Appendix C: Guidance on Links & Relationships

This Version
- [http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks](http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks)

Latest Version
- [http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks](http://open-services.net/bin/view/Main/OslcCoreSpecAppendixLinks)

Authors
- Dave Johnson

Contributors
- The [OSLC Core Specification Workgroup](http://open-services.net)

Table of Contents

**OSLC Core Specification**

By: the OSLC Core Workgroup

- [OSLC Core Specification](#)
  - Relationships in OSLC
  - Expressing relationships for OSLC
    - 1) Link
      - Don't make assumptions about the target of links
    - 2) Anchor
      - Anti-pattern: Link with inlined values of target
  - References

In this informative appendix we offer advice to OSLC workgroups on modeling links and relationships between OSLC resources.

**Relationships in OSLC**
The Core workgroup's guidance to OSLC domain workgroups is that relationships are uni-directional and in most cases relationships should be modeled as links. We offer two ways to express relationships, a link and an anchor. Let's define those terms and others that we will use in this document:

- **Relationship**: a relationship is said to exist between two resources if there is something that connects them; they work together, belong together, are similar or should be considered together. There may be different types of relationships.
- **Link**: a link is a URI reference from one resource, the **subject** or source, to another resource, said to be the **object** or target. In RDF and in OSLC we use links to model relationships and like relationships, links are uni-directional.
- **Anchor**: an anchor is a mechanism for attaching property-values to a link, thus annotating a relationship with values that are about that relationship.

Now that we've defined our terminology, let's move on to expressing relationships.

### Expressing relationships for OSLC

The RDF data model gives us a number of value-types to express relationships, some more complex than others. We expect that the majority of links will be expressed with the most simple option, URI Reference or as it is known in conversation, a Link. For more complex cases, when a relationship must be annotated with property-values, we use what is called an Anchor, which is a link plus information that annotates or labels that link. Read the sub-sections below for details and examples of Links and Anchors.

1) **Link**

**Link from subject to object.**

Use this when you need a simple link, you don't want to inline any summary of the target and you do not need to annotate the link with property values. Most relationships should be represented this way.

In RDF terminology, a link is called a URI Reference. In OSLC, you express a link as a property with OSLC value-type of **Resource** and OSLC representation of **Reference**. For example, here is a property `terms:subscribesTo` that you might find inside a customer resource, it links to a resource that is "subscribedTo" by the customer:

**Example #1: relationship expressed as simple link (i.e. URI reference)**

Link to validation, triples and graph: [RDF Validation](#)
Don't make assumptions about the target of links

Relationships in OSLC resources are at their simplest an RDF property whose object is a URI. In some cases within one system, it is necessary to require a closed link, i.e. require the object of a link be of one or more specific resource types. In general, it is better to be open like the web and not place restrictions on the type of resource linked to. The property's purpose and name should clearly reflect the scenarios it is supporting. Since the usage of these relationship properties may exist for a long period of time, specification authors should use great care in determining these.

As resources evolve over time, and they adapt to different situations, different types will be exposed as targets to existing link types. Well behaved clients should gracefully handle resource types they don't expect when exercising links in resources. Well behaved clients should gracefully handle resource types they don't expect when exercising links in resources. Specifications should allow links to be open ended (have any type, specifying Range any), and use text in the property description to suggest expected types, without being limiting. For example: “Change request affects a plan item. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.” Some implementations may only work well if the link object comes from a set of “known” or “expected” types. The following graceful degradation sequence is suggested for providers when an unexpected type is encountered:

1. If possible, tolerate the unexpected link target type and allow the client’s request to proceed normally.
2. Else, if possible, allow the client’s request to proceed and include an informational message with the response.
3. Else, fail the client’s request with the most appropriate 4xx Bad Request HTTP status code.

2) Anchor

Local Resource holds link from subject to object plus property-values that annotate the relationship.

Relationships are expressed as RDF triples and triples do not have property values - only resources have property values. Links are used to express relationships. Many relationships can be represented by a single link, but some relationships need to be annotated with property values. For example, if I am trying to represent the subscriber relationship between a customer and a magazine, I may need to record an expiration date for the relationship. To do this, we need to use an RDF concept known as reification. Reification is a way to make a statement about a statement. See RDF Syntax Specification Section 4: Statements about Statements for more information on this concept (reference: RDF Syntax).
Assuming the subscriber scenario, example #2 shows how you would use reification to express the annotation.

**Example #2: Link as reified statement in RDF/XML format**

Link to validation, triples and graph: [RDF Validation](#)

```xml
<rdf:RDF
    xmlns:terms="http://example.com/terms/
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <terms:Customer rdf:about="http://example.com/customers/4321">
        <terms:subscribesTo rdf:resource="http://example.com/magazines/Field_and_Stream" />
    </terms:Customer>

    <terms:Customer rdf:about="http://example.com/customers/4321">
        <terms:subscribesTo rdf:resource="http://example.com/magazines/Cat_Fancy" />
    </terms:Customer>

    <rdf:Statement rdf:about="#n1">
        <terms:expirationDate>2010-06-03</terms:expirationDate>
        <terms:annualPriceUSD>23.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/online" />
        <rdf:subject rdf:resource="http://example.com/customers/4321"/>
        <rdf:object rdf:resource="http://example.com/magazines/Field_and_Stream"/>
        <rdf:predicate rdf:resource="http://example.com/terms/subscribesTo" />
    </rdf:Statement>

    <rdf:Statement rdf:about="#n2">
        <terms:expirationDate>2010-01-22</terms:expirationDate>
        <terms:annualPriceUSD>15.95</terms:annualPriceUSD>
        <terms:delivery rdf:resource="http://example.com/terms/mail" />
        <rdf:subject rdf:resource="http://example.com/customers/4321"/>
        <rdf:object rdf:resource="http://example.com/magazines/Cat_Fancy"/>
        <rdf:predicate rdf:resource="http://example.com/terms/subscribesTo" />
    </rdf:Statement>

</rdf:RDF>
```

But there is a better RDF/XML form, made possible by the rdf:ID attribute. OSLC implementations should prefer this next form because it is easier to read and easier to parse.

Link to validation, triples and graph: [RDF Validation](#)

**Example #3: Link as reified statement in RDF/XML format (preferred form)**
And here’s how to express the same within the JSON format specified by OSLC Core. Because we already represent value-type Resource as a JSON object, we can simply add annotating property-values directly into that. In this case, adding `terms:expirationDate` to the `terms:subscribesTo` object.

**Example #4: Link as reified statement in OSLC JSON format**
Anti-pattern: Link with inlined values of target

With the OSLC query syntax it is possible to get query resources with links and for each link some inlined values from the target of the link. This allows a client to get links and some information from the target of each link in one HTTP request per page of results, saving the client the expense of dereferencing each link. This pattern is useful for query, but is not recommended in OSLC resource design.

Below is an example of a Blog Entry resource containing a link to an uploaded-file resource of type `oslc_blog:UploadedFile` and with the `dcterms:title` of that resource inlined. This enables the client to know the type and title of the target resource without having to HTTP GET it.

**Anti-pattern Example: a link with inlined values of target**
In OSLC defined resource terminology, the property above is type Resource with representation Inline. The problem with this is that it adds redundant property-values to a resource. When you link to a target resource that has a `dcterms:title` value and you inline that `dcterms:title` in the subject resource, in the RDF data model this means that you have just given the object resource two titles. What if the titles do not match? Can a client change the title in the subject resource?

That brings us to the end of the guidance. Please provide feedback to the OSLC Core Workgroup mailing-list:

- Mailing List Registration
- Mailing List Archives

References
