Open Services for Lifecycle Collaboration Core Specification Version 2.0 Query Syntax

This Version

- http://open-services.net/bin/view/Main/OSLCCoreSpecQuery

Latest Version

- http://open-services.net/bin/view/Main/OSLCCoreSpecQuery

Authors

- Arthur Ryman

Contributors

- The OSLC Core Specification Workgroup

Table of Contents

Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in RFC-2119. This document is a mixture of normative and informative text. See the OSLC Core Specification Glossary below for definitions of these terms.

Overview

This document defines a standard set of OSLC query parameters that other OSLC domain specifications **MAY** use to query

resources.

Graph Patterns

The OSLC standard query parameters define *graph patterns* that are matched against the resources managed by the service, which we view as comprising an RDF graph. The pattern matching process begins at some *starting subject resource* and matches a set of triples in the RDF graph (see also RDF Concepts and Abstract Syntax: Graph Data Model). The starting subject resources used for pattern matching are determined by the the Query Capability. A Query Capability defines a starting subject resource, namely the base URI itself (see `oslc:queryBase` in the OSLC Core Specification - Query Capability). The result of the query is the set of all triples matched by the pattern, beginning the matching processing with the starting subject resource defined by the Query Capability.

The full graph pattern defined by the query parameters is composed of two kinds of graph pattern, which we refer to as *property tree patterns* and *member list patterns*. These patterns **MAY** be used individually or **MAY** be combined in a single query, but one or the other **MUST** be present. The result of matching the full graph pattern is the union of the results of matching each kind of pattern.

Property Tree Patterns

A property tree pattern is defined by the presence of the `oslc.properties` query parameter. The data model of a resource is viewed as consisting of a set of properties of the resource. The values of these properties are either literal values, such as numbers, strings, and dates, or links to other resources. Linked resources may link to other resources, which may have their own properties, and so on, to arbitrary depths. However, the scope of the query **MAY** be limited to the boundaries of the service as defined by the appropriate domain specification. Properties of linked resources may be viewed as nested properties of the initial resource, where a nested property is a property of an object of a property of the starting resource. The `oslc.properties` query parameter lets you match both the immediate and nested properties of a starting subject resource.

Member List Patterns

A member list pattern is defined by the presence of at least one of the following query parameters: `oslc.where`, `oslc.searchTerms`, `oslc.orderBy`, and `oslc.select`. In this pattern, the starting subject resource is viewed as a container of other resources. The members of the container are defined by one or more multi-valued membership properties of the starting subject resource. The membership properties are given by the the properties described as membership properties (via `oslc:isMemberProperty`) in the Resource Shape of the starting subject resource.The pattern matching process is summarized by the following sequence of steps, and fully defined in later sections. Default processing occurs in steps where the corresponding query parameter is not present:

1. `oslc.where` - filter the member list, keeping only those member resources that satisfy the boolean test on the member resource properties
2. `oslc.searchTerms` - score each member resource using a full text search on it text-valued properties, and sort them in descending order of score
3. `oslc.orderBy` - sort the members using the sort keys, using the full text search score as the primary key
4. `oslc.select` - match the immediate and nested properties of the member resources (like in `oslc.properties`)

Prefixed Names

In addition to the preceeding query parameters, we also define `oslc.prefix` which is used to define the prefixes for prefixed names that appear in the other query parameters. The `oslc.prefix` query parameter is not itself part of the graph pattern. Other specifications MAY use `oslc.prefix` to define prefixes for prefixed name used other query parameters. A Prefixed Name is an abbreviation for a URI as defined in he SPARQL Query Language (reference: SPARQL).

Query Parameters

The query parameters defined here are intended to satsify simple query requirements, and be easy to both use and implement. More complex query requirements should be satisfied using a full query language, e.g. SQL or SPARQL.

This specification formally defines the syntax of the query parameters using Extended Backus-Naur Form (EBNF) and informally illustrates their semantics with simple examples. For an additional discussion of the semantics of the query parameters see OSLC Simple Query Semantics V1 which is a non-normative informational reference. If there is any disagreement between that reference and this specification, then this specification takes priority.

The query parameters MAY contain characters that MUST be URL encoded when transmitted in an HTTP request. The examples below show unencoded query parameters for clarity. See URL Encoding for a further discussion of this topic.

This specification defines the syntax of a set of related query parameters that MAY be used to perform queries. We have adopted a consistent naming convention for these query parameters to identify them as common across all OSLC specifications, namely all query parameter names are prefixed with the characters " `oslc.` " The following sections define these query parameters.

An OSLC domain specification MAY use some or all of these query parameters, and SHOULD use these rather than defining new query parameters that have the same or very similar meanings.

Each of these query parameters SHOULD appear at most once in a query. The behavior is undefined when a query parameter appears more than once.

