Exposing arbitrary actions on RDF resources

New here? Sign up! (http://open-services.net/forums/member/register)

Recent changes (http://open-services.net/wiki/core/Special:Recentchanges)
Core home (http://open-services.net/wiki/core)
All pages (http://open-services.net/wiki/core/Special:Titles)
Categories (http://open-services.net/wiki/core/Special:Categories)
Random page (http://open-services.net/wiki/core/Special:Random_page)
Uploaded Files (http://open-services.net/wiki/core/Special:Files)
RSS (http://open-services.net/wiki/core/rss/)
Basic syntax guide (http://stackoverflow.com/editing-help)
Linking and categories syntax (http://expressionengine.com/user_guide/modules/wiki/wiki_syntax.html)

Sort of a ghost town here
Active specification development is now at Core TC (http://open-services.net/workgroups/core-tc)

Want to contribute?

1. Register (/forums/member/register/)
2. Complete Members Agreement (/legal-agreements/members-agreement/)
3. Complete WPA (http://open-services.net/legal-agreements/core-wpa)

Mailing list (http://open-services.net/mailman/listinfo/oslc-core_open-services.net)
Workgroup information (http://open-services.net/workgroups/core)

History (http://open-services.net/wiki/core/Exposing-arbitrary-actions-on-RDF-resources/history) View (http://open-services.net/wiki/core/Exposing-arbitrary-actions-on-RDF-resources) Links to this page (http://open-services.net/wiki/core/Special:Associated_Pages/Exposing-arbitrary-actions-on-RDF-resources) 2015 February 17 05:42 am

Redirected from: Actions 2.0 (http://open-services.net/wiki/core/Actions-2.0/noredirect)

Open Services for Lifecycle Collaboration
Actions Specification Version 2.0

Status: Finalization - 17 February 2015
Issues: Actions 2.0 Issues (http://open-services.net/wiki/core/Actions-2.0-Issues)

This Version
http://open-services.net/wiki/core/Actions-2.0/ (http://open-services.net/wiki/core/Actions-2.0/)

Latest Version
http://open-services.net/wiki/core/Actions-2.0/ (http://open-services.net/wiki/core/Actions-2.0/)

Previous Version
This is the first version of this specification. The 2.0 designation shows its alignment with Core 2.0, as is this community’s practice.

Authors
John Arwe (http://open-services.net/bin/view/Main/JohnArwe) (IBM, OSLC-Core)
Martin Pain (IBM, OSLC-Automation Lead)
Samuel Padgett (IBM, OSLC-Change-Management Lead)
SteveSpeicher (http://open-services.net/bin/view/Main/SteveSpeicher) (IBM, OSLC-Core Lead)

Contributors
See Contributors section below

Contents
Open Services for Lifecycle Collaboration
Actions Specification Version 2.0
Intended audience
Terminology
Overview (non-normative)
What are actions?
High level interaction model
What do consumers need to know?
Domain-specific consumers
Providers
Generic or cross-domain consumers
Templates
Description

- Re-use by domain specifications
- Discovering the actions available from a resource
- Action resources
- Types of actions
- Action bindings: instructions for executing available actions
- Future actions
- Recognizing the interaction patterns used by each binding

Interaction patterns

- Final execution status
  - Pattern: HTTP request with empty body
    - Pattern recognition rule
    - Execution
  - Pattern: HTTP request with Resource Shape to describe the request body
    - Pattern recognition rule
    - Execution
  - Pattern: HTTP request with fixed body
    - Pattern recognition rule
    - Execution
  - Pattern: Automation request
    - Pattern recognition rule
    - Additional provider constraints
    - Execution
  - Pattern: Delegated UI dialog for immediate execution
    - Pattern recognition rule
    - Execution
  - Pattern: Delegated UI dialog for later execution
    - Pattern recognition rule
  - Pattern: Automation Creation Factory

Resources

- Actions Namespace
- Other namespaces
- Resource: Action
  - Action Properties
  - Action subtypes
  - Common Property: oslc-action
- Resource: Request
  - Request Properties
  - Resource: results
- Resource: ParameterInstance
  - ParameterInstance Properties

Best Practices

- Minimize the number of HTTP round-trips

Specification profiles

- Specification profiles overview
- Using specification profiles
- Recognizing profiles
  - Why are profiles only identified in documentation, not data?
- Choosing one or more profiles to implement
- Other specifications
- Terminology
- Specification profile definitions
  - Profile: POST RDF described by a OSLC Resource Shape to the Action resource
  - Profile: Create an Automation Request

Appendix A: Constructing an HTTP request from an http:Request resource

Appendix B: Examples

Appendix D: Notices and References

Contributors

Reporting Issues on the Specification

License and Intellectual Property

References

Appendix E: Changes

Changes during 2.0 convergence phase

Changes to OSLC Core vocabulary

Notation and Conventions


Any sections marked as notes, as examples, or as non-normative do not affect compliance.

Intended audience

This section is non-normative; it provides a guide as to which sections of the specification it is important for different audiences to understand and comply with.

- Implementors of OSLC consumers - To reduce implementation effort, implementations may choose to be compliant with only a subset of this specification. To support interoperability between implementations that do so it is recommended that consumers pick one or more specification profiles and support, as a minimum, the subset of this specification included in those profiles.
- Implementors of OSLC providers - To reduce implementation effort, implementations may choose to be compliant with only a subset of this specification. To support interoperability between implementations that do so it is recommended that providers pick one or more specification profiles and ensure that every action that they provide has at least one Action binding that falls within the constraints of each profile.
- Authors of specifications that re-use Action resources - It is recommended that domain specifications reusing Action resources select a specification profile that implementations of the domain specification should support as a minimum. Such specifications can refer to a profile that is defined or referenced in this specification, or may define their own. Such specifications may also want to define additional interaction patterns, but are advised to reuse or extend the ones defined in this specification if possible, to aid interoperability.

Terminology

Action - Linked Data-based hypertext describing an end - the desired semantics - and some means to accomplish that end via HTTP-based interaction pattern(s) that may or may not also involve user interaction. Although the term’s use in this specification is primarily to actions available when the response is formed, that limit does not come from this definition. Action resources can be used to describe other varieties, like past, future, or potential actions; enabling this additional flexibility in the definition is intentional, and it has been exploited already by at least the Automation domain (http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/).

Action binding - a means of executing an action, or instructions for executing an action. Each action may have multiple action bindings to allow the action to support multiple interaction patterns. Each action binding uses an interaction pattern and specifies values (e.g. URL and request body contents) that the consumer must apply to that interaction pattern to execute that particular action.

Interaction pattern - an abstract definition of how a consumer and provider interact in order to execute an action. The definition includes a rule consumers use to recognize that an action binding uses the interaction pattern, instructions for sending messages, and instructions that consumers use to determine the results. For a simple interaction pattern using a single-request HTTP flow, determining the result can be as simple as inspecting the HTTP status code.
What are actions?
This specification provides a means of advertising actions that can be performed on (or in the context of) a specific resource. For example, a change request can have a “resolve” action or other state transitions (http://open-services.net/wiki/change-management/State-Transitions-for-Change-Requests#CM-Actions-with-the-Core-Workgroup-Actions-Proposal). Providers implementing this specification allow consumers to list and select actions that can be executed on a given resource R without needing to understand R’s type, properties, lifecycle, etc. See What do consumers need to know? Each action provides a few primary pieces of information: “what does this action do?”, “how do I execute this action?”, and “how do I determine if the action succeeded or failed?”. These are described in the normative specification sections below.

