

oneM2M logo

| oneM2M Technical Report | oneM2M Technical Report |
| --- | --- |
| Document Number | TR-0070-V-5.0.0 |
| Document Name: | Enhanced filtering and Queries |
| Date: | 2024-09-02 |
| Abstract: | This TR provides an overview about the current status of querying and filtering in oneM2M, formulates new use cases, and provides an overview about relevant query languages. |
| Template Version: January 2020 (do not modify) |  |

The present document is provided for future development work within oneM2M only. The Partners accept no liability for any use of this report.

The present document has not been subject to any approval process by the oneM2M Partners Type 1. Published oneM2M specifications and reports for implementation should be obtained via the oneM2M Partners’ Publications Offices.

About oneM2M

The purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

More information about oneM2M may be found at: http//www.oneM2M.org

Copyright Notification

1. 2020, oneM2M Partners Type 1 (ARIB, ATIS, CCSA, ETSI, TIA, TSDSI, TTA, TTC).

All rights reserved.

The copyright and the foregoing restriction extend to reproduction in all media.

Notice of Disclaimer & Limitation of Liability

The information provided in this document is directed solely to professionals who have the appropriate degree of experience to understand and interpret its contents in accordance with generally accepted engineering or other professional standards and applicable regulations. No recommendation as to products or vendors is made or should be implied.

NO REPRESENTATION OR WARRANTY IS MADE THAT THE INFORMATION IS TECHNICALLY ACCURATE OR SUFFICIENT OR CONFORMS TO ANY STATUTE, GOVERNMENTAL RULE OR REGULATION, AND FURTHER, NO REPRESENTATION OR WARRANTY IS MADE OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR AGAINST INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. NO oneM2M PARTNER TYPE 1 SHALL BE LIABLE, BEYOND THE AMOUNT OF ANY SUM RECEIVED IN PAYMENT BY THAT PARTNER FOR THIS DOCUMENT, WITH RESPECT TO ANY CLAIM, AND IN NO EVENT SHALL oneM2M BE LIABLE FOR LOST PROFITS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES. oneM2M EXPRESSLY ADVISES ANY AND ALL USE OF OR RELIANCE UPON THIS INFORMATION PROVIDED IN THIS DOCUMENT IS AT THE RISK OF THE USER.

