aQL

artifactory Query Language

# Reference Cards

JFrog

# Artifactory Query Language (AQL)

This guide presents a brief reference to Artifactory Query Language.

For full details, please visit wiki.jfrog.com/aql

## Page Contents

## Overview

Artifactory Query Language (AQL) is specially designed to find artifacts stored within Artifactory's repositories 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.

Here's a quick example that shows the power of AQL.

Example:

Find all artifacts of the **"artifactory"** build which use an Apache license.

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

## Syntax and Usage

```
items.find(<criteria>)                    //the basic find criteria in JSON format
  .include(<fields>)                      //(optional) the fields to include in the output
     .sort(<order and fields>)            //(optional) sort fields and order
         .limit(<num_records>)            //(optional) max. output records
```

To execute a query, use the following Artifactory REST API call

**POST /api/search/aql**

Here's an example using cURL:

```
curl -uadmin:password -X POST                                   //apply the call
  http://<host>:<port>/artifactory/api/search/aql -d            //The root URL
     items.find({"repo":{"$eq":"jcenter-cache"}})               //the actual query
```

# 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 above.

| 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 | When the item was last modified |
| | updated | Date | When the item was last updated |
| | 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) |
| | 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 their fields and use them in 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

Search criteria must be specified in JSON format and can be applied to both fields and properties.

## Field Criteria

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

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

Examples:

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 field of the stat entity and not of the item entity.

```
items.find({"stat.downloads":{"$gt":"5"}})
```

### Tip:

You can use a short notation to specify an "equals" criterion on a field:

```
{"field" : "value"}
```

Example:

//Find items whose **"name"** field equals **"utest.class"**

| | |
|---|---|
| Regular notation | `items.find({"name":{"$eq":"utest.class"}})` |
| Short notation | `items.find({"name":"utest.class"})` |

## Properties Criteria

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

Example:

Find items with a property named "os" whose value matches the expression **"linux*"**.

```
items.find({"@os" : {"$match" : "linux*"}})
```

### Tip:

You can use a short notation to specify an **"equals"** criterion on a property:

```
{"@<property_name>" : "<property_value>"}
```

Example:

Find items with associated properties named **"artifactory.licenses"** 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 can be compounded into logical expressions using **"$and"** or **"$or"** operators. The default is **"$and"**.

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

Examples:

Find all items that are files and are in either the **"jcenter"** or **"my-local"** repositories.

```
items.find({"type" : "file","$or":[{"repo" : "jcenter", "repo" : "my-local" }]})
```

Find all items in a repository called **"my_local"** that have a property with a key called **"artifactory.licenses"** and a value that is any variant of **"LGPL"**.

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

## 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

When using **`$match`** and **`$nmatch`**, **`"*"`** replaces any string and **`"?"`** replaces any character.

Examples:

Find items whose **`"path"`** field matches the expression **`"*org/jfrog*"`**:

```
items.find({"path": {"$match" : "*org/jfrog*"}})
```

Find all items whose **`"os"`** property includes **`"lin"`**:

```
items.find({"@os" : {"$match" : "*lin*"}})
```

To search for any property with a specific value, you can specify **`"@*"`** as the key.

Example:

Find items that have any property with a value of **`"GPL"`**:

```
items.find({"@*":"GPL"})
```

To search for any property with a specific key, you can specify **`"*"`** as the value.

Example:

Find items that have an **`"artifactory.licenses"`** property with any value:

```
items.find({"@artifactory.licenses":"*"})
```

## Specifying Output Fields

Each query displays a default set of fields in the result set, however you can override this to display any set of fields.

### Displaying All Fields

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

Example:

To display all fields:

```
items.find().include("*")
```

### Displaying Specific Fields

If you specify any field from the **`item`** domain, then this will override the default output setting.
Use: **`.include("<field1>","<field2>",...)`**

Example:

Display only the **`"name"`** and **`"repo"`** fields of all items:

```
items.find().include("name","repo")
```

### Displaying Fields from Other Entities

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

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.

Example:

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")
```

Display the default **`item`** fields, as well as the **`stat`** fields.

```
items.find().include("stat")
```

Display the default **`item`** fields, as well as the **`stat`** and the **`property`** fields.

```
items.find().include("stat", "property")
```

Display the **`"name"`** and **`"repo"`** fields, as well as the **`stat`** fields.

```
items.find().include("name", "repo", "stat")
```

### Filtering Properties by Key

You can also use the **`.include`** qualifier to display specific properties and filter out all the rest.

Examples:

Display the "**name**" and "**repo**" fields along with all properties.

```
items.find().include("name", "repo", "property.*")
```

Display the "**name**" and "**repo**" fields and the "**version**" property of each item.

```
items.find().include("name", "repo", "@version")
```

## Sorting

AQL implements a default sort order, however, you can specify a different sort order using the **`.sort`** qualifier at the end of your query:
Use: **`.sort({"<$asc | $desc>" : ["<field1>", "<field2>",... ]})`**

Example:

Find all jars and sort them by **repo** and **name**.

```
items.find({"name" : {"$match":"*.jar"}}).sort({"$asc" : ["repo","name"]})
```

## Coming soon...

And that's just the beginning. As we continue to release more AQL domains, it will take less and less to do more and more. Keep your eyes open for new capabilities in upcoming releases like...

Looking for archives that have "readme" files sequestered in them? No problem:

```
items.find({"archive.entry_name" : {"$match":"README.*"}})
```

Which of my builds has a dependency that uses an "EUPL" license?

```
builds.find({"module.dependency.item.@artifactory.licenses":"EUPL"})
```

What are all the dependencies of Artifactory build number 31232?

```
dependencies.find({"module.build.name":"artifactory","module.build.number":"31232"})
```

What is the list of modules in Artifactory?

```
modules.find({"build.name": "artifactory"})
```

JFrog

jfrog.com

Published May 2015. For more information, visit jfrog.com/aql.