Using scope and filter expressions

Introduction

How to use create Expression objects used by many API endpoints as filter or scope definitions.

Many API endpoints accept an expression that defines a search filter or scope for an instruction.

To use such API endpoint you have to create a JSON object describing that expression and send it in the body or your request. Because these expressions can be very complex, they are not used in query strings of GET requests and only utilized in the body of a POST or PUT request.

Expressions used both for filtering and scope definition have the same structure, so this page applies to both types. Attributes of filter expressions depend on the API endpoint (and the entity bring filtered) and will be listed in the endpoint's documentation. Scope attributes are pre-defined and more information can be found here: Defining the scope.

If you're working with Tachyon version 3.2 or earlier, please read the Limitations in Tachyon versions up to 3.2 (inclusive) that were present in those versions of Tachyon and were removed in Tachyon 3.3.

Lastly, many API endpoints which allow filtering also allow sorting. Sort expressions have a different syntax and more information on that can be found on the Sort Definition page.

On this page:

Building an expression

Anatomy of an expression

Expressions can be built using two types of elements: an expression element (sometimes called a 'leaf') and an expression tree object (called a 'node'):

- An expression element (leaf) - used to define a single condition on an attribute. Elements cannot have any child elements
- An expression tree object - containing a list of expression elements with a logic operator that's applied to that list (called a 'node'). Please note that the operator is applied to all elements on the list.

You should be aware that:

- The expression tree object does not deal with attributes - instead it performs a logic operation on a collection of elements and / or other expression tree objects
- If your expression has only one element in it (for example, just one attribute to filter or scope on), you should send a single expression element without the encompassing expression tree object (see below)
- You can nest expressions to create a multi level tree, an expression tree object can have elements and other tree objects as operands.

Available expression operators

Available operators depend on whether you're dealing with an expression element or expression tree object.

Expression element (leaf)

The available operators are essentially SQL operators.

You can use following the operators:

Operator
Like
== (equality)
!= (inequality)
>
<
>=
<=

⚠ Not all operators work with all data types.

Expression tree object (node)

The only available logical operators are AND, OR and NOT.

⚠ NOT is a unary operator and can only have one operand.

Expression object

Expression element

```
{
        "Attribute": "",
        "Operator": "",
        "Value": "",
        "DataType": ""
}
```

- Attribute - the columns or element to scope or filter on
- Operator - defines the operator for the expression (from the above list)
- Value - the expression
- DataType - defines the attribute's data type.

The DataType property can be omitted in data sent to the API. It's designed for use by client code to validate the value set for given element.

Supported data types

- string
- integer
- datetime
- double
- boolean

Handling boolean values

Boolean values can only be used with the equality operator comparing against words 'false' and 'true'.

Expression tree object

Expression tree

```
{
        "Operator": "",
        "Operands": []
}
```

Operator - defines the logical operator applied to the operands, it can be AND, OR and NOT

Expression - tree object follows normal logic operator rules, for example a tree object with an 'AND' will only return true if all expressions inside it are true while an object with 'OR' will return true if at least one expressions inside it is true.

'NOT' operator is a unary operator which negates whatever its sole operand evaluates to.

Operands - is an array of Expression elements defined above.

Limitations in Tachyon versions up to 3.2 (inclusive)

- The only operator supported in the expression tree object is AND
- The system only supports single level expression trees, for example you cannot have an expression tree object and an operand for another expression tree object.

Examples

Single element expression

An expression that limits a string element called OsType to a value of Windows.

Json for a single element expression

```
{
        "Attribute": "OsType",
        "Operator": "==",
        "Value": "Windows"
}
```

Multi element expression

An expression tree consisting of three expression objects.

- The first limits a string element called OsType to a value of Windows
- The second limits an integer element called OsVer to a value of 7
- The third one limits a string element called DeviceType to a value of Desktop.

Json for a multi element expression

```
{
        "Operator": "AND",
        "Operands": [
        {
                "Attribute": "OsType",
                "Operator": "==",
                "Value": "Windows"
        },
        {
                "Attribute": "OsVer",
                "Operator": "==",
                "Value": "7"
        },
        {
                "Attribute": "DeviceType",
                "Operator": "==",
                "Value": "Desktop"
        }
        ]
}
```

Multi level expression

An expression tree with multiple levels.

This expression can be used to find computers running Windows OS and are either a desktop or laptop.

The outer level of this tree has two elements - the first limits the OsType to Windows, the other contains the inner expression, which itself limits DeviceType to either desktop or laptop.

JSON for multi level expression

```
{
    "Operator": "AND",
    "Operands": [
    {
        "Attribute": "OsType",
        "Operator": "==",
        "Value": "Windows"
    },
    {
        "Operator": "OR",
        "Operands": [{
            "Attribute": "DeviceType",
            "Operator": "==",
            "Value": "Laptop"
        },
        {
            "Attribute": "DeviceType",
            "Operator": "==",
            "Value": "Desktop"
        }]
    }]
}
```

Negating an expression

Negating an expression.

To get only devices which are not Windows desktops you can write an expression selecting Windows desktops and then negate it.

```
{
    "Operator": "NOT",
    "Operands": [{
        "Operator": "AND",
        "Operands": [{
            "Attribute": "OsType",
            "Operator": "==",
            "Value": "Windows"
        },
        {
            "Attribute": "DeviceType",
            "Operator": "==",
            "Value": "Desktop"
        }]
    }]
}
```