What do consumers need to know?
By using this explicit, loosely-coupled description of actions that are available on a given resource R, consumers do not need to understand:
- the rdf:type of R, or the domain that defines it
- the native lifecycle of R
- the properties of R unrelated to Actions, or the meaning of their values

instead, the consumer only needs to know:
- the URI of R
- how to find the actions and their bindings that exist on R (as defined by this specification)
- how to execute those bindings (as defined by this specification)
- how to determine the success/failure of its request to execute the action (as defined by this specification)

Therefore this specification frees the execution of these actions from domain specifics, instead providing a common point of interoperability between multiple domain specifications.

Domain-specific consumers
The first-order concern of domain specifications is consumers written to that specification. Domain specifications provide one or more specification profiles for the consumers to support. Those profile(s) define the minimum subset of this specification that the consumer must implement. The consumer itself is not aware of specification profiles at run time – it is a design-time decision about what interaction patterns to implement with what (if any) restrictions, and how to document those design-time decisions.

The process that a consumer supporting a given profile (whether or not that profile was provided by a domain specification) would follow to find resources with actions it is capable of consuming is usually to form and make an HTTP request, but some interaction patterns dictate a different process. The instructions for forming and making the request are found by following oslc:binding link(s) and examining the object resources.
- Given the action to execute and the binding(s) for that action that the consumer is compatible with (found using the process above)
- If more than one binding is compatible with the consumer, choose one using whatever criteria the consumer prefers, e.g. it can have an order of preference (decided at implementation time) based on the interaction patterns’ cost of execution (e.g. time, resources), or the desired interaction with the user (e.g. dialogs, or ability to execute at a later time).
- Follow the instructions of the selected interaction pattern to execute the action via the chosen binding. For example, if the binding is an http://Request resource, form the HTTP request from it.

Note: no mention is made of how many HTTP requests are required to retrieve the Actions/bindings/etc. data mentioned, since that is implementation-dependent. Best practices suggest minimizing them, especially when UI presentation performance is important.

Providers

The second-order concern of domain specifications is providers written to that specification. The process such a provider would follow to expose resources with actions its domain-specific clients are capable of consuming is:

1. Decide what the set of interesting resources is.
2. The most likely case is resources owned/exposed by the provider itself.
   - For each interesting resource:
     - Find any existing actions that the provider can add a new binding for
     - The action that the binding is for must be semantically equivalent to the provider’s action to be exposed, where equivalence is defined by the provider of the action. (There is no loosely-coupled means of determining semantic equivalence between providers, other than when two bindings are already on the same oslc:Action. So usually new bindings can only be added to existing actions by the same provider that created the bindings already on the action, or some implementation-specific means would have to exist for the two providers to agree that the two bindings are semantically equivalent).
   - Add the provider’s action to be exposed, either as a new oslc:binding link or as a new oslc:action link.

Generic or cross-domain consumers

Domain specifications often care little about consumers not written to that particular specification, beyond adhering to basic HTTP and Linked Data principles. Core specifications, in effect, enable cross-domain consumers.

The process such a consumer would follow to find resources with actions it is capable of consuming, and executing those, is the same as for a domain-specific consumer, but where the interaction patterns supported by the consumer are as wide a set as the implementation allows, rather than being based on a specification profile. A generic consumer could, for example, look at all actions (oslc:action predicates) across a set of resources, present those to a user, and let the user select one or more to invoke. The only restriction on the set of interaction patterns is the set of those understood by the consumer code, since there is no single domain specification or profile involved.

Templates

One particular usage pattern for Actions is their use as templates, especially by providers. An analogy is a partially-filled-out form, for example a drug prescription, that one copies and submits (renews) multiple times. It is not always easy, given one of those forms, to distinguish between the original and the copies; they have nearly identical content, but they are used very differently. The original is used to manufacture new copies, and each copy is “trailed” for a set quantity and type of medicine.

The same pattern occurs with schedulers: during a “configuration” phase, a form is filled out (for future use) and saved; later in time, during one or more “execution” or “use” phases, a copy of the saved form is submitted in order to cause a result such as executing an Action. The resource acting as the saved form and the copies made from it have very similar representations, but distinct identifiers (URIs). The copying process is usually accomplished by submitting a representation (of the resource acting as the template) to a creation factory, which responds to successful requests with a newly minted URI (identifying the new copy). Creation factories implemented via HTTP POST have considerable discretion in how they use the input representation, and in some cases (like OSLC Automation (/wiki/automation)) the act of creation may carry with it side effects, like execution of an Automation Request. There is nothing specific to OSLC Automation in the template usage pattern, however; indeed, readers might well recognize a similar pattern in saved queries and favorite lists on many Web sites. Hence, the interaction patterns described later can generally each be used as part of a template pattern at a higher level of abstraction.

Description

Re-use by domain specifications

Domain specifications re-using this specification SHOULD make their readers aware of Best Practices for representing actions and their related resources.

Domain specifications re-using this specification are strongly RECOMMENDED to require at least one specification profile for interoperability within the target domain. They SHOULD re-use the profiles defined or referenced in this specification where possible, or they MAY define new specification profile(s) when no existing profile meets their needs. The working groups defining them must consult with OSLC Core before defining new profiles, to determine if they are better added to Core or kept separate.

Domain specifications re-using this specification MAY add new constraints beyond the ones here. Domain specifications re-using this specification MUST NOT weaken absolute constraints (MUST, MUST NOT) in this specification.

Domain specifications re-using this specification MAY define new predicates beyond the ones here that link to actions not available when the response is formed. Likewise they MAY define new action types, as OSLC Automation 2.1 (/wiki/automation/OSLC Automation Specification Version 2.1/does for teardown actions (/wiki/automation/OSLC Automation Specification Version 2.1/TeardownAction) type): widely applicable action types SHOULD be allocated (by the Core working group) within the OSLC Core vocabulary.

Domain specifications re-using this specification MUST comply with requirements imposed by later sections in this specification, for example those in recognizing interaction patterns and specification profiles. The preceding list of examples might be incomplete; domain specification authors are responsible for finding and understanding all requirements levied on them in other sections.

Discovering the actions available from a resource

Any resource MAY link to any number of Actions, using the oslc:action predicate as shown in the graphic below. The oslc:action predicate links to actions available when the response is formed. (In a distributed system, clients can lose race conditions that result in an action that appears as “available” having its execution requests rejected. However, it is not expected that actions will change frequently.)

Action resources

All resources in this specification, such as Actions, action bindings, and HTTP requests, MAY be represented as either blank nodes or URI-identified resources, and when identified by URIs either hash or non-hash URIs MAY be used.

Types of actions

Action resources identify what type of actions they are, so that consumers can know how to choose an action (for programmatic selection) or how to display an action (for selection by a user). For loose integration, some general action types are defined in this specification, and types applicable to individual applications are defined in specifications for those domains (for example, OSLC Change Management (/wiki/change-management) and OSLC Automation (/wiki/automation)). This specification lists some known action sub-types in Resources: Action subtypes.

Action types are identified by rdf:type values, e.g. oslc:action|TeardownAction, which SHOULD all be rdfs:subClassOf oslc:Action. Since informal OSLC Core guidance states that providers cannot depend upon client-side inferencing, action representations are multi-typed: they MUST contain oslc:Action as one type, and MAY (usually do, in practice) contain additional types that convey more specific semantics necessary for programmatic consumption. The additional type URI MAY belong to any vocabulary or namespace, however working groups in need of new types should consult with Core before defining new actions outside the Core namespace. Widely applicable action types should be allocated (by the Core working group) within the OSLC Core vocabulary.

