletion of any resources for which it allows creation.

8. Automation Specification Guidance

This section is non-normative, i.e. it does not affect compliance.

8.1 Canceling the execution of an automation request

The Automation Request and Automation Result resources have an `oslc_auto:desiredState` attribute. A consumer can indicate a desire to cancel the execution of an automation by doing a PUT to the artifact with this attribute set to a value of `http://open-services.net/ns/auto#canceled`. If the service provider supports cancelation of automation executions, the receipt of a PUT with this attribute set should trigger the necessary provider processing. If the cancelation is successful, the service provider should set the appropriate artifact `oslc_auto:state` to `http://open-services.net/ns/auto#canceled`.

- When only an Automation Request is active (Automation Result not created yet), the consumer should request cancelation by setting `oslc_auto:desiredState` on the Automation Request.
- When Automation Requests and Automation Results are active (in an `oslc_auto:state` other than `oslc_auto:canceled` or `oslc_auto:complete`), the consumer should request cancelation by setting `oslc_auto:desiredState` on the Automation Request.
- When only an Automation Result is active (Automation Request completed, canceled or no longer exists), the consumer should request cancelation by setting `oslc_auto:desiredState` on the Automation Result.
- Consumers are responsible for checking the status code of the response to the request for cancelation and for checking the `oslc_auto:state` of the resource.

8.1.1 Responses to Cancelation Requests

If a service provider does not support cancelation of an automation, or if an error occurs preventing successful cancelation, the service provider should respond to the PUT request with an HTTP status code 500 and an [OSLCCore3] Error Resource detailing the cause for the failed cancelation.

8.2 State consistency

The Automation Request and Automation Result resources have an `oslc_auto:state` attribute. Automation service providers should, where possible, enforce state consistency for related Automation Requests and Results. In general, this means an Automation Result in a final state (completed, canceled) should not have a related Automation Request in a non-final state. Other contradictions such as completed Automation Result with a new Automation Request should also be avoided. Suggested consistent (C) and inconsistent (I) states are:

			Automation Result			
AutoRequest	**new**	**queued**	**inProgress**	**canceling**	**canceled**	**complete**
new	C	I	I	I	I	I
queued	C	C	I	I	I	I
inProgress	C	C	C	I	I	I
canceling	C	C	C	C	C	C
canceled	I	I	I	C	C	I
complete	C	C	C	C	C	C

8.3 Parameters Added During Execution

When Automation Requests are created for an Automation Plan, the creator of the request supplies `oslc_auto:inputParameter` attributes based on the `oslc_auto:parameterDefinition` attributes found in the Automation Plan instance. There are scenarios where a provider may add additional parameters during the course of execution and a consumer of Automation Results might wish to discover what these added parameters will be. One method of discovery is for the consumer to simply examine the `oslc_auto:outputParameter` attributes of the Automation Result. This may not be sufficient for consumers who have a need to know the added parameters prior to executing the Automation Plan.

Service providers can advertise which parameters will be added during the course of execution using the `oslc:readOnly` attribute of the `oslc:Property` resource which is the basis for the `oslc_auto:parameterDefinition` in the Automation Plan. By setting `oslc:readOnly` to **true**, the provider indicates that this parameter is not available for the consumer to set, but will or may be added to the Automation Result's `oslc_auto:outputParameters`. Whether it is guaranteed to be added to the Result is based on the value of `oslc:occurs` for the specific parameterDefinition.

Example 1: An Automation Plan parameterDefinition fragment showing a parameter guaranteed to be added during execution

```
<oslc_auto:parameterDefinition>
    <oslc:name>DeployedIPAddress</oslc:name>
    <oslc:readOnly>true</oslc:readOnly>
    <oslc:occurs>http://open-services.net/ns/core#Exactly-one</oslc:occurs>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</oslc_auto:parameterDefinition>
```

Example 2: An Automation Plan parameterDefinition fragment showing a parameter which may be added during execution

```
<oslc_auto:parameterDefinition>
    <oslc:name>FailedTestName</oslc:name>
    <oslc:readOnly>true</oslc:readOnly>
    <oslc:occurs>http://open-services.net/ns/core#Zero-or-many</oslc:occurs>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</oslc_auto:parameterDefinition>
```


9. Compliance

This specification is based on OSLC Core Specification. OSLC Automation consumers and service providers **MUST** be compliant with both the core specification and this Automation specification, and **SHOULD** follow all the guidelines and recommendations in both these specifications. [auto-24]

The following table summarizes the requirements from OSLC Core Specification as well as some (but not all) additional requirements specific to Automation. See the full content of the Automation specification for all requirements. Note that this specification further restricts some of the requirements for OSLC Core Specification as noted in the Origin column of the compliance table. See further sections in this specification or the OSLC Core Specification to get further details on each of these requirements.