In the following sections, syntax is formally defined using a common extended form of BNF. Informal definitions and comments are delimited by /* and */.

oslc.where

This query parameter defines part of a member list pattern. If a member list pattern is defined but this query parameter is not present then it defaults to the boolean condition `true`, i.e. it matches all members of the member list.

A resource may have an associated set of related resources defined by one or more membership properties. Service consumers often need to search for subsets of these related resources that satisfy certain conditions. The `oslc.where` query parameter lets you specify the conditions that these related resources must satisfy. It is like the `WHERE` clause of a SQL statement.

For example, suppose that the following URL represents the set of all bug resources managed by a service, and that is has a single membership property, `cm:memberBug`:

```
http://example.com/bugs
```

Suppose bug reports have a `dcterms:identifier` property that gives their bug number, e.g. `"4242"`. The following URL filters this set and selects only those bugs that have a `dcterms:identifier` property whose value is `"4242"`:

```
http://example.com/bugs?oslc.where=dcterms:identifier="4242"
```

Note that the preceding query there is only one membership property, namely `cm:memberBug`.

Conditions may use the usual binary comparison operators and be combined using the boolean conjunction operator, " `and` ". For example, suppose bugs have `cm:severity` and `dcterms:created` properties. The following example finds high severity bugs created after April 1, 2010:

```
http://example.com/bugs?oslc.where=cm:severity="high" and dcterms:created>"2010-04-01"
```

In this example, the ISO 8601 date format is used since the `xsd:dateTime` format is recommended for use with the `dcterms:created` property. In general, the format used for a property depends on the definition of the property.

The following URL illustrates the use of nested properties. It finds the set of bugs created by John Smith:

```
http://example.com/bugs?oslc.where=dcterms:creator{foaf:givenName="John" and foaf:familyName="Smith"}
```

Property values will often refer to or be other resources (in RDF/XML the `rdf:resource` attribute is used to identify the resource). Suppose bugs are linked to testcases using a property name `qm:testcase` where the prefix `qm:` stands for the URI `http://qm.example.com/ns`. The following query finds all bugs that are linked to testcase `http://example.com/tests/31459`:

```
http://example.com/bugs?oslc.prefix=qm=<http://qm.example.com/ns>&
        oslc.where=qm:testcase=<http://example.com/tests/31459>
```

Syntax

The syntax of the `oslc.where` query parameter is defined by the `oslc_where` term in the following BNF grammar:

```
oslc_where    ::= "oslc.where=" compound_term
compound_term ::= simple_term (space? boolean_op space? simple_term)*
simple_term   ::= term | scoped_term
space         ::= " " /* a space character */
boolean_op    ::= "and"
term          ::= identifier_wc comparison_op value | identifier_wc space in_op space? in_val
scoped_term   ::= identifier_wc "{" compound_term "}"
identifier_wc ::= identifier | wildcard
identifier    ::= PrefixedName
PrefixedName  ::= /* see "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/#rPrefixedName */
wildcard      ::= "*"
comparison_op ::= "=" | "!=" | "<" | ">" | "<=" | ">="
in_op         ::= "in"
in_val        ::= "[" value ("," value)* "]"
value         ::= uri_ref_esc | literal_value
uri_ref_esc   ::= /* an angle bracket-delimited URI reference in which > and \ are \-escaped. */
literal_value ::= boolean | decimal | string_esc (LANGTAG | ("^^" PrefixedName))?
boolean       ::= "true" | "false"
decimal       ::= /* see "XML Schema Part 2: Datatypes Second Edition", http://www.w3.org/TR/xmlschema-2/ */
string_esc    ::= /* a string enclosed in double quotes, with certain characters escaped. See below. */
LANGTAG       ::= /* see "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/#rLANGTAG */
```

wildcard

As in the case of `oslc.properties`, the `wildcard` matches any property.

boolean_op

The `boolean_op` term represents a boolean operation that lets you combine simple boolean expressions to form a compound boolean expression.

The only boolean operation allowed is " `and` " which represents conjunction. The boolean operator " `or` " for disjunction is not allowed in the interests of keeping the syntax simple. The effect of " `or` " in the simple case of testing a property for equality with one of several values can be achieved through the use of the " `in` " operator. For example, the following query finds bugs with severity " `high` " or " `medium` ":

```
http://example.com/bugs?oslc.where=cm:severity in ["high","medium"]
```

space

The `space` term represents a single space character. A space character **MAY** be used to delimit the `binary_op` term in the `compound_term` term to improve readability.

comparison_op

The `comparison_op` term represents one of the following binary comparison operators:

=	test for equality
!=	test for inequality
<	test less-than
>	test greater-than
<=	test less-than or equal
>=	test greater-than or equal

Semantics of datatypes and operations on these datatypes **MUST** work as defined in the SPARQL Query Language (reference: SPARQL).

in_op

The `in_op` term represents the operator " `in` " which is a test for equality to any of the values in a list. The list is a comma-separated sequence of values, enclosed in square brackets, whose syntax is defined by the term `in_val`.

value

The `value` term represents either a URI reference (`uri_ref_esc`) or a literal value (`literal_value`).

literal_value

The `literal_value` term represents either a plain literal or a typed literal.

