Artifactory Query Language

Overview

Artifactory Query Language (AQL) is specially designed to find artifacts stored within Artifactory based on any number of search criteria. Its syntax offers a simple way to formulate complex queries that specify any number of search criteria, filters, sorting options, and field output parameters. AQL is exposed as a RESTful API which, when possible, uses data streaming to provide output data resulting in extremely fast response times and low memory consumption. Currently, AQL can only extract data that resides in your instance of Artifactory, so it runs on **local repositories** and **remote repository caches**.

Here are a couple of examples that show the power of AQL.

Example 1:

Return all artifacts of the "artifactory" build whose license is not an Apache license.

```
items.find
(
        {
            "$and":
                [
                {"@build.name":{"$eq":"artifactory"}},
            {"@artifactory.licenses":{"$nmatch":"Apache-*"}}
        ]
        }
)
```

Example 2:

Return all artifacts of the "artifactory" build that has been downloaded more than 100 times during the month of January 2014.

Display the artifact name, its repository, its path in the repository and the number of times it was downloaded

```
items.find
(
        {
            "$and":
                [
                {"@build.name":{"$eq":"artifactory"}},
            {"stat.downloads":{"$gt":100}},
            {"stat.downloaded":{"$gte":"2014-01-01"}},
            {"stat.downloaded":{"$lt":"2014-02-01"}}
        ]
    }
).include("name","repo","path","stat.downloads")
```

Page Contents

Usage

Syntax

```
items.find(<criteria>).include(<fields>).sort(<order_and_fields>).limit(<num_records>)
```

where:

criteria	The find criteria in valid JSON format
fields	(Optional) There is a default set of fields for query output. This parameter lets you specify a different set of fields that should be included in the output
order_and_fields	(Optional) The fields on which the output should be sorted, and the sort order. A default set of fields and sort order is defined for each entity type.
num_records	(Optional) The maximum number of records that should be extracted. If omitted, all records answering the query criteria will be extracted.

Execution

To execute an AQL query, use the Artifactory Query Language REST API.

Entities and Fields

AQL operates in the context of three entity types.

You may issue a **find** request on an **item** entity type according to the syntax below, and configure your request to display **item** fields.

Entity Type	Field Name	Type	Description
item	repo	String	The name of the repository in which this item is stored
	path	String	The full path associated with this item
	name	String	The name of the item
	created	Date	When the item was created

	modified	Date	File system timestamp indicating when the item was last modified
	updated	Date	When the item was last uploaded to a repository.
	created_by	String	The name of the item owner
	modified_by	String	The name of the last user that modified the item
	type	Enum	The item type (file/folder/any). If `type` is not specified in the query, the default type searched for is **`file`**
	depth	int	The depth of the item in the path from the root folder
	original_md5	String	The item's md5 hash code when it was originally uploaded
	actual_md5	String	The item's current md5 hash code
	original_sha1	String	The item's sha1 hash code when it was originally uploaded
	actual_sha1	String	The item's current sha1 hash code
	size	long	The item's size on disk

In addition to **item**, there are also the **property** and **stat** entity types.

While you may not issue a **find** request directly on these entity types, you may display the fields of these entities that are associated with items that were returned by your **find** request. (i.e. you may display any **property** or **stat** of an item that is returned by your **find** request), or use them in your specification of search criteria.

Entity Type	Field Name	Type	Description
property	key	String	The property key
	value	String	The property value
stat	downloaded	date	The last time an item was downloaded
	downloads	int	The total number of downloads for an item
	downloaded_by	String	The name of the last user to download this item

Constructing Search Criteria

The **criteria** element must be a valid JSON format statement composed of the criteria that specify the items that should be returned. It is essentially a compound boolean statement, and only elements for which the statement evaluates to **true** are returned by the query.

Each criterion is essentially a comparison statement that is applied either to a field or a property. Please see the full list of Comparison Operators. While each criterion may be expressed in complete general format, AQL defines shortened forms for readability as described below.

Field Criteria

The general way to specify a criterion on a field is as follows:

```
{"<field>" : {"<comparison operator>" : "<value>"}}
```

If the query applied is to a different entity type, then it must be pre-pended by the entity type.