Any consumer or service provider behaviors are allowed unless explicitly prohibited by this or dependent specifications; conditional permissive requirements, especially those qualified with "**MAY**", are implicitly covered by the preceding clause. While technically redundant in light of that broad permission, OSLC specifications do still make explicit **MAY**-qualified statements in cases where the editors believe doing so is likely to add clarity.

REQUIREMENTS ON OSLC CONSUMERS

Number	Requirement	Level	Origin(s)	Meaning
Auto-1	Unknown properties and content	**MUST**	Core	OSLC clients **MUST** preserve unknown content.
Auto-2	Unknown properties and content	**SHOULD**	Core	OSLC clients **SHOULD** assume an OSLC service will discard unknown property values.

REQUIREMENTS ON OSLC SERVICE PROVIDERS

Number	Requirement	Level	Origin(s)	Meaning
Auto-3	Unknown properties and content	**MUST**	Core	OSLC service providers **MUST** return an error code if recognized content is invalid.
Auto-4	Unknown properties and content	**SHOULD**	Core	OSLC service providers **SHOULD NOT** return an error code for unrecognized content.
Auto-5	Unknown properties and content	**MAY**	Core	OSLC service providers **MAY** ignore unknown content.
Auto-6	Resource Operations	**MUST**	Core	OSLC service providers **MUST** support resource operations via standard HTTP operations.
Auto-7	Resource Paging	**MAY**	Core	OSLC services **MAY** provide paging for resources.
Auto-8	Partial Resource Representations	**SHOULD**	Core	OSLC service providers **SHOULD** support HTTP GET requests for retrieval of a subset of a resource's properties via the oslc.properties URL parameter.
Auto-9	Partial Resource Representations	**MAY**	Core	OSLC service providers **MAY** support HTTP PUT requests for updating a subset of a resource's properties via the oslc.properties URL parameter.
Auto-10	Service Provider Resources	**MAY**	Core	OSLC service providers **MAY** provide a Service Provider Catalog resource.
Auto-11	Service Provider Resources	**MUST**	Core	OSLC service providers **MUST** provide a Service Provider resource.
Auto-12	Creation Factories	**MAY**	Core	OSLC service providers **MAY** provide creation factories to enable resource creation via HTTP POST.
Auto-13	Query Capabilities	**SHOULD**[1]	Automation, Core	OSLC service providers **SHOULD** provide query capabilities to enable clients to query for resources.
Auto-14	Query Syntax	**MUST**[2]	Automation, Core	If a service provider supports a OSLC query capabilities, the query capabilities **MUST** support the OSLC Core Query Syntax.
Auto-15	Query Syntax	**MAY**	Core	OSLC query capabilities **MAY** support other query syntax.
Auto-16	Delegated UI Dialogs	**SHOULD**	Core	OSLC service providers **SHOULD** allow clients to discover, via their service provider resources, any Delegated UI Dialogs they offer.
Auto-17	Delegated UI Dialogs	**SHOULD**	Core	OSLC service providers **SHOULD** offer delegated UI dialogs for resource creation.
Auto-18	Delegated UI Dialogs	**SHOULD**	Core	OSLC service providers **SHOULD** offer delegated UI dialogs for resource selection.
Auto-19	UI Preview	**SHOULD**	Core	OSLC Services **SHOULD** offer UI previews for resources that may be referenced by other resources.
Auto-20	HTTP Basic Authentication	**MAY**	Core	OSLC Services **MAY** support Basic Auth.
Auto-21	HTTP Basic Authentication	**SHOULD**	Core	OSLC Services **SHOULD** support Basic Auth only over HTTPS.
Auto-22	OAuth Authentication	**MAY**	Core	OSLC service providers **MAY** support OAuth.
Auto-23	OAuth Authentication	**SHOULD**	Core	OSLC service providers that support OAuth **SHOULD** allow clients to discover the required OAuth URLs via their service provider resource.
Auto-24	Error Responses	**MAY**	Core	OSLC service providers **MAY** provide error responses using Core-defined error formats.