A plain literal is a quoted string (`string_esc`), optionally followed by a language tag (`LANGTAG`). For example, `"Bonjour"@fr` is a plain literal with a language tag for French. If the range of a property is National Language strings and no language tag is provided in a literal value, then the service **SHOULD** infer the language tag from the HTTP request or provide a default value.

A typed literal is formed by suffixing a quoted string (`string_esc`) with "^^" followed by the prefixed name (`PrefixedName`) of a datatype URI. If the range of a property includes literal values from more than one datatype, then a typed literal **MUST** be used in order to avoid ambiguity. Otherwise a plain literal **MAY** be used and the service **SHOULD** infer the datatype.

The terms `boolean` and `decimal` are short forms for typed literals. For example, `true` is a short form for `"true"^^xsd:boolean`, `42` is a short form for `"42"^^xsd:integer` and `3.14159` is a short form for `"3.14159"^^xsd:decimal`.

decimal

The `decimal` term represents a decimal number as defined in XML Schema Part 2: Datatypes Second Edition. As mentioned above, this term is a short form for typed literals whose datatype URIs are either `xsd:integer` or `xsd:decimal`. An integer literal value is a special case of a decimal literal value, namely one in which the decimal point is omitted from the lexical representation. For example, `42` is a valid decimal number which happens also to be a valid integer and so it is a short form for the typed literal `"42"^^xsd:integer`.

`string_esc`

The `string_esc` term represents an arbritrary sequence of characters. The sequence of characters is enclosed in double quote (") characters. Therefore, the double quote character itself **MUST** be escaped. More generally, all occurrences of the double quote character in the string **MUST** be replaced by the sequence \" and all occurrences of the backslash character \ in the actual value **MUST** be replaced by the sequence \\ in the escaped value. This escaping process **MUST** be reversed to give the actual value of the string.

oslc.searchTerms

This query parameter defines part of a member list pattern. If a member list pattern is defined but this query parameter is not present then the matching process assigns the same effective score of 0 to each member of the list and does **NOT** change the order of the member list

Resource properties often contain text so it is useful to search for resources that contain specified terms. The `oslc.searchTerms` query parameter lets you perform a full text search on a set of resources. In a full text search, each resource is matched against the list of search terms and assigned a numeric score. A high score indicates a good match. The matching resources are returned in the response, sorted by the score in descending order. Each resource that is returned in the response is annotated with a special property, `oslc:score`, that gives its match score.

An OSLC domain specification that supports full text search **SHOULD** specify which resource properties are indexed so that search results are consistent across implementations.

When `oslc.searchTerms` is used in the request, each matching resource (hit) in the response MAY contain an `oslc:score` property. Note that `oslc:score` is not purely a property of the resource since it also depends on the search terms. It is therefore a pseudo-property whose validity is limited to the HTTP response.

The `oslc:score` property **MUST** be a non-negative number and **SHOULD** be in the range from 0-100. Results **MUST** be ordered with the entry with the largest `oslc:score` occurring first.

The `oslc.orderBy` query parameter **MAY** be used with `oslc.searchTerms`. When `oslc.orderBy` is used with `oslc.searchTerms` the result **MUST** be first sorted in descending order by the `oslc:score` pseudo-property, and then by the other sort keys specified in `oslc.orderBy`. This behavior is like prepending the list of sort keys specified in `oslc.orderBy` with the key `-oslc:score`. However, the pseudo-property `oslc:score` **MUST NOT** appear explicitly in `oslc.orderBy`.

The `oslc.where` query parameter **MAY** be used with `oslc.searchTerms`. When `oslc.where` is used with `oslc.searchTerms` then the set of resources searched for matches **MUST** be restricted to only those resources that satisfy the conditions in `oslc.where`. For example, the following query returns the high severity bugs that deal with database performance:

```
http://example.com/bugs?oslc.where=cm:severity="high"&oslc.searchTerms="database","performance"
```

Syntax

The syntax of the `oslc.searchTerms` query parameter is defined by the `oslc_searchTerms` terms in the following BNF grammar:

```
oslc_searchTerms ::= "oslc.searchTerms=" search_terms
search_terms     ::= string_esc ("," string_esc)*
```

oslc.orderBy

This query parameter defines part of a member list pattern. If a member list pattern is defined but this query parameter is not present then there are no sort keys and the matching process does **NOT** change the order of the member list.