For example:

```
//Find items whose "name" field matches the expression "*test.*"
items.find({"name": {"$match" : "*test.*"}})

//Find items that have been downloaded over 5 times.
//We need to include the "stat" specifier in "stat.downloads" since downloads is a query of the stat entity and
not of the item entity.
items.find({"stat.downloads":{"$gt":"5"}})
```

Short notation for Field criteria

AQL supports a short notation for search criteria on fields.

An "equals" ("$eq") criterion on a field may be specified as follows:

`{"<field>" : "<value>"}`

Example	Find items whose "name" field equals "ant-1.9.4.jar"
Regular notation	`items.find({"name":{"$eq":"ant-1.9.4.jar"}})`
Short notation	`items.find({"name":"ant-1.9.4.jar"})`

Properties Criteria

The general way to specify a criterion on a property is as follows:

```
{"@<property_key>":{"operator":"<property_value>"}}
```

For example:

```
 //Find item with property named "os" and value that matches the expression "linux*"
items.find({"@os" : {"$match" : "linux*"}})
```

Short notation for properties criteria

AQL supports a short notation for search criteria on properties.

An "equals" ("$eq") criterion on a property may be specified as follows:

{"@<property_key>" : "<property_value>"}

Example	Find items with associated properties named "license" with a value that equals "GPL"
Regular notation	`items.find({"@artifactory.licenses" : {"$eq" : "GPL"}})`
Short notation	`items.find({"@artifactory.licenses" : "GPL"})`

Compounding Criteria

Search criteria on both fields and properties may be nested and compounded into logical expressions using "**$and**" or "**$or**" operators. If no operator is specified, the default is **$and**

```
<criterion>={<"$and"|"$or">:[{<criterion>},{<criterion>}]
```

Criteria may be nested to any degree

Note that since search criteria can be nested to any degree, you may construct a logical search criteria with any degree of complexity required.

Here are some examples:

```
//This example shows both an implicit "$and" operator (since this is the default, you don't have expressly
specify it, but rather separate the criteria by a comma), and an explicit "$or" operator.
//Find all items that are files and are in either the jcenter-cache or my-local repositories.
items.find({"type" : "file","$or":[{"repo" : "jcenter-cache", "repo" : "my-local" }]})

//Find all the items that are either in a repository called "debian" and whose name ends with ".deb" or are in
a repository called "yum" and whose name ends with ".rpm".
items.find(
        {
                "$or":
                [
                        {
                                "$and":
                                [
                                        {"repo" : "debian"} ,
                                        {"name" : {"$match" : "*.deb"}}
                                ]
                        },
                        {
                                "$and":
                                [
                                        {"repo" : "yum"} ,
                                        {"name" : {"$match" : "*.rpm"}}
                                ]
                        }
                ]
        }
)

//Find all items in a repository called "my_local", and that have a property with a key called "license" and
value that is any variant of "LGPL".
items.find({"repo" : "my_local"},{"@artifactory.licenses" : {"$match" : "*LGPL*"}})
```

Matching Criteria on a Single Property ($msp)

A search that specifies several criteria on properties may sometimes yield unexpected results.

This is because items are frequently annotated with several properties, and as long as any criterion is true for any property, the item will be returned in a regular **find**.

But sometimes, we need to find items in which a single specific property answers several criteria. For this purpose we use the **$msp (match on single property)** operator.

The fundamental difference between a regular **find** and using the **$msp** operator is:

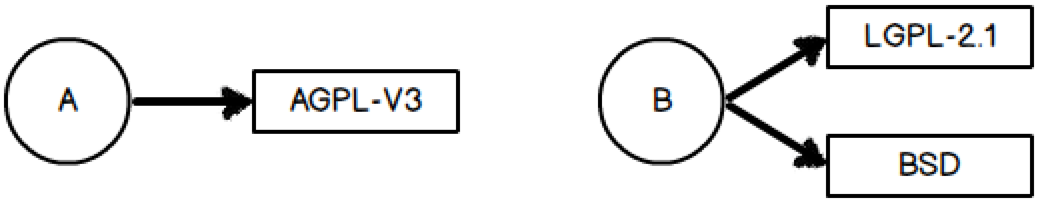
- **find** will return an item if **ANY** of its properties answer **ALL** of the criteria in the search term.
- **$msp** will only return an item if **ONE** (and only one) of its properties answers **ALL** of the criteria in the **$msp** term.