Action types should be at a level of abstraction suitable for human and/or programmatic consumption.
Implementations MUST NOT assume that actions with the same action types are semantically equivalent. Actions SHOULD use a combination of both standardised and (if needed) non-standardised types to make the identification of its type more specific. The standardised types allow loosely-coupled integration, and the more specific non-standard types allow comparison between actions and value-add for consumers that do know about the non-standard terms at implementation time. (However, as stated earlier in this paragraph, implementations cannot assume that actions with exactly the same set of RDF:type values are identical – they are only as equivalent as those types define.)

As a hypothetical example, two actions could both identify themselves as ns:DeletionAction, but one could delete a child resource of the current context, and one could delete the context itself. It is more useful to (some) consumers if the actions identify themselves as being of-type [oslc:Action, ns:DeletionAction, x:DeleteDelegateResource] and [oslc:Action, ns:DeletionAction, x:DeleteCurrentResourceAction], respectively. This is similar to what the standard types allow loosely-coupled integration, and the specific non-standard types allow comparison between actions and value-add for consumers that do know about the non-standard terms at implementation time.

**Action bindings: instructions for executing available actions**

Each available action contains one or more oslc:binding predicates, each of which links to a resource containing instructions for executing the action, such as the [HTTP request resource](http://www.w3.org/TR/HTTP-in-RDF10#RequestClass) shown below; the resource shape allows zero bindings in other cases.

A consumer executes an action by following a single interaction pattern, applying information supplied in the action binding to the pattern. Out of the patterns that it supports, the consumer MUST use a pattern whose recognition rule is matched by the binding. If multiple patterns match (theoretically possible, but not currently specified), the consumer MAY use any interaction pattern whose recognition rule is matched by the action binding. This specification permits the use of resources with types in addition to those defined in interaction patterns in this specification, but it assigns them no meaning and when they are used interoperability will be reduced. Consumers MUST tolerate other resource types, but MAY ignore them.

Each binding describes a means that the consumer can use to execute the action, by matching one or more interaction patterns and supplying data needed to perform it. Each of the bindings linked to from a single Action MUST be semantically equivalent – the different bindings are different means of achieving the same desired result, within the bounds of what the Action’s type(s) require and prohibit. However, the different bindings MAY (and almost always will) source any parameters from different sources (e.g. some use a dialog), MAY have different numbers of parameters, MAY be used to execute the action at different times (e.g. some can be used to schedule it for execution at a later time), and MAY have different side effects unless its types ‘semantics’ specifically prohibit them.

The consumer picks a binding based on which interaction pattern(s) the consumer understands and which one it wants to use. The consumer then executes the action in the manner defined by the interaction pattern.

**Future actions**

This specification defines a predicate for discovering actions that are not available when the response is formed but that might become available at some point in the future, possibly on a related resource. See [OSLC Automation 2.1](http://www.w3.org/2004/01/ods-lc-automation)’s associated scenarios for example uses specific to the Automation domain, and the paragraph below on resource shapes for another. Domain specifications SHOULD re-use oslc:futureAction rather than inventing their own new term, when the semantics are compatible. This specification also defines a oslc:executes predicate for linking from a currently available action to a corresponding future action; this was also motivated by Automation 2.1’s scenarios.

![Diagram](image)

Resource shapes can be used as a general way to expose the set of actions that a resource is capable of supporting but cannot be executed at the point in time when action discovery is occurring. Existing resources can link to a shape, for example via oslc:instanceShape. Create factories can link to a shape describing the resources they are capable of manufacturing, via a oslc:resourceShape link from a creation factory (bin/view/Main/OslcCoreSpecification). In each case, if the shape has one or more oslc:futureAction properties, then clients will expect that future action resources describe actions that the client might find linked to instances that the shape applies to, via the oslc:action predicate. That is, if a shape B applies to a resource R, and the shape B has an oslc:futureAction link to action A, then a client would expect that at some point in the future, the client could link to an oslc:futureAction link to a “add log entry” action, where those “add log entry” actions might not be available all the time, or to every user, or on every work order the factory creates, and clients have to tolerate that; future actions are hints, not guarantees.

This specification does not require that future actions have any bindings, that any bindings they do have are complete, or what the relationship is between any bindings present on a future action and those present when it becomes available for execution. Domain specifications re-using this specification MAY add such requirements.

**Recognizing the interaction patterns used by each binding**

Each interaction pattern defines how consumers should recognize it.

Each action binding can match more than one interaction pattern, in which case the consumer can choose which one to use.

When defining new interaction patterns in other specifications, the new patterns SHOULD either:

- extend an existing interaction pattern, but add in additional recognition conditions that use predicates under the control of the extending specification (to avoid incorrectly matching existing action bindings that do not follow the newly defined extension interaction pattern).
- These extension interaction patterns MUST be 100% compatible with the patterns they extend – that is, when Action bindings use these extension patterns the providers MUST support those bindings being executed as either the base or extension pattern.
- New interaction patterns that extend existing patterns MUST include an explicit reference to the extended pattern’s recognition rule in the new pattern’s recognition rule.
- They MAY also include a copy of the extended pattern’s rule alongside that reference for clarity.

or:

- define a completely new interaction pattern that is recognized by (at least) an RDF:type value that is under the control of the extending specification.

**Interaction patterns**

Consumers invoke actions to achieve a certain desired result, by choosing a specific binding and interaction pattern to construct one or more messages. A single action can have multiple bindings, and a single binding can potentially be used with multiple interaction patterns. Some interaction patterns always consist of a single message, but others permit or require multiple messages to achieve the same desired result. This distinction becomes critical when a consumer is trying to determine whether or not its desired result has been achieved, based on message responses - that is, what the final status of the action’s execution is.

When using interaction patterns that always consist of a single HTTP message flow, consumers expect the HTTP status code to equate to the success or failure of the desired result (the final status of executing the action): if a success status code (2xx class) is returned, consumers interpret that to mean that the action ran successfully. Single-message interaction pattern definitions SHOULD avoid other interpretations.

When using interaction patterns that sometimes or always consist of multiple HTTP message flows, in general consumers cannot expect “the” HTTP status code to equate to the success or failure of the desired result (the final status of executing the action), because the issue of which message’s status code to use arises. Multi-message interaction patterns MUST define how a consumer unambiguously determines the final status of its desired result from the messages.

Non-normative note: feedback on draft specifications indicated more general scenarios exist outside the Automation domain, that could be satisfied by re-using existing Core capabilities if Core Actions was explicit on how to do so.

Non-normative concrete example: if work order has a shape with a future action link to a “add log entry” action, then clients will expect that the work order is likely to have an oslc:action link to at least one “add log entry” action; if a work order creation factory B has a resource shape with a future action link to a “add log entry” action, then clients will expect that all work orders it creates are likely to have an oslc:action link to at least one “add log entry” action. Those “add log entry” actions might not be available all the time, or to every user, or on every work order the factory creates, and clients have to tolerate that; future actions are hints, not guarantees.
Each interaction pattern specifies:

- A pattern recognition rule - the restrictions that bindings must adhere to in order for consumers to recognize that pattern as one a binding uses.
- Instructions for execution of a binding that use that interaction pattern.
- Instructions for finding the final status of executing a binding using that interaction pattern.
- Optionally, additional provider constraints that must be adhered to when using that pattern.