The `oslc.orderBy` query parameter lets you sort the result set. It is like the `ORDER BY` clause of a SQL statement.

You can specify a list of one or more immediate or nested properties of the resources in the member list, and a sort direction for each where " `+` " means ascending order and " `-`" means descending order. The following example sorts the high severity bugs by the family and given names of the creator, with most recently created first:

```
http://example.com/bugs?oslc.orderBy=dcterms:creator{+foaf:familyName,+foaf:givenName},-dcterms:created
    &oslc.where=cm:severity="high"
```

The properties in the `oslc.orderBy` list are sort keys. The member list is sorted by the sort keys, in the indicated direction. The sorting order of the property values **MUST** be the same as that used for evaluating the binary comparison operators in the `oslc.where` query parameter.

Each sort key **SHOULD** be the name of a single-valued property of the each resource in the result set. The sorting behavior is undefined if the sort key properties are not single-valued.

Syntax

The syntax of the `oslc.orderBy` query parameter is defined by the `oslc_orderBy` term in the following BNF grammar:

```
oslc_orderBy        ::= "oslc.orderBy=" sort_terms
sort_terms          ::= sort_term ("," sort_term)*
sort_term           ::= scoped_sort_terms | ("+" | "-") identifier
scoped_sort_terms   ::= identifier "{" sort_terms "}"
```

oslc.select

This query parameter defines part of a member list pattern. If a member list pattern is defined but this query parameter is not present, then this (absent) criterion does not affect the list of members returned.

The `oslc.select` query parameter lets you specify which immediate and nested properties of the resources in the member list to match. It is like the `SELECT` clause of a SQL statement.

The syntax of the `oslc.select` query parameter is the same as that of the `oslc.properties` query parameter. However, the property names that appear in the `oslc.select` query parameter are those that belong to the resources in the member list as opposed to those that belong to the starting subject resource.

For example, the following URL finds the high severity bugs and includes their creation date and the family name of the creator in the HTTP response:

```
http://example.com/bugs?oslc.select=dcterms:created,dcterms:creator{foaf:familyName}&oslc.where=cm:severity="high"
```

Syntax

The `oslc.select` query parameter is defined by the term `oslc_select` in the following BNF grammar:

```
oslc_select ::= "oslc.select=" properties
properties  ::= property ("," property)*
property    ::= identifier | wildcard | nested_prop
nested_prop ::= (identifier | wildcard) "{" properties "}"
```

The `oslc.select` query parameter uses the same syntax as the `oslc.properties` query parameter. The difference between them is in the meaning of the identifiers. In `oslc.properties`, the identifiers are the names of properties of the starting subject resource. In `oslc.select`, the starting subject resource has an associated list of member resources, and the identifiers are the names of properties of the resources that are contained in the member list.

For example, suppose that a bug resource has the membership property `cm:comment` that relates it to comment resources, and that the bug and each comment have `dcterms:modified` properties that give their last modification dates. The following query specifes that the response should include the modification date of bug 4242 and the modification date of each of its comments:

```
http://example.com/bugs/4242?oslc.properties=dcterms:modified&oslc.select=dcterms:modified
```

wildcard

As in the case of `oslc.properties`, the `wildcard` is equivalent to the list of all properties of the member resources. The query parameter `oslc.select=*` specifies that pattern **MUST** match all the properties of the resources in the member list.

URL Encoding

The query parameter syntax defined in this specification permits the use of characters that **MUST** be properly encoded when transmitted in HTTP requests. For example:

Not encoded:

```
?oslc.where=dcterms:title="test case 1" and dcterms:modified>="2008-12-02T18:42:30"
```

Encoded:

```
?oslc.where=dcterms%3Atitle%3D%22test%20case%201%22%20and%20dc%3Amodified%3E%3D%222008-12-02T18%3A42%3A30%22
```

References

These are the specifications referenced by the OSLC Core Query Syntax.

- BNF Backus-Naur Form
- Dublin Core - Dublin Core Metadata Element Set, Version 1.1

- FOAF - Friend of a Friend (FOAF)
- HTTP - Hyper-text Transfer Protocol (HTTP/1.1)
- RDF/XML Concepts - RDF/XML Concepts and Abstract Syntax
- RDF/XML Syntax - RDF / XML Syntax Specification (Revised)
- URI Syntax - URI Generic Syntax
- XML Namespaces - Namespaces in XML 1.0 (Third Edition)
- XML Base - XML Base (Second Edition)
- XSD Datatypes - XML Schema Part 2: Datatypes Second Edition
- SPARQL - SPARQL Query Language for RDF, Version 1.1

Topic revision: r18 - 03 Mar 2014 - 12:25:42 - SteveSpeicher