Here is an example.

Consider two items A and B.

A has a license property with value **AGPL-V3**

B has two license properties . One is **LGPL-2.1**, and the other **BSD**



Now let's assume we want to find items that use any variety of GPL license as long as it's NOT LGPL.

In our example we would expect to only get **Item A** returned since item B does use LGPL-2.1.

As a first thought, we might write our query as follows:

```
items.find({"@artifactory.licenses":{"$match":"*GPL*"}},{"@artifactory.licenses":{"$nmatch":"*LGPL*"}})
```

But this query returns **both items A and B** even though B does have an LGPL license.

Item A is returned because it clearly answers both criteria.

However, B is also returned. The LGPL-2.1 license answers the first criterion. The BSD license answers the second criterion (it's not an LGPL license).

If we use the **$msp** operator as follows:

```
items.find({"$msp":[{"@artifactory.licenses":{"$match":"*GPL*"}},{"@artifactory.licenses":{"$nmatch":"
*LGPL*"}}]})
```

Then only Item A is returned. As soon as the find detects the LGPL license in Item B, it is rejected.

Comparison Operators

The following table lists the full set of comparison operators allowed:

Operator	Types	Meaning
$ne	string, date, int, long	Not equal to
$eq	string, date, int, long	Equals
$gt	string, date, int, long	Greater than
$gte	string, date, int, long	Greater than or equal to
$lt	string, date, int, long	Less than
$lte	string, date, int, long	Less than or equal to
$match	string	Matches
$nmatch	string	Does not match

Using Wildcards

To enable search using non-specific criteria, AQL supports wildcards in common search functions.

Using Wildcards with $match and $nmatch

When using the "**`$match`**" and "**`$nmatch`**" operators, the "*" wildcard replaces any string and the "?" wildcard replaces a single character.

"Catch all" Notation on Properties

In addition to supporting "`$match`" and "`$nmatch`", AQL supports a notation that uses wildcards to match **any** key or **any** value on properties.

If you specify "@*" as the property key, then it means a match on any key.

If you specify "*" as the property value, then it means a match on any value

Example	Find items that have any property with a value of "GPL"
Regular notation	`items.find({"$and" : [{"property.key" : {"$eq" : "*"}}, {"property.value" : {"$eq" : "GPL"}}]})`
Short notation	`items.find({"@*":"GPL"})`

Example	Find any items annotated with any property whose key is "license" (i.e. find any items with a "license" property)
Regular notation	items.find({"$and" : [{"property.key" : {"$eq" : "license"}}, {"property.value" : {"$eq" : "*"}}]})
Short notation	`items.find({"@artifactory.licenses":"*"})`

Be careful not to misuse widlcards

Wildcard characters ("*" and "?") used in queries that do not conform to the above rules are interpreted as literals.

Examples

To avoid confusion, here are some examples that use the "*" and "?" characters explaining why they are interpreted as wildcards or literals.

Query	Wildcard or Literal	Explanation	What the query returns
`items.find({"name":{"$match":"ant-1.9.4.*"}})`	Wildcard	Wildcards on fields are allowed with the `$match` operator.	All items whose name matches the expression "ant-1.9.4.*"
`items.find({"name":{"$eq":"ant-1.9.4.*"}})`	Literal	Wildcards on fields are only allowed with the `$match` and `$nmatch` operators.	Only find items whose name is literally "ant-1.9.4.*"
`items.find({"@artifactory.licenses":"*"})`	Wildcard	For properties, this short notation is allowed and denotes any value	All items with a property whose key is "license"
`items.find({"@artifactory.licenses":"*GPL"})`	Literal	This is the short notation replacing the $eq operator for properties, but it does not use the "catch all" notation for properties.	All items with a license whose value is literally "*GPL"
`items.find({"@artifactory.licenses":{"$match":"*GPL*"}})`	Wildcard	Wildcards on properties are allowed with the `$match` operator.	All items with a license matches the expression "*GPL*"

Date and Time Format

AQL supports Date and Time formats according to a W3C profile of the ISO 8601 Standard for Date and Time Formats.