# Contents

[1 Scope](#scope)  
[2 References](#references)  
    [2.1 Normative references](#normative-references)  
    [2.2 Informative references](#informative-references)  
[3 Definition of terms, symbols and abbreviations](#Xe254da9c2f303321d6c9a4dcbd53386c9eebb59)  
    [3.1 Terms](#terms)  
    [3.2 Symbols](#symbols)  
    [3.3 Abbreviations](#abbreviations)  
[4 Conventions](#conventions)  
[5 Introduction](#introduction)  
    [5.1 Current Method of Filtering in oneM2M](#current-method-of-filtering-in-onem2m)  
    [5.2 Semantic Discovery and Query](#semantic-discovery-and-query)  
    [5.3 Shortcomings of the Current Filtering Method](#Xf80c556cae77f7f48c39509cff0179584a2b145)  
    [5.4 Evaluation Criteria of <action>, <dependency>, and <state> Resource Types](#X5baf74f567a999370f07d6e979ff045fcdb9470)  
[6 Requirements for IoT & oneM2M Resource Querying and Filtering](#Xc4f05f3f34d824919fcf82dffa94dbb21647360)  
    [6.1 Existing Requirements in TS-0002](#existing-requirements-in-ts-0002)  
    [6.2 New Use Cases](#new-use-cases)  
        [6.2.1 Searching for Specific Protocol Schemes in *pointOfAccess* Attribute](#X3d97681bba941338daeff8e8a15d4b2fb13343a)  
        [6.2.2 Monitor Temperature Values](#monitor-temperature-values)  
            [6.2.2.1 Compare Using <flexContainer> Resources](#compare-using-flexcontainer-resources)  
            [6.2.2.2 Compare Using <container> and <contentInstance> Resources](#Xa3325b4bbedec0611217e3789dcfc8942f9fa45)  
        [6.2.3 Search for a Tag in *label* OR Where Different Attributes are Larger Than a Specific Values](#X0635cfe1a7e6e1c90973259f3e30c459dc2b658)  
        [6.2.4 Search for Subscriptions that Monitor a Certain Child-Resource Type](#X79794c6b9f9c4ea69ccaad5ebdd632941695b00)  
        [6.2.5 Reacting on Multiple Attribute Changes for <action> Resources](#X73c01637c2cda98892d91147b376053db70267e)  
    [6.3 Requirements](#requirements)  
        [6.3.1 Functional Requirements](#functional-requirements)  
        [6.3.2 Non-Functional Requirements](#non-functional-requirements)  
[7 Query Languages](#query-languages)  
    [7.1 Query Languages Comparison](#query-languages-comparison)  
        [7.2 Query Languages Examples](#query-languages-examples)  
            [7.2.1 SQL](#sql)  
            [7.2.2 XQuery](#xquery)  
            [7.2.3 JSON Query](#json-query)  
    [7.3 Alternatives to Improve oneM2M’s Filters and Queries](#X66c7a4a33f71e7296b36a59efacf69e8de00eeb)  
    [7.3.1 Enhancing the Existing *Filter Criteria* Method](#Xcbefe0ad4676246b56eb3d059cfdb3c400d5700)  
            [7.3.1.1 Filter Groups](#filter-groups)  
            [7.3.1.5 Advantages and Disadvantages](#advantages-and-disadvantages)  
    [7.3.2 Defining a Simplified New Filter and Query Language](#Xd5181b94d2ac84047ca09bab7d0f55b0d6e6817)  
            [7.3.2.1 S-expressions](#s-expressions)  
            [7.3.2.2 Supported Comparisons](#supported-comparisons)  
            [7.3.2.3 Attribute Paths](#attribute-paths)  
            [7.3.2.4 Functions and Iterators](#functions-and-iterators)  
            [7.3.2.5 Advantages and Disadvantages](#advantages-and-disadvantages-1)  
[8 Conclusion](#conclusion)  
[Annexes](#annexes)  
[Annex <A>:Title of annex]  
    [ First clause of the annex]  
        [A.1.1 First subdivided clause of the annex](#Xfd1c62da1e91329f6b85c1020e52ed8bbc9513d)  
[Annex <y>:Bibliography]  
[History](#history)

# 1 Scope

This document presents possible requirements for an enhanced filtering and query mechanism as well as a discussion about possible solutions.

# 2 References

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references,only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

## 2.1 Normative references

The following referenced documents are necessary for the application of the present document.

* [1] TS-0001: “Functional Architecture”
* [2] TS-0002: “Requirements”

## 2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

* [i.1] oneM2M Drafting Rules <https://member.onem2m.org/static_Pages/others/Rules_Pages/oneM2M-Drafting-Rules-V1%202%202.doc>
* [i.2] Wikipedia entry on “S-expressions” <https://en.wikipedia.org/w/index.php?title=S-expression&oldid=1178440297>

# 3 Definition of terms, symbols and abbreviations

Delete from the above heading the word(s) which is/are not applicable.

## 3.1 Terms

For the purposes of the present document, the following terms and definitions apply:

TODO

SPARQL: SPARQL is a semantic query language to retrieve and manipulate RDF data. It is a W3C specification.

## 3.2 Symbols

N/A

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

TODO

CSE: Common Services Entity

IMEI: International Mobile Equipment Identity

JSON: JavaScript Object Notation

RDF: Resource Description Framework

SQL: Structured Query Language

XML: Extensible Markup Language

XSD: XML Schema Definition

# 4 Conventions

The key words “Shall”, “Shall not”, “May”, “Need not”, “Should”, “Should not” in this document are to be interpreted as described in the oneM2M Drafting Rules [i.1]

# 5 Introduction

This clause provides an overview about the current filter am query methods in oneM2M.

## 5.1 Current Method of Filtering in oneM2M

The current method for providing filters and queries in oneM2M requests is currently done by providing ***FILTER CRITERIA*** in a request. This is a complex attribute that contains a structure of matching conditions attributes and filter handling attributes that can be applied for certain RETRIEVE and DISCOVERY operations. The detailed description and the applicability is specified in TS-0001, clause 8.1.2 and table 8.1.2-2.

To summarise its use:

* *Matching conditions* are evaluated against resources. When the evaluation is true, the matched resources are part of the matching result.
* The *filter handling conditions* provide additional input used to determine the filtering result (e.g. the maximum number of resources to be included in the filtering result).
* The *filtering result* may be composed of zero, one or more resources and determines the target resources for Discovery-based operations.

