CloudBlue
https://cloudblue.com

Resource Query Language (RQL)

This article has been generated from the online version of the documentation and might be out of date. Please, make sure to always refer to the online version of the documentation for the up-to-date information.

Auto-generated at June 28, 2022

Introduction

Data querying is a critical component of the CloudBlue Connect APIs. With the advance of rich client-driven applications, new querying techniques are needed, and Resource Query Language (RQL) defines a very simple but extensible query language specifically designed to work within URIs and query for collections of resources.

Video Tutorial

Syntax

RQL consists of named operators that take a set of arguments and syntactically follows the standard parenthesis's-based call syntax used by many modern languages (Python, JavaScript, C, etc) as prefix notation operators.

Everything in RQL can be expressed as a set of such operators by nesting and sequencing them. Nothing needs to be altered to add new operators. For example, to express a query for all the assets where the 'status' is 'processing':

```
GET /assets?in(status,(processing))
```

RQL is specifically designed for the web, using URI friendly delimiters, and using URI character encoding for all text/string values. The example above could alternately be written in RQL as:

```
GET /assets?status=processing
```

This syntax makes it easy to use RQL with a variety of existing tools and processes. Please refer to the following sections our documentation for details on supported operators and syntax.

Operators

Definition of the common operators supported by the CloudBlue Connect modules.

Comparison

The following comparison operations are supported

eq (property, value)

Filters for objects where the specified property's value is **equal** to the provided value. Example:

```
GET /assets?status=processing
GET /assets?eq(status,processing)
```

```
GET /assets?status=eq=processing
```

The above example will filter all assets with status equal to processing.

ne (property, value)

Filters for objects where the specified property's value is **not equal** to the provided value. Example:

```
GET /assets?ne(status,processing)
GET /assets?status=ne=processing
```

The above example will filter all assets with status NOT equal to processing.

gt (property, value)

Filters for objects where the specified property's value is **greater** than the provided value. Example:

```
GET /assets?gt(events.created.at,2020-01-01T00:00:00+00:00)
GET /assets?events.created.at=gt=2020-01-01T00:00:00+00:00
```

The above example will filter all assets created after January 01, 2020 excluding the ones created exactly on January 01, 2020.

ge (property, value)

Filters for objects where the specified property's value is **greater or equal** than the provided value. Example:

```
GET /assets?ge(events.created.at,2020-01-01T00:00:00+00:00)
GET /assets?events.created.at=ge=2020-01-01T00:00:00+00:00
```

The above example will filter all assets created after January 01, 2020 including the ones created exactly on January 01, 2020.

lt (property, value)

Filters for objects where the specified property's value is **less** than the provided value. Example:

```
GET /assets?lt(events.created.at,2020-01-01T00:00:00+00:00)
GET /assets?events.created.at=lt=2020-01-01T00:00:00+00:00
```

The above example will filter all assets created before January 01, 2020 excluding the ones created exactly on January 01, 2020.

le (property, value)

Filters for objects where the specified property's value is **less or equal** than the provided value. Example:

```
GET /assets?le(events.created.at,2020-01-01T00:00:00+00:00)
GET /assets?events.created.at=le=2020-01-01T00:00:00+00:00
```

The above example will filter all assets created before January 01, 2020 including the ones created exactly on January 01, 2020.

Search

like (property, pattern)

Filters for objects where the specified property's value **matches the pattern (case sensitive)**. Example:

```
GET /assets?like(product.name,*best*)
```

The above example will filter all assets with the product.name property containing "best" string matching the case, i.e. all of the following values: "Best", "BEST" and "BEst" will NOT match the filter criteria.

ilike (property, pattern)

Filters for objects where the specified property's value **matches the pattern which can include optional special symbol '*'**. Example:

```
GET /assets?ilike(product.name,*best*)
```

The above example will filter all assets with the product.name property containing "best" string ignoring the case, i.e. all of the following values: "Best", "BEST" and "BEst" will match the filter criteria.

search = pattern

Search allows filtering by all properties supporting such lookups that match a given pattern. Example:

```
GET /assets?search=cloud
```

The above example will filter all assets any property containing "cloud" string ignoring the case, just like it would be with the

'*cloud*' search pattern of the ilike operator applied to all the properties supporting the 'search=true' lookup.