Non-normative note: specifying oslc:usage = oslc:default on at least one of an action's bindings can help consumers decide between bindings. This is useful whenever multiple bindings exist on the action and several bindings satisfy the recognition rule for the same interaction pattern. If the provider marks one as a default, consumers have a clearer choice.

Final execution status

The Actions vocabulary defines a predicate oslc:finalStatusLocation whose RDF object conveys instructions telling a consumer how to determine the actual result of using the interaction pattern. This specification describes the following values for the oslc:finalStatusLocation predicate, and their meanings when they are used. Other specifications may define additional values. Exactly one value must be present; if a provider implementation happens to work in such a way that multiple values would always give the same result and it wishes to cater to both sets of consumers, it needs to create one binding for each value as described in the note below.

- http:StatusCode is used when the HTTP 1.1 status code conveys the actual result. This includes the status code class processing defined in HTTP, that is that any 2xx status code means success, and so on. Only single-message interaction patterns can use this value.
- oslc:Dialog is used when the OSLC Core delegated dialog protocol determines how to determine the actual result, which uses a combination of the HTTP status code and the oslc:results output object.
- oslc:ActionDialog is used when the action dialog protocol determines how to determine the actual result, which uses a combination of the HTTP status code and the oslc:results output object.
- oslc:action:AutomationResult is used when an Automation Result's state and verdict properties (see Automation Result's state and verdict properties) convey the actual result. This is typically used for cases where providers expect an action's implementation to be long-running, and therefore to require an asynchronous style (see Asynchronous and Synchronous Automation Execution) using multiple messages.

Other values can be used, but some consumers will not understand them.

Non-normative note: The final status location is defined in the binding's RDF (and is included in the pattern's recognition rule) - not just implied by which pattern is used - so that in future other interaction patterns can be defined that are identical to existing interaction patterns, except in the location of the final status. For example, if a provider always performs synchronous execution of the Automation Request (see Asynchronous and Synchronous Automation Execution) and sets the response's status code to match the Automation Result's verdict (see Asynchronous and Synchronous Automation Execution), and it wants to make its actions executable both by consumers that only understand OSLC Automation requests and by consumers that only understand standard HTTP requests, such a provider could create two bindings that differ only in the location of the oslc:finalStatusLocation triple. Core Actions provides no facility to avoid repeating the rest of the binding information, although standard RDF facilities can be used.

Pattern: HTTP request with empty body

This interaction pattern consists of the consumer making an HTTP request with a zero-length body to a URL specified by the provider to execute the action.

- Interaction pattern (See Interaction patterns)
- Identifier: "pattern-http-empty-body"
- Name: HTTP request with empty body
- Contained in profile: POST RDF described by a OSLC Resource Shape to the Action resource ("profile-action-shape-post")

Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one rdf:type property MUST have the value http:Request, and
- the http:body property MUST have the value rdf:nil.
- the oslc:finalStatusLocation property MUST have the value http:StatusCode.

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

See the Examples appendix (see Examples) for a full RDF example.

Execution

To execute an action using this interaction pattern, a consumer uses the binding's properties to send a request as described in Appendix: Constructing HTTP requests from http:Request resources, but with a Content-Length header with a value of zero, and an empty request body.

Pattern: HTTP request with Resource Shape to describe the request body

This interaction pattern allows an Action to describe the request body as an OSLC resource shape.

- Interaction pattern (See Interaction patterns)
- Identifier: "pattern-http-resource-shape"
- Name: HTTP request with Resource Shape to describe the request body
- Contained in profile: POST RDF described by a OSLC Resource Shape to the Action resource ("profile-action-shape-post")

Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one rdf:type property MUST have the value http:Request, and
- the http:body property MUST link to a resource with at least one rdf:type of oslc:ResourceShape.
- the oslc:finalStatusLocation property MUST have the value http:StatusCode.

A binding is deemed to use this pattern if it meets these restrictions.
See the Examples appendix (/wiki/core/Actions-2.0-Examples/#example-resource-shape) for a full RDF example.

Execution

To execute an action using this interaction pattern, a consumer sends a request as described in Appendix: Constructing HTTP requests from http:Request resources. The consumer constructs the HTTP body by finding or constructing an RDF resource that matches the defined resource shape and using an appropriate serialization of that resource as the HTTP request body.

Pattern: HTTP request with fixed body

This interaction pattern allows an action binding to provide a resource that is to be serialized as the request body.

- **Interaction pattern** (See Interaction patterns)
- **Identifier**: "pattern-http-fixed-body"
- **Name**: HTTP request with fixed body
- **Contained in profiles**: (None)

Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one rdf:type property MUST have the value http:Request, and
- the http:body property MUST link to a resource that meets these conditions:
  - at least one rdf:type property MUST have the value oslc-automation:ParameterInstance, and
  - MUST have exactly one rdf:value property that links to the resource that is to be serialized
- the oslc:finalResultLocation property MUST have the value http:StatusCode.

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

See the Examples appendix (/wiki/core/Actions-2.0-Examples/#example-body-repn) for a full RDF example.

Execution

To execute an action using this interaction pattern, a consumer sends a request as described in Appendix: Constructing HTTP requests from http:Request resources. The consumer constructs the HTTP body by serializing the resource that is the object of the rdf:value property of the oslc-automation:ParameterInstance resource, and the consumer uses the HTTP Content-Type header to specify the media type used on the request to execute the action.

Non-normative note: The consumer will have to choose a media type that is appropriate. Appropriate media types could be detected from the http:headers property on the http:Request resource if present, or from the Content-Type used on previous responses the consumer has received from the provider (for example, when retrieving the rdf:value’s representation), or (when the binding specifies a POST request) from an Accept-Post response header from the resource linked to by the http:requestURI property.

Pattern: Automation request

This interaction pattern allows an Action to be executed by creating an Automation Request.

- **Interaction pattern** (See Interaction patterns)
- **Identifier**: "pattern-automation-request"
- **Name**: Automation request
- **Contained in profile**: Create an Automation Request ("profile-automation-request")

Pattern recognition rule

For any action binding that uses this interaction pattern:

- at least one rdf:type property MUST have the value http:Request, and
- the http:body property MUST link to a resource that meets these conditions:
  - at least one rdf:type property MUST have the value oslc-automation:AutomationRequest,
  - the oslc:finalResultLocation property MUST have the value oslc-automation:AutomationResult.

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

Non-normative note: the rdf:type property requirement implies that only RDF representations are compatible with this pattern, as would be expected from its dependence on RDF resources defined in OSLC Automation.
This pattern is defined by the OSLC Automation specification:

dialogs, which return a URI to a domain- or implementation-specific resource - action dialogs only have to return a verdict whose URI identifies the final status of the action’s execution).

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

For any action binding that uses this interaction pattern:

- **MUST** specify POST as the value of the http:method property in the action binding’s http:Request resource
- **MUST** specify an oslc-automation:executesAutomationPlan property on the Automation Request, pointing to the Automation Plan that this action binding executes.
- are **RECOMMENDED** to use the http:requestURI property of the action binding’s http:Request resource to link to a standard creation factory (described by a oslc:CreationFactory resource) that accepts Automation Requests for the appropriate service provider, so clients can obtain the creation factory’s URI from the oslc:creation property.
- **MUST** accept any valid Automation Requests for the Automation Plan executed by this action binding (i.e. it must not rely on the consumer submitting exactly the Automation Request that was provided by the Action), if and only if a GET on the http:requestURI returns a resource with an rdf:type of oslc:CreationFactory

### Execution

When executing an action binding according to this interaction pattern, a consumer follows the execution instructions of the HTTP request with fixed body interaction pattern, with the exception that the consumer constructs the HTTP body content from the Automation Request. The consumer may construct this body content by merely serialising the Automation Request that has been provided, or they may alter that Automation Request to provide additional or different parameter values if they understand the parameters that the linked Automation Plan takes, or they may find another way to construct an Automation Request for that Automation Plan (e.g. by using a stored Automation Request created earlier, or by finding a deferred-execution creation dialog (/wiki/automation/OSLC-Automation-Specification-Version-2.1#Template-Dialog-interaction-pattern) to create the Automation Request).