Further details are also provided in TS-0001 [[1](#X265a31bd79b370d6ad9fbecf643e6fa5b1c09d5)] , clause 10.2.6 “Discovery-related Procedures”.

The *matching conditions* is a set of attributes that specify for some common and universal resource attributes matching (or comparison) a value and the kind of comparison that should be applied if present. For example, the value of the *createdBefore* attribute is compared against the *creationTime* attribute during matching, and it will be matched if a resource’s *createdTime* attribute is before the timestamp specified in *createdBefore*. Through the specific nature of the attributes it is possible to compare against the respective attributes in resources not only for equality, but also for smaller, larger, before, after, etc. However, these attribute-specific comparisons are only defined for a limited set of resource attributes.

Besides the pre-defined set of ***FILTER CRITERIA*** attributes it is possible to compare all these attributes attributes, and also the custom attributes in <flexContainer> specialisations, against equality. However, equality is the only comparison type for most of oneM2M’s attributes.

It is possible to add multiple queries and to logical combine them with setting the *filterOperations* ***FILTER CRITERIA*** attribute. This attribute specifies the logical operator that is used for all the given filter expressions. No sub-queries are allowed, eg. to mix different boolean operations. Negation of query results is also not possible.

In addition ***FILTER CRITERIA*** *filter handling* attributes are used to limit the number of filter result, specifying an offset in the filter result, or limit the search depth in the resource tree.

TS-0004 [[1](#X265a31bd79b370d6ad9fbecf643e6fa5b1c09d5)] , clause 7.3.3.17.16 “Conditions on labelsQuery attribute” defines a tiny query language to do add basic comparison operators to handle filtering on a resource’s *label* attribute (which is a list and may contain multiple tags, or tokens). However, this filter query is defined only for the *labels* attribute, and not for other scalar or list-type attributes.

TODO: Example

## 5.2 Semantic Discovery and Query

Both semantic discovery and query are not part of this document because they use a different method for handling queries, but they still should be shortly summarised here because they are an example of an additional query method in oneM2M that work together with the methods provided by the ***Filter Criteria*** request attribute.

Semantic queries are different from the filter method described in clause [5.1](#X2ab4dc6b940256de13b09144bb874586142aa5c). The filtering methods described in that clause are part of the request handling. A result set of resources or references is created or filtered afterwards according to the given configuration in the ***FILTER CRITERIA*** request attribute.

For semantic discovery and query two different methods are defined that use semantic service components to compute the result set according not only by just comparing queries and values, but by deriving explicit and implicit information from the semantic information that are provided together with the resources as well as from external information that are semantically referenced.

Resources may have child resources of type <semanticDescriptor>, which contain additional semantic information that may describe a resource respective the data contained in that resource further. For example, the data from a temperature sensor may be annotated this way to explain that data producing device is a thermostat used indoor in an industrial context, and that the data is in degree Celsius. The same semantic method can be used to semantically describe the device itself.

The differences between semantic discovery and semantic query operations are:

* A *semantic discovery* uses the information stored in the <semanticDescriptor> child-resources together with other information to build a result set for a semantic expression. For example, “discover and return all the temperature data from devices that are used indoor”.
* A *semantic query* is used to send and process a semantic expression by the semantic service component of the CSE. It may use all the available semantic information from the resources in the resource tree. The result, however, is not a set of resources or resource references, but a semantic result string (usually RDF triples).

The query format used in both cases is SPARQL.

## 5.3 Shortcomings of the Current Filtering Method

The current filtering method described in clause [5.1](#X2ab4dc6b940256de13b09144bb874586142aa5c) has limitations that prevents to run more sophisticated RETRIEVE and DISCOVERY requests in the CSE. This means that an AE that needs more elaborate filtering capabilities has to implement them itself, which in case of a restricted device is not always easy or even possible.