List

in (property, array-of-values)

Filters for objects where the specified property's value **matches any value from the specified array of options.** Example:

```
GET /assets?in(status,(processing,active))
```

The above example will filter all assets with status values from the list of possible value (processing, active).

It is possible to specify just 1 value in the array, in this case query syntax will look like in the following example:

```
GET /assets?in(status,(active))
```

The above example will filter all assets in the 'active' status only.

out (property, array-of-values)

Filters for objects where the specified property's value **does NOT match any value from the specified array of options.** Example:

```
GET /assets?out(status,(processing,active))
```

The above example will filter all assets with status values NOT from the list of possible value (processing, active).

Logical

and (query1, query2, ...)

Allows combining results of 2 or more queries with the logical AND operator. Example:

```
/products?id=PRD-0000-0001&like(name=*best*)
/products?and(id=PRD-0000-0001,like(name=*best*))
/products?id=PRD-0000-0001,like(name=*best*)
```

The above example will filter all products with either ID that equals 'PRD-0000-0001' AND name containing the string 'best' in them.

or (query1, query2, …)

Allows combining results of 2 or more queries with the logical OR operator. Example:

```
/products?(id=PRD-0000-0001|like(name=*best*))
/products?or(id=PRD-0000-0001,like(name=*best*))
/products?(id=PRD-0000-0001;like(name=*best*))
```

The above example will filter all products with either ID that equals 'PRD-0000-0001' OR name containing the string 'best' in them.

not (query)

Allows to **invert** results of the query that is passed to this operator. Example:

```
GET /assets?not(product.name=empty())
```

The above example will filter all assets linked to products with a non-empty name.

Constants

null ()

Special constant allowing to reference values that are not set. A null value does not point to any particular value. A null value is a false value. Example:

```
GET /assets?product.description=null()
GET /assets?eq(product.description,null())
```

The above example will filter all assets linked to products with a non-initialized description property. Note that null is not the same as empty value covered in the following section of this document.

empty ()

Special constant allowing to reference empty values (e.g. ''). Example:

```
GET /assets?product.description=empty()
GET /assets?eq(product.description,empty())
```

The above example will filter all assets linked to products with an empty description property. Note that null is not the same as

empty value covered in the previous section of this document.

Pagination

limit = value

Allows limiting the number of objects returned as a result of the query. If not provided, returns the default number of objects for the specific endpoint. Example:

```
GET /assets?limit=100
```

The above example will return up to 100 assets of the query results.

offset = value

Allows offsetting objects calculated as a result of the query. If not provided, defaults to 0. Offset is typically used together with the limit property. Example:

```
GET /assets?offset=500&limit=100
```

The above example will return up to 100 assets starting from the offset 500 from the beginning of the collection.

Other

ordering (+property1, -property2, …)

Allows defining the ordering of the resulting query by one or multiple properties. The '+' stands for ASC ordering and '-' for DESC order. By default, any property is assumed to have ASC ('+') order. Example:

```
GET /assets?ordering(+events.created.at,-product.name)
GET /assets?ordering(events.created.at,-product.name)
```

The above example will order (sort) results of the query first by the events.created.at property followed by the product.name.

select (+property1, -property2, …)

Allows including/excluding a set of fields with optimization of querying at the same time. Example:

```
GET /assets?select(+stats,-product)
GET /assets?select(stats,-product)
```

The above example will include stats and exclude product information form each object returned by the query.

Advanced Cases

Special characters in Values

If you need to pass special characters like **&^$? etc** or whitespaces as values to RQL operators, you need to put those into one of the supported quotation marks **“** or **‘** as illustrated by the following example:

```
GET /assets?product.name='white space & special^ symbols!'
```

In case you need to use a quotation mark symbol within the value, you can do that by using different quotation mark as an envelope as shown in the following example:

```
GET /assets?product.name='i am "happy" is quoted here'
GET /assets?product.name="i am 'happy' is quoted here"
```

Allowing you to pass these special symbols as values.

Star character in like and ilike operators

The star ***** character has a special meaning in the **like** and **ilike** operators as it matches zero or more of any characters in its place from the parameter value. For example, “*now” pattern will match the “Available now” and simply “now” strings, but will not patch the “Available now!” because of the last character.

But what if we need to use the star * character itself in the pattern? In that case, the star character can be escaped as ‘*’, i.e. like in the following example:

```
GET /assets?like(product.name,*best\**)
```

Where like operator will literally be searching for the “best*” string somewhere in the product.name parameter.