The client’s desired result is to successfully complete execution of the action; creating the Automation Request is merely the first step. The status of this desired result is determined using the corresponding Automation Result’s state and verdict properties (/wiki/automation/OSLC-Automation-Specification-Version-2.1/#State-and-Verdict-properties), as would be the case with any other Automation Request, not by using the HTTP status codes. Automation permits both (http://open-services.net/bin/view/Main/OslcCoreSpecification#Delegated_User_Interface_Dialogs) single-message and multiple-message interactions, but the client **MUST** use the state and verdict for determining the status of the client’s desired result (/wiki/core/Exposing-arbitrary-actions-on-RDF-resources/#Interaction-patterns) when the HTTP status codes indicate that the creation was successful.

**Pattern: Delegated UI dialog for immediate execution**

This interaction pattern consists of the consumer displaying a delegated UI dialog to a user for them to perform the action immediately, and waiting for the action to be completed or cancelled before returning to the consumer.

- **Interaction pattern** (See Interaction patterns)
- **Identifier:** “pattern-action-dialog”
- **Name:** Delegated UI dialog for immediate execution
- **Contained in profiles:** (None)

**Pattern recognition rule**

For any action binding that uses this interaction pattern:

- at least one rdf:type property **MUST** have the value oslc:Dialog, and
- at least one oslc:usage property **MUST** have the value oslc:ActionDialog.
- the oslc:finalStatusLocation property **MUST** have the value oslc:ActionDialog.

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

### Execution

To execute an action binding using this interaction pattern, a consumer follows the instructions in the OSLC Core Delegated UI specification (http://open-services.net/bin/view/Main/OslcCoreSpecification#Delegated_User_Interface_Dialogs) to display the dialog. The dialog is responsible for everything necessary to execute the action, including but not limited to accepting any parameters and displaying any output, and **SHOULD** be displayed until the action is completed or cancelled. The dialog **MUST** return a oslc:results object conforming to the resource shape for action dialog results.

If the HTTP status code indicates success, the consumer determines whether or not the action succeeded using the output oslc:results object’s verdict property. (This is different than selection dialogs and creation dialogs, which return a URI to a domain- or implementation-specific resource - action dialogs only have to return a verdict whose URI identifies the final status of the action’s execution).

- oslc-automation:passed means that the action was executed successfully
- oslc-automation:failed means that the action was attempted but failed
- oslc-automation:canceled means that the dialog was cancelled, whether or not the action was attempted

**Non-normative note:** This interaction pattern is intended to be used where the actions will complete very quickly, otherwise the user is stuck with the delegated UI dialog on the screen waiting for the action to complete.

**Non-normative note:** The dialog displays and gives the user the option to start the action, and should be displayed until the action is completed or cancelled. It may give the option to cancel before the action has started and/or while the action is running. oslc-automation:canceled is used for either of those cases.

**Pattern: Delegated UI dialog for later execution**

This pattern is defined by the OSLC Automation specification: Deferred execution dialog interaction pattern (/wiki/automation/OSLC-Automation-Specification-Version-2.1/#Template-Dialog-interaction-pattern).

- **Interaction pattern** (See Interaction patterns)
- **Identifier:** “pattern-delegated-execution-dialog”
- **Name:** Delegated UI dialog for later execution
- **Contained in profiles:** (None)

**Pattern: Automation Creation Factory**
Resources

This section defines the resources of the Actions specification.

Actions Namespace

The namespace used for resources and properties defined in this specification is as follows:

- Namespace URI: http://open-services.net/ns/core#
- Default Prefix: oslc

Other namespaces

This specification re-uses content from namespaces defined by other specifications. This specification assumes the following mappings from prefixes to namespaces, in addition to those suggested in OSLC Core (http://open-services.net/bin/view/Main/OSLCCoreSpecAppendixA).

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
<tr>
<td>http-methods</td>
<td><a href="http://www.w3.org/2011/http-methods#">http://www.w3.org/2011/http-methods#</a></td>
</tr>
<tr>
<td>oslc-automation</td>
<td><a href="http://open-services.net/ns/auto#">http://open-services.net/ns/auto#</a></td>
</tr>
</tbody>
</table>

Resource: Action

- Name: Action
- Description: An Action provides a representation of an action, process or operation that can be executed against the subject resource, for example start or stop.
- Type URI: http://open-services.net/ns/core#Action

Action Properties

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:title</td>
<td>Exactly-one</td>
<td>Unspecified</td>
<td>XMLLiteral</td>
<td><a href="http://open-services.net/ns/core#Exactly-one">http://open-services.net/ns/core#Exactly-one</a></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>oslc:description</td>
<td>Zero-or-one</td>
<td>Unspecified</td>
<td>XMLLiteral</td>
<td><a href="http://open-services.net/ns/core#Zero-or-one">http://open-services.net/ns/core#Zero-or-one</a></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>rdf:type</td>
<td>One-or-many</td>
<td>Unspecified</td>
<td>Resource</td>
<td>Reference</td>
<td>Any</td>
<td>The resource type URIs. At least one type MUST be oslc:Action.</td>
</tr>
</tbody>
</table>

OSLC Core: Common Properties

- dc:terms:title
- dc:terms:description
- rdf:type

OSLC Core Actions: Start of additional properties

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:binding</td>
<td>Zero-or-many</td>
<td>Unspecified</td>
<td>AnyResource</td>
<td>Either</td>
<td>Any</td>
<td>Instructions for executing the action. It is likely that the target resource will be an <a href="http://open-services.net/ns/core#Binding">http://open-services.net/ns/core#Binding</a></td>
</tr>
<tr>
<td>oslc:executes</td>
<td>Zero-or-one</td>
<td>Unspecified</td>
<td>AnyResource</td>
<td>Either</td>
<td>Any</td>
<td>Link to the future action, if any, that a currently available action realizes. It is likely that the target resource will be an oslc:Action, but that is not necessarily the case. See future actions for example uses.</td>
</tr>
</tbody>
</table>

Action subtypes

The following is an informal, and potentially incomplete, registry of action subtypes:

- oslc:automation:TeardownAction - tears down an already-deployed resource

Before defining new subtypes, please coordinate with the community.

Common Property: oslc:action

In addition to resource definitions, this specification defines one property below that can occur in any RDF resource. This property is defined to be a well-known way of finding available actions on any resource. Resource types may want to specify their own property/ies instead, but this one is defined as a sensible default.

Automation WG proposes that this property be moved to Core’s “common properties” appendix (http://open-services.net/bin/view/Main/OSLCCoreSpecAppendixA) during the finalization phase.

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:action</td>
<td>Zero-or-many</td>
<td>Unspecified</td>
<td>AnyResource</td>
<td>Either</td>
<td>Any</td>
<td>An action that the provider asserts is available for execution by clients at the time the representation was formed. In a distributed system, clients can lose race conditions that result in an “available”-appearing action’s execution requests being rejected. It is likely that the target resource will be an oslc:Action, but that is not necessarily the case; when it is an oslc:Action, it will have at least one binding</td>
</tr>
</tbody>
</table>

XML content.