Number	Requirement	Level	Origin(s)	Meaning
Auto-25	RDF/XML Representations	MUST[3]	Automation, Core	OSLC service providers MUST accept RDF/XML representations on PUT requests.
Auto-26	RDF/XML Representations	MUST[3]	Automation, Core	RDF/XML representations on POST requests whose semantic intent is to create a new resource instance.
Auto-27	XML Representations	MAY[2]	Automation, Core	OSLC service providers MAY provide a XML representation for HTTP GET, POST and PUT requests that conform to the Core Guidelines for XML.
Auto-28	JSON Representations	MAY[2]	Automation, Core	OSLC service providers MAY provide JSON representations for HTTP GET, POST and PUT requests that conform to the Core Guidelines for JSON.
Auto-29	HTML Representations	SHOULD[3]	Automation, Core	OSLC service providers SHOULD provide HTML representations for HTTP GET requests.

- [1]The OSLC Core Specifications indicates service providers MAY provide Query Capabilities. This specification for OSLC Automation makes Query Capability support a SHOULD requirement.
- [2]The OSLC Core Specifications indicates service providers MAY support the OSLC Query Syntax. This specification for OSLC Automation makes OSLC Query Syntax support a MUST requirement for service providers providing query capabilities.
- [3]For V2 of the OSLC Automation specification, support for all HTTP methods for all automation resources is not required. See the HTTP Method support table for details.

10. Conformance

Implementations of this specification need to satisfy the following conformance clauses.

Clause Number	Requirement
auto-1	Automation Providers **MUST** provide RDF/XML representations. The RDF/XML representation **SHOULD** follow the guidelines outlined in the [OSLCCore3] for RDF/XML.
auto-2	Automation Providers **MUST** accept RDF/XML representations and **MAY** accept XML representations. Automation Providers accepting RDF/XML **SHOULD** be prepared for any valid RDF/XML document. If XML is accepted, Automation Providers **SHOULD** be prepared for representations that follow the guidelines outlined in the [OSLCCore3].
auto-3	Automation Providers **MAY** accept XML and JSON representations. Automation Providers accepting XML or JSON **SHOULD** be prepared for representations that follow the guidelines outlined in the [OSLCCore3].
auto-4	Automation Providers **MUST** provide RDF/XML and **MAY** provide JSON, XML, and Atom Syndication Format XML.
auto-5	`application/rdf+xml` Automation Providers **MUST** respond with RDF/XML representation without restrictions.
auto-6	OSLC Automation service providers **MUST** provide a [OSLCCore3] Service Provider Resource that can be retrieved at a implementation dependent URI.
auto-7	If an OSLC Automation service provider supports the creation of resources, there **MUST** be at least one [OSLCCore3] Creation Factories entry in the Services definition.
auto-8	The Query Capability **MUST** support these OSLC query parameters and **MAY** support others:
auto-9	OSLC Automation 2.1 consumers **MUST** interpret an `oslc_auto:AutomationRequest` creation dialog that has neither `oslc:usage oslc_auto:ImmediateExecution` set nor `oslc:usage oslc_auto:DeferredExecution` set as being an “immediate-execution creation dialog”. This is to maintain compatibility with OSLC Automation 2.0 providers.
auto-10	If an OSLC Automation provider offers both immediate-execution and deferred-execution creation dialogs, it **MUST** provide `oslc_auto:ImmediateExecution` or `oslc_auto:DeferredExecution` as a `oslc:usage` value (respectively) on the `oslc:Dialog` creation dialog resources. Such a provider **SHOULD** also set `oslc:default` as a `oslc:usage` value on the immediate-execution dialogs, to guide OSLC Automation 2.0 consumers to use those dialogs and not the deferred-execution ones.
auto-11	A Deferred-Execution Creation Dialog is a [OSLCCore3] resource creation delegated user interface dialog that creates an Automation Request but **does not** make it eligible for execution. A deferred-execution creation dialog **MUST** comply with all Core requirements on [OSLCCore3] resource creation delegated user interface dialogs. One important consequence of this is that all facilities available on resource creation delegated user interface dialogs, for example [OSLCCore3] pre-filling, apply equally to deferred-execution creation dialogs.
auto-12	The Automation provider **MUST** describe how to immediately execute an Automation Request created by a deferred-execution dialog using one or more `oslc:binding` properties on the `oslc_auto:DeferredExecution` dialog resource. If the deferred-execution dialog is discoverable from a Service in a [OSLCCore3] Service Provider, then the provider **MUST** supply at least one immediate-execution binding whose target uses the Automation creation factory interaction pattern. If multiple `oslc:binding` properties are present, they **MUST** be equivalent alternatives to each other, as defined by Core Actions.
auto-13	The consumer is expected to find a parameter definition from the Automation Plan with its `oslc:propertyDefinition` property set to `http://open-services.net/ns/auto#executionEnvironment`, and to create an InputParameter on the Automation Request for that parameter definition, specifying the execution environment to use (choosing out of those specified on the Automation Plan). If that parameter definition’s `oslc:occurs` property is exactly-one or one-or-more, then the consumer **MUST** specify an execution environment, Otherwise, the consumer **MAY** specify an execution environment.
auto-14	OSLC Automation service providers can identify the state and verdict using references to property values in the OSLC Automation vocabulary or to property values that are not in the Automation vocabulary (i.e. in the service provider’s own vocabulary). It is expected that the state and verdict values will be URI references to property values, but inline resources defining the state and verdict property values are also valid. Automation service providers **MUST** use at least one verdict (Automation Results) and state (Automation Requests and Results) defined in the OSLC automation vocabulary in addition to any states or verdicts not in the Automation vocabulary.
auto-15	This specification defines the RDF class `oslc_auto:TeardownAction`, as an `rdfs:subClassOf oslc:Action`, with the meaning that any action of this type **MUST** have the semantics of tearing down some deployed resource. It is likely that such a deployed resource was deployed using an OSLC Automation deployment plan, but this **MAY** not be the case. That is, a tear-down action typically has the opposite semantics from a `oslc_auto:Deploy` sub-domain Automation Plan or Request, even if the service provider offers no equivalents in its Automation Plan collection.
auto-16	• at least one `rdf:type` property UST have the value `oslc:CreationFactory`. • at least one `oslc:resourceType` property **MUST** have the value `oslc_auto:AutomationRequest`. • at least one `oslc:usage` property **MUST** have the value `oslc_auto:ImmediateExecution`. • the `oslc:finalStatusLocation` property **MUST** have the value `oslc_auto:AutomationResult`.
auto-17	The binding **MUST** adhere to the requirements on Creation Factories as defined by the [OSLCCore3].
auto-18	This interaction pattern does not define how the consumer forms the representation that is sent to the creation factory in the create request. If a consumer does not know how to construct or locate such a representation, then it **MUST NOT** execute a binding using this interaction pattern. The deferred execution dialog interaction pattern defines one way to construct such a representation.
auto-19	The client’s desired result is to successfully complete executing the Automation Request, not just to create it. The status of this desired result is determined using the corresponding Automation Result’s state and verdict properties, as would be the case with any other Automation Request, not by using the HTTP status codes. Automation permits both single-message and multiple-message interactions, but the client **MUST** use the state and verdict for determining the [OSLCCore3] status of the client’s goal when the HTTP status codes indicate that the creation was successful.
auto-20	• at least one `rdf:type` property **MUST** have the value `oslc:Dialog`. • at least one `oslc:usage` property **MUST** have the value `oslc_auto:DeferredExecution`. • the `oslc:finalStatusLocation` property **MUST** have the value `oslc:Dialog`.