The complete date and time notation is specified as:

`YYYY-MM-DDThh:mm:ss.sTZD (eg 2012-07-16T19:20:30.45+01:00)`

Date/Time specified in partial precision is also supported: (i.e. specify just the year, or year and month, year, month and day etc.)

For example, the following query will return all items that were modified after July 16, 2012 at 30.45 seconds after 7:20pm at GMT+1 time zone:

```
 //Find all the items that have been modified after 2012-07-16T19:20:30.45+01:00
items.find({"modified" : {"$gt" : "2012-07-16T19:20:30.45+01:00"}})

//Find all the items that have been modified after 2012-07-01
items.find({"modified" : {"$gt" : "2012-07-01"}})
```

For full details, please refer to the W3C documentation.

Specifying Output Fields

Each query displays a default set of fields in the result set, however you have complete control this and may specify which fields to display using an optional **include** directive in your query.

You can even specify to display fields from other entities related to your result set.

Displaying All Fields

Use: **.include("*")**

For example:

```
//Find all items, and display all the item fields
items.find().include("*")
```

Displaying Specific Fields

Each query displays a default set of fields in the output. Using the *.include* directive you can override this default setting and specify any particular set of fields you want to receive in the output.

Use: **.include("<field1>", "<field2>"...)**

For example:

```
//Find all items, only display the "name" and "repo" fields
items.find().include("name", "repo")
```

You can also display specific fields from other entities associated with those returned by the query.

If you specify any field from the **item** domain, then this will override the default output setting, and only the **item** fields you expressly specified will be displayed.

If you only specify fields from the **property** or **stat** domains, then the output will display the default fields from the **item** domain, and in addition, the other fields you expressly specified from the **property** or **stat** domains.

For example:

```
//Find all items, and display the "name" and "repo" fields as well as the number of "downloads" from the
corresponding "stat" entity
items.find().include("name", "repo", "stat.downloads")

//Find all items, and display the default item fields fields as well as the stat fields
items.find().include("stat")

//Find all items, and display the default item fields as well as the stat and the property fields
items.find().include("stat", "property")

//Find all items, and display the "name" and "repo" fields as well as the stat fields
items.find().include("name", "repo", "stat")
```

Users without admin privileges.

Users without admin privileges, need to include at least the following three fields in the `include` directive: `name, repo` and `path.`

Filtering Properties by Key

As described above, the primary use of the .include directive is to specify output fields to display in the result set.

This notion is applied in a similar way in regard to properties. Each item may be annotated with several (even many) properties. In many cases you may only be interested in a specific subset of the properties, and only want to display those.

So the .**include** directive can be used to filter out unwanted properties from the result, and only display (i.e. "include") those you are interested in.

For example, to display all the properties annotating an item found :

```
//Find all items, and display the "name" and "repo" fields, as well as all properties associated with each item
items.find().include("name", "repo", "property.*")
```

However, if you are only interested in a specific property (e.g. you just want to know the version of each item returned), you can filter out all other properties and only include the property with the key you are interested in:

```
//Find all items, and display the "name" and "repo" fields, as well as the key and value of the "version"
property of each item
items.find().include("name", "repo", "@version")
```

Sorting

AQL implements a default sort order, however, you can override the default and specify any other sort order using fields in your output by adding the `.sort` directive to the end of your query as follows:

.sort({"<$asc | $desc>" : ["<field1>", "<field2>",...]})

You can only specify sorting on fields that are displayed in the output (whether they are those displayed by default or due to a **`.include`** directive.

Here are some examples:

```
 // Find all the jars in artifactory and sort them by repo and name
items.find({"name" : {"$match":"*.jar"}).sort({"$asc" : ["repo","name"]})

 // Find all the jars in artifactory and their properties, then sort them by repo and name
items.find({"name" : {"$match":"*.jar"})include("@").sort({"$asc" : ["repo","name"]})
```