The resource type URIs.

Title (reference: Dublin Core) of the resource, represented as rich text using XHTML content. SHOULD include only content that is valid and suitable inside an XHTML <div> element.

Descriptive text (reference: Dublin Core) about the resource, represented as rich text using XHTML content. SHOULD include only content that is valid and suitable inside an XHTML <div> element.

The resource type URIs. At least one type MUST be oslc:Action.

Instructions for executing the action. It is likely that the target resource will be an http://open-services.net/ns/core#Action.

Link to the future action, if any, that a currently available action realizes. It is likely that the target resource will be an oslc:Action, but that is not necessarily the case. See future actions for example uses.

Unspecified
Resource: Request

Note: this resource is re-used by this specification, not defined by it. This specification defines the resource shape (how the existing W3C vocabulary (http://www.w3.org/TR/HTTP-in-RDF10/#RequestClass) is re-used and extended).

- Name: Request
- Description: A Request provides a representation of an HTTP request. (Reference: HTTP in RDF)
- Type URI: http://www.w3.org/2011/http#Request (http://open-services.net/ns/core#Request)

Request Properties

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>One-or-many (<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>)</td>
<td>Unspecified</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>The resource type URIs. At least one type MUST be httpRequest.</td>
</tr>
<tr>
<td>HTTP in RDF:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>http:requestURI</td>
<td>Exactly-one (<a href="http://open-services.net/ns/core#Exactly-one">http://open-services.net/ns/core#Exactly-one</a>)</td>
<td>Unspecified</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>HTTP Request-URI for the request. (Reference: HTTP 1.1)</td>
</tr>
<tr>
<td>http:mthd</td>
<td>Exactly-one (<a href="http://open-services.net/ns/core#Exactly-one">http://open-services.net/ns/core#Exactly-one</a>)</td>
<td>Unspecified</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>HTTP method for the request. The target resource SHOULD come from the http-methds vocabulary. Many clients are likely to support only widely used methods such as POST (<a href="http://www.w3.org/2011/http-methods">http://www.w3.org/2011/http-methods</a>).</td>
</tr>
<tr>
<td>http:body</td>
<td>Exactly-one (<a href="http://open-services.net/ns/core#Exactly-one">http://open-services.net/ns/core#Exactly-one</a>)</td>
<td>Unspecified</td>
<td>Either (<a href="http://open-services.net/ns/core#Either">http://open-services.net/ns/core#Either</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>HTTP protocol version for the request. Providers would be wise to support HTTP 1.1 (value: &quot;1.1&quot;) in order to achieve interoperability in practice.</td>
</tr>
<tr>
<td>oslc:usage</td>
<td>Zero-or-many (<a href="http://open-services.net/ns/core#Zero-or-many">http://open-services.net/ns/core#Zero-or-many</a>)</td>
<td>Unspecified</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>Expected usage pattern for the resource. A value of oslc:default can also be used when multiple bindings are present on a single action to hints to clients which is preferred.</td>
</tr>
</tbody>
</table>

OSLC Core Actions: Start of additional properties

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:AuthToken</td>
<td>One-or-many (<a href="http://open-services.net/ns/core#One-or-many">http://open-services.net/ns/core#One-or-many</a>)</td>
<td>True</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>URI of the request. Providers need to expose at least one standard URI for interoperability, but they can expose other more specific ones if needed as long as all URIs on a single result are semantically compatible. It is likely that the target resource will be one of the following, but that is not necessarily the case: oslc:automation:verdict, oslc:ActionDialog, oslc:AutomationRequest, <a href="http://open-service.net/ns/core#verdict">http://open-service.net/ns/core#verdict</a></td>
</tr>
</tbody>
</table>

The HTTP Vocabulary in RDF 1.0 W3C Working Draft (http://www.w3.org/TR/HTTP-in-RDF10/) defines additional content for describing HTTP requests, beyond those listed above; only a subset of HTTP in RDF is used by Core Actions to address known scenarios. Implementations that need additional content SHOULD re-use the W3C vocabulary where possible instead of inventing their own, for example if specific headers are required, but client interoperability may be limited.

Non-normative note: We have talked to the W3C members involved in writing this vocabulary about extending http:requestURI to allow to be a RDF link instead of a string literal whose [string] value happens to be a URI. Currently no new work is being done on the vocabulary, but there were no objections to the usage.

Resource: results

This resource is returned by the delegated UI dialog for immediate execution to indicate the status ("verdict") of the execution of the action.

Note: this resource is re-used by this specification, not defined by it. This specification defines the resource shape (how the existing type (http://open-services.net/ns/core#results) is re-used and extended).

- Name: results
- Description: The results from a delegated user interface (UI) dialog.
- Type URI: http://open-services.net/ns/core#results (http://open-services.net/ns/core#results)

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc-automation:verdict</td>
<td>One-or-many (<a href="http://open-services.net/ns/core#One-or-many">http://open-services.net/ns/core#One-or-many</a>)</td>
<td>True</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>URI of the verdict. Providers need to expose at least one standard URI for interoperability, but they can expose other more specific ones if needed as long as all URIs on a single result are semantically compatible. It is likely that the target resource will be one of the following, but that is not necessarily the case: oslc:automation:verdict, oslc:ActionDialog, oslc:AutomationRequest, <a href="http://open-service.net/ns/core#verdict">http://open-service.net/ns/core#verdict</a></td>
</tr>
</tbody>
</table>

...
This specification defines the resource shape (how the existing type) is re-used and extended).

ParameterInstance Properties

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>One-or-many (<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>)</td>
<td>Unspecified</td>
<td>Resource (<a href="http://open-services.net/ns/core#Resource">http://open-services.net/ns/core#Resource</a>)</td>
<td>Reference (<a href="http://open-services.net/ns/core#Reference">http://open-services.net/ns/core#Reference</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>The resource type URIs. At least one type MUST be oslc:automation:ParameterInstance.</td>
</tr>
<tr>
<td>rdf:value</td>
<td>Exactly-one (<a href="http://open-services.net/ns/core#One-or-many">http://open-services.net/ns/core#One-or-many</a>)</td>
<td>Unspecified</td>
<td>AnyResource (<a href="http://open-services.net/ns/core#AnyResource">http://open-services.net/ns/core#AnyResource</a>)</td>
<td>Either (<a href="http://open-services.net/ns/core#Either">http://open-services.net/ns/core#Either</a>)</td>
<td>Any (<a href="http://open-services.net/ns/core#Any">http://open-services.net/ns/core#Any</a>)</td>
<td>The value of the parameter, either a resource or a typed literal. Untyped literals can be used in place of string-typed literals.</td>
</tr>
</tbody>
</table>

Best Practices

Minimize the number of HTTP round-trips

Actions will often be and/or link to inline resources, possibly anonymous ones (without their own URL, i.e. blank nodes). But they can be separate, addressable resources if needed. Since a client might need to retrieve Action descriptions, e.g. in a UI, in order to find the one with the correct type (if it is looking for a particular type of action), it is likely advantageous to minimize the number of HTTP round trips by returning the representation of all a resource’s actions along with the resource’s representation. The same pattern applies to oslc:binding object resources.

Specification profiles

Specification profiles overview

An specification profile is a coherent subset of this specification. A specification profile includes one or more interaction patterns for which each Action governed by that profile must provide an Action binding. The additional constraints simplify the range of code clients are required to implement, making it cheaper and easier to adopt. This is a specification-only concept, not a runtime concept.