Clause Number	Requirement
auto-21	• **MUST** adhere to the requirements on Deferred-Execution Creation Dialog as defined by this specification. • **MUST** have at least one `oslc:binding` property on the deferred-execution creation dialog resource, as required by that section. Each of these properties binds the deferred-execution dialog to one or more immediate-execution bindings, which are used in the Execution stage. Once these immediate-execution bindings are executed (in the Execution stage) they immediately execute the action. Hence, they are called immediate-execution bindings. These immediate-execution bindings accept a copy of the configuration previously created by the deferred-execution dialog, to execute the action in the way the user configured but without the user being present at the time at which it is executed. If an immediate-execution binding uses more than one interaction pattern, then at least one of them **MUST** be in the list of permitted immediate-execution bindings below.
auto-22	Immediate-execution bindings **MAY** use any of the following interaction patterns for the execution of this interaction pattern. In each case, the input representation **MUST** be replaced by the representation saved during the configuration stage, regardless of whether it is used as a request body, dialog pre-fill, or other purpose by the patterns listed below.
auto-23	Consumers **MUST NOT** use these interaction patterns on immediate-execution bindings, even if the binding meets the pattern's recognition rule:
auto-24	This specification is based on OSLC Core Specification. OSLC Automation consumers and service providers **MUST** be compliant with both the core specification and this Automation specification, and **SHOULD** follow all the guidelines and recommendations in both these specifications.

Appendix A. Version Compatibility

Appendix B. Samples

This section is non-normative.

This section is non-normative, i.e. it does not affect compliance.

See OSLC Automation Version 2.1 Samples

Appendix C. Acknowledgments

This section is non-normative.

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

Participants:

Fabio Ribeiro, Koneksys (Editor)
Jim Amsden, IBM (Chair)
Graham Bachelor, IBM (Chair)
Michael Fiedler
John Arwe
Charles Rankin
Paul McMahan
Martin Pain, IBM
Umberto Caselli
Stephen Rowles
Steve Speicher