To summarise the limitations:

* Comparison operations such as “greater”, “less”, “earlier”, “later” are defined for only a limited set of common and universal attributes.
* Other resource attributes that are not part of this predefined set, including <flexContainer> custom attributes, can only be compared for equality.
* Filtering of attributes that are part of complex attributes is not possible.
* Filtering on elements of a list or sub-strings in a string is not possible.
* Filter expressions that contain multiple attributes can only be combined via “AND”, “OR”, or “XOR” logical operators.
* Filter expressions cannot be negated (logical operator “NOT”)

## 5.4 Evaluation Criteria of <action>, <dependency>, and <state> Resource Types

In addition to the queries and filters provided in requests some resources also use a similar method to perform their tasks. The <action> and <dependency> resources provide a simple method to react on changes in other resources without notifying an Application Entity to perform the necessary requests itself. In the resource type <state> the *evalCriteria* attribute is used to control state transitions.

These resource types have attributes that contain an evaluation criteria that is used for the comparison as soon as an update happens to a target resource. This is sufficient for simple scenarios, such as switching on and off a light depending on the value of an updated *content* attribute in a <container> resource, but more sophisticated scenarios, for example evaluating multiple or complex attributes, are not possible.

To summarise the limitations:

* It is only possible to evaluate a single “top-level” attribute of a subject resource.
* The comparison operations are limited to a few basic comparison operations (e.g. “in” to test for the existence in a list is not possible).

# 6 Requirements for IoT & oneM2M Resource Querying and Filtering

This section provides discussions on existing an new requirements for querying and filtering.

## 6.1 Existing Requirements in TS-0002

The following requirements regarding resource querying and filtering can be found in oneM2M TS-0002 [[2](#X43b58990f2b8fb6933d1a1555450410c613db3d)].

Table 6.1-1: List of existing requirements in TS-0002

| Requirement ID | Description | Release |
| --- | --- | --- |
| OSR-083 See REQ-2015- 0593R02 | The oneM2M System shall be able to filter information from oneM2M Devices for a given set of parameters. | Not implemented |
| ONT-018 See REQ-2018- 0057R01 | The oneM2M system shall support semantic query and discovery across heterogeneous ontologies including support of automatic ontology mapping and semantic reasoning. | 4 |
| ONT-018 See REQ-2018- 0057R01 | The oneM2M system shall support semantic query and discovery across heterogeneous ontologies including support of automatic ontology mapping and semantic reasoning. | 4 |

The first requirement implies a technical filter or query while the other requirements are related to semantic querying and discovery.

## 6.2 New Use Cases

The following is a list of new use cases that are interesting for IoT application as well as for additional functionality for common services within the CSE.

### 6.2.1 Searching for Specific Protocol Schemes in *pointOfAccess* Attribute

A management system (Application Entity) needs to find out which other Application Entities are request reachable through a specific protocol, e.g. *MQTT*. For this, the Application Entity performs a query to search for the sub-string “mqtt://” in the entries of the *pointOfAccess* attribute of all resources with the *resourceType* “Application Entity”.

**Requirement(s)**

* Filter on a sub-string in list attributes.

### 6.2.2 Monitor Temperature Values

#### 6.2.2.1 Compare Using <flexContainer> Resources

An application entity wants to retrieve only those [temperature] <flexContainer> specialisations where the *currentTemperature* attribute is above a certain value.

**Requirement(s)**

* Filter on a custom attribute that is greater than a comparison value.

#### 6.2.2.2 Compare Using <container> and <contentInstance> Resources

An application entity wants to retrieve only those <container> resources where any of the child <contentInstance> resources contains a temperature above a certain value.

**Requirement(s)**

* Filter on the values of attributes in child resources.

### 6.2.3 Search for a Tag in *label* OR Where Different Attributes are Larger Than a Specific Values

An application entity needs to retrieve resources that have specific a tag present in a *label* attribute, OR those <flexContainer> specialisations where custom attributes *a* AND *b* are BOTH larger than specific values.

**Requirement(s)**

* Filter on a single tag in a list attribute.
* Filter on multiple attributes that are greater than specific values.
* Combine comparison expressions with OR and AND.

### 6.2.4 Search for Subscriptions that Monitor a Certain Child-Resource Type

An application entity needs to retrieve all the <subscription> resources that monitor <contentInstance> resources. This involves a comparison to the *childResourceType* list attribute of the *eventNotificationCriteria* complex attribute.

**Requirement(s)**

* Filter on an attribute that is part of a complex type (child-attribute of another attribute).

### 6.2.5 Reacting on Multiple Attribute Changes for <action> Resources

An application entity wants to install an <