

# Table of Contents

1. Query Language (BIAPIQuery) .....	1
1.1. Summary of Operators .....	1
1.2. Value Types .....	1
1.3. Array Query Support (elemMatch) .....	1
1.4. IN Clause Enhancements .....	2
1.5. Variables and Resolvers .....	2
1.6. Reference Equality .....	3
1.7. Examples .....	3

# 1. Query Language (BIAPIQuery)

This section documents the ANTLR-based filter language used by repositories (e.g., MorphiaRepo) and the rule engine to construct MongoDB/Morphia filters.

## 1.1. Summary of Operators

- Equals: ':' (field:value)
- Not equals: '!='
- Less than / Greater than: '<' / '>'
- Less-than-or-equal / Greater-than-or-equal: '<=' / '>='
- Exists (field present): ':~' (no value)
- In list: '^' followed by a list [v1,v2,...]
- Boolean literals: true / false
- Null literal: null
- Logical AND: '&&'
- Logical OR: '||'
- Logical NOT (on a single expression): '!!'
- Grouping: parentheses '(' and ')'

## 1.2. Value Types

- Strings: unquoted or quoted with "\"" (quotes allow spaces and punctuation)
- Whole numbers: prefix with '#' (e.g., #10)
- Decimals: prefix with '.' (e.g., 19.99)
- Dates: yyyy-MM-dd
- DateTime: ISO-8601 (e.g., 2024-07-28T10:15:30Z)
- ObjectId: 24-char hex
- References: @@<ObjectId> resolves to a Morphia @Reference field instance
- Variables: \${...} substituted from PrincipalContext/ResourceContext and resolver-provided variables (see Variables and Resolvers below)

## 1.3. Array Query Support (elemMatch)

The grammar supports array filtering via elemMatch using ':=' with braces. Example:

```
lineItems:= { (sku:"ABC" && quantity:> #0) || (sku:"DEF" && backordered:true) }
```

This produces a Mongo \$elemMatch on field 'lineItems' with the nested expression composed inside. You can nest multiple predicates with '&&' / '||' and parentheses as needed. Internally, the

listener composes the inner filters and wraps them in `Filters.elemMatch(...)`.

Notes: - `elemMatch` is ideal for filtering documents that contain an array of subdocuments where at least one element satisfies the nested criteria. - The inner expression supports the full set of operators supported for basic expressions, including numbers, booleans, strings, dates, etc.

## 1.4. IN Clause Enhancements

The IN operator (`:'^')` supports both literal lists and variable-driven lists.

1) Literal values:

```
customerId:^(5f1e1a5e5e5e5e5e5e5e5e5e51,5f1e1a5e5e5e5e5e5e5e5e5e52]
```

- `ObjectId` values are detected from 24-hex strings and converted to `ObjectId`.
- Numbers, booleans, ISO datetimes, and yyyy-MM-dd dates are also coerced appropriately.

2) Variable-driven lists via `${var}` inside brackets:

```
customerId:^( ${accessibleCustomerIds} ]
```

- If `'accessibleCustomerIds'` is published as a `Collection/array` via an `AccessListResolver` (see below), each element is type-coerced.
- If it is published as a single string (e.g., CSV), it is split by comma and each value is type-coerced.
- Empty or blank variables yield an empty list (matching none).

Type coercion rules used for each element: - 24-hex → `ObjectId` - `'true'/'false'` → `boolean` - integer → long - decimal → double - ISO datetime → `java.util.Date` - yyyy-MM-dd → `java.time.LocalDate` - otherwise → string

## 1.5. Variables and Resolvers

Variables in expressions are written as `${name}`. The framework populates variables from: - `PrincipalContext` and `ResourceContext` (standard variables): `principalId`, `pAccountId`, `pTenantId`, `ownerId`, `orgRefName`, `resourceId`, `action`, `functionalDomain`, `area` - `AccessListResolver` implementations (custom variables): collections or values computed per request

Examples:

- Constrain by tenant and org for the current principal:

```
functionalDomain:"order" && dataDomain.tenantId:${pTenantId} &&  
dataDomain.orgRefName:${orgRefName}
```

- Restrict to an access-list resolved for the caller:

```
customerId:^([${accessibleCustomerIds}])
```

Authoring tips: - For single values, you can use variables anywhere a literal value can appear: field:\${ownerId} - For lists, prefer the bracket form with a single \${var} inside the brackets for best results.

## 1.6. Reference Equality

When comparing a @Reference field to an id, prefer the REFERENCE token using '@@' prefix:

```
ownerRef:@@5f1e1a5e5e5e5e5e5e5e5e5e51
```

The listener reflects on the modelClass to instantiate the referenced type and set its id accordingly.

## 1.7. Examples

- Simple string and number comparisons:

```
status:"OPEN" && total:>= ##100.00
```

- Exists and not equals:

```
externalId:~ && status!="CANCELLED"
```

- Grouping and OR:

```
(type:"INVOICE" && total:> ##0.0) || type:"QUOTE"
```

- Array match:

```
lines:= { sku:"ABC" && quantity:> #0 }
```

- IN with resolver list:

```
customerId:^([${accessibleCustomerIds}])
```