This specification is very general, broad and extensible. As such, writing consumer implementations that support the entire specification might require more effort than would be required to solve the narrower problem(s) of a specific domain’s scenarios. To reduce the standard amount of effort required, we define a number of specification profiles (hereafter, “profiles”), which specify subsets of the specification as a minimum base for interoperability between implementations of this specification.

To be clear, the intention of profiles is to reduce the amount of the specification that consumers need to implement in order to be interoperable with providers. Therefore, it is intended that the profiles restrict the providers (by requiring them to provide, as a minimum, action bindings that conform to certain restrictions) so that consumers can include more assumptions about what they will find.

Profiles are chosen at the time that an provider/consumer is implemented. They inform what subset of this specification is implemented. They are not a runtime concern. Interaction patterns are what clients match action bindings against at runtime; a provider’s design-time choice of supported specification profiles determines which interaction patterns the implementation supports, and hence which clients it can successfully integrate with.

Using specification profiles

Provider implementations SHOULD choose one or more profiles that they will comply with. To “comply with” a profile a provider MUST, for each Action resource served by that provider, provide at least one Action binding that meets the constraints of that profile - as well as meeting any other requirements imposed on a provider by that profile.

Consumer implementations SHOULD choose one or more profiles to support. To “support” a profile a consumer MUST be able to recognize and execute all the interaction patterns included in the profile, and meet any other requirements imposed on a consumer by that profile. Consumers will work best when integrated with providers that comply with at least one of the profiles that they support.

Recognizing profiles

Providers SHOULD state, in their user or integration documentation, which profile(s) they comply with. This information SHOULD include the names of the profiles with hyperlinks or URL references to the specifications in which they were defined. (If only a subset of a provider’s actions comply with a profile, the documentation SHOULD make it clear which subset that is).

Why are profiles only identified in documentation, not data?

This section is non-normative.

Matching profiles is intended to be an implementation-time and integration configuration problem. When a user is deciding if two different components are compatible with each other, they will look at the documentation and see that a consumer supports a given set of profiles of the OSLC Actions specification. They will then look at the providers that they are considering using and see which profiles of this specification that they comply with. They should only expect the integrations to be successful where there is a match of at least one profile between the consumer and provider involved. The identifiers provided for the profiles are intended to allow the profiles to be referred to unambiguously in product documentation.

It is possible that implementations that do not share a common profile may still work together, at least for some of their actions. For example, a provider may have some actions that have very simple Action bindings and some that have more complex ones. The simpler Action bindings may, on inspection, be seen to adhere to the restrictions of a given profile, even if that was not the intention of the provider. A consumer that supports that profile would then be able to execute the actions with the simpler Action bindings, even though the provider had not targeted itself at that profile. For this reason the profiles are not identified in the data, but instead implementations have to look at the Actions available and see if they comply with the interaction patterns and other restrictions that they support.

In other words, compatibility between a consumer and a given Action binding is based on the interaction pattern being used and any subset or restriction of that pattern that the consumer is compatible with. Profiles merely guide consumers as to which interaction patterns (or restrictions thereof) they should implement, and to encourage providers to produce Action bindings that do not require complex consumers.

Choosing one or more profiles to implement

When implementing a consumer or a provider, profile(s) SHOULD be chosen by considering the providers or consumers (respectively) that the implementation is to integrate with. This can be done by looking at specific implementations that already exist, or by using the profile identified by a domain specification that the implementations share, or by considering the likely capabilities of the implementations to be integrated.

Most implementations of this specification will occur in the context of another specification, for a specific domain. In those cases, implementations SHOULD choose, as a minimum, the profile(s) required by that domain specification.

Other specifications
When these restrictions are included in a specification profile, the providers MUST comply with them. This section is here for specification profiles to conveniently restrict http:Request resources for simple specification profiles. Standard restrictions on http:Request resources for simple specification profiles in OSLC Automation specification profile definitions

Profile: POST RDF described by a OSLC Resource Shape to the Action resource

- Specification profile (See Specification profiles)
- Identifier: "profile-action-shape-post"
- Name: POST RDF described by a OSLC Resource Shape to the Action resource
- Contains: HTTP request with empty body ("pattern-http-empty-body"), HTTP request with Resource Shape to describe the request body ("pattern-http-resource-shape")

To meet the constraints of this profile, Action bindings MUST use one of the following Core Actions' interaction patterns:

- HTTP request with empty body ("pattern-http-empty-body")
- HTTP request with Resource Shape to describe the request body ("pattern-http-resource-shape")

The following additional constraints are imposed on action bindings, regardless of the interaction pattern used:

- The oslc:requestURI object MUST be the URL of the action resource that links to it.
- The http:method object MUST be http:methods:POST.
- The bindings MUST adhere to the standard restrictions on http:Request resources for simple specification profiles in Appendix A: Constructing HTTP requests from http:Request resources.

Known adopters of this profile:

- OSLC Change Management 3.0 example ([http://open-services.net/wiki/change-management/State-Transitions-for-Change-Requests#CM-Actions-with-the-Core-Workgroup-Actions-Proposal](http://open-services.net/wiki/change-management/State-Transitions-for-Change-Requests#CM-Actions-with-the-Core-Workgroup-Actions-Proposal)) (Note: the number 3.0 is a current best-guess; it might theoretically change)

Profile: Create an Automation Request

- Specification profile (See Specification profiles)
- Identifier: "profile-action-shape-post"
- Name: Create an Automation Request
- Contains: Automation Request ("pattern-automation-request")

To meet the constraints of this profile, action bindings MUST use the following Core Actions' interaction pattern:

- Automation Request ("pattern-automation-request")

The following additional constraints are imposed on action bindings:

- The bindings MUST adhere to the standard restrictions on http:Request resources for simple specification profiles in Appendix A: Constructing HTTP requests from http:Request resources.

Known adopters of this profile:


Appendix A: Constructing an HTTP request from an http:Request resource

HTTP requests are described in RFC2616 ([http://www.w3.org/Protocols/rfc2616/rfc2616.html](http://www.w3.org/Protocols/rfc2616/rfc2616.html)) and other documents referenced by that one, and any documents that later supersede it.

The HTTP Vocabulary in RDF ([http://www.w3.org/TR/HTTP-in-RDF/](http://www.w3.org/TR/HTTP-in-RDF/)) provides a mapping between RDF terms and HTTP concepts. (This specification takes that vocabulary with one change, as follows: the value type of the requestURI property is expected to be an RDF URI, not a string literal).

This appendix is an interpretation of how to use those two documents together to construct an HTTP request based on an http:Request resource. It is expressed in normative language.

As specification profiles can require that providers offer action bindings that don’t use some of these features, then consumers MAY decide not to implement all of these features. However, if a consumer has not implemented on these features then they MUST NOT attempt to use a binding that uses that feature. For example, if a consumer has not implemented support for the http:headers property, then it MUST NOT attempt to construct and submit an HTTP request for a binding that contains that property.

When constructing and sending an HTTP request based on an http:Request object, a consumer MUST:

- use the HTTP version specified by the http:method property.
- use the HTTP method specified by the http:method property.
- use the request URI specified by http:requestURI.
- include the headers specified by the http:headers property, if present.

The interaction patterns referencing this appendix will specify how to construct the request body.

Ordinarily the response to this request will be one of the following, although any status code allowed by HTTP is permitted. Consumers should note that status codes are extensible, and hence HTTP requires applications to understand the generic 1xx-5xx status code classes ([http://tools.ietf.org/html/rfc2616#section-6.1.1](http://tools.ietf.org/html/rfc2616#section-6.1.1)), even if it also checks for specific status codes it needs to take action on, in order to gracefully handle the introduction of new status codes in the future.

- **200 (OK)** to indicate that the action has completed
- **201 (Created)** to indicate that the request resulted in the creation of a new resource; the Location response header provides the URL of the newly created resource.
- **202 (Accepted)** to indicate that the request has been queued and will be dealt with at a later time, at which point it might succeed or fail. The HTTP specification ([http://tools.ietf.org/html/rfc2616#section-10.2.3](http://tools.ietf.org/html/rfc2616#section-10.2.3))] suggests returning a monitor to track the status of the execution, but provides no standard definition for such a monitor. If OSLC providers use the 2xx status code, they can consider using the OSLC Automation Specification's mechanisms, so that clients have a known pattern to code for.
- **4xx** or **5xx** status code to indicate failure.

Depending upon the interaction pattern within which a request is made, the HTTP status code might or might not give the status of the consumer’s desired result (successfully completing an action). For example, the interaction patterns might require the consumer to use the OSLC Automation specification’s mechanisms to monitor the action’s progress and success/failure ([http://open/services.net/wiki/automation/OSLC-Automation-Specification-Version-2.0#State-and-Verdict-properties](http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.0#State-and-Verdict-properties)).

Standard restrictions on http:Request resources for simple specification profiles

This section is here for specification profiles to conveniently restrict http:Request resources to simple requests, without having to repeat these restrictions in every profile that uses them. These restrictions only apply to specification profiles that explicitly reference this section. Implementations not governed by a specific profile, or governed by profiles that do not include these restrictions, can ignore this list.

When these restrictions are included in a specification profile, the providers MUST, for each action linked to from an oslc:action predicate, provide at least one binding that:
Appendix B: Examples

This section is non-normative.
The examples are on a separate page: Actions 2.0 Examples (http://open-services.net/wiki/core/Actions-2.0-Examples)

Appendix D: Notices and References

Contributors
- John Arwe (http://open-services.net/bin/view/Main/JohnArwe) (IBM, OSLC-Core)
- Martin Pain (IBM, OSLC-Automation Lead)
- Samuel Padgett (IBM, OSLC-Change-Management Lead)
- SteveSpeicher (http://open-services.net/bin/view/Main/SteveSpeicher) (IBM, OSLC-Core-Lead)

Reporting Issues on the Specification

The working group participants who author and maintain this working draft specification, monitor a distribution list where issues or questions can be raised, see Core Mailing List (http://open-services.net/mailman/listinfo/oslc-core.open-services.net)

Also the issues found with this specification and their resolution can be found at Core 2.0 Issues (http://open-services.net/wiki/core/OSLC-Core-V2-Issues/).

License and Intellectual Property

We make this specification available under the terms and conditions set forth in the site Terms of Use (http://open-services.net/terms/), IP Policy (http://open-services.net/ip-policy/), and the Workgroup Participation Agreement for this Workgroup (http://open-services.net/legal-agreements/performance-monitoring-wpa/).

References

- HTTP in RDF - HTTP Vocabulary in RDF 1.0 (http://www.w3.org/TR/HTTP-in-RDF10/)
- OSLC Core - OSLC Core Specification 2.0 (http://open-services.net/bin/view/Main/OSLCCoreSpecification)
- Dublin Core 1.1 - Dublin Core Metadata Element Set, Version 1.1 (http://dublincore.org/documents/2010/10/11/docs/)
- FOAF - Friend of a Friend (FOAF) v0.98 (http://xmlns.com/foaf/spec/20100809.html)
- JSON - JSON (http://www.json.org/)
- OAuth 1.0a - RFC5849 - The OAuth 1.0 Protocol (http://tools.ietf.org/html/rfc5849)
- XML Namespaces - Namespaces in XML 1.0 (Third Edition) (http://www.w3.org/TR/REC-xml-names/)
- Linked Data Platform - Linked Data Platform 1.0 (http://www.w3.org/TR/lpdp/)

Appendix E: Changes

Changes during 2.0 convergence phase

- Addressed comments from May 15, 2014 meeting (/wiki/automation/AutomationMeetings20140515/#Minutes)
- 2014-07-17: Added short future actions section to address convergence feedback
- 2014-09-11: Align with Core 2 instead of anticipating Core 3 when requiring media types (RDF/XML not Turtle)
- 2014-09-11: Editorial changes in response to Ian’s review, see July+August meeting minutes for details
- 2014-09-11: Align with Core 2 instead of anticipating Core 3 when requiring media types (RDF/XML not Turtle)
- 2014-07-17: Added short future actions section to address convergence feedback
- 2014-09-12: Remove “cherry picking”, whereby a single binding could have >1 occurrence of predicates like httpVersion or body and thus match multiple interaction patterns
- 2014-09-12: Add oslc:usage 0:* in binding resource table to match existing prose
- 2014-10-09: Editorial changes
- 2014-10-10: Move futureAction and executes predicates from Automation to Core vocabulary
- 2015-01-28: Corrected reference to a domain specification Action type in Automation 2.1 future actions

Changes to OSLC Core vocabulary

These are the additions that this specification makes to the OSLC Core Vocabulary (/wiki/core/CoreVocabulary/).

- Predicates
  - oslc:binding. Description: The binding of some abstract action (the semantics of a process or operation) onto a more concrete interaction pattern or endpoint(s). For example, it can link an oslc:Action to a description of the messages (e.g. HTTP) that must be exchanged in order to perform the action. In OSLC Core Actions it is expected that the object will be of type http:Request, oslc:Dialog, or oslc:CreationFactory, but this is not necessarily the case.
  - oslc:creation. Description: A link from any resource to an action that is available when the response is formed to execute on that subject resource. It is expected that the object will be of type oslc:Action, and will contain at least one oslc:binding predicate, but this is not necessarily the case.
  - oslc:execute. Context: Automation 2.1 future actions section. Description: Link from a currently available action to its corresponding future action.
  - oslc:execute. Context: Automation 2.1 future actions section. Description: Link from an action that will become available at a future time, possibly on another resource. For example, a future action on an Automation Plan can become available after a corresponding Automation Request is created, or after it completes; or a resource shape can use it to describe actions that clients can expect to be available on resources that the shape applies to, for example those created by a particular creation factory.
  - oslc:finalStatusLocation. Description: The location at which the final status of a process, operation or action will be available. The meaning of any particular value of this predicate must be defined by an appropriate specification. For example, the OSLC Core Actions specification defines that if the value of this predicate is the URI http:StatusCode then the final status is to be taken from the HTTP response status code of a request that is described by the subject resource. OSLC Core Actions also defines the meaning of oslc:Dialog, oslc:Action and oslc:Automation:ExecutionResult in this context.

- Classes:
  - oslc:Action. Description: An Action provides a representation of an action, process or operation that can be executed against the subject resource, for example start or stop.

- Individuals:
  - oslc:ActionDialog. Description: An individual that represents that a dialog is to be used (oslc:usage) as a dialog to execute an oslc:Action. When the dialog is completed, the action’s execution has been completed. (This is the added value on top of merely identifying a resource as rdf:type oslc:Dialog. This is in contrast to oslc-automation:DeferredExecutionDialog, which can be used as part of executing an action, but after the dialog completes the consumer has more steps to take.) The OSLC Core Actions specification defines additional status information that can be returned by dialogs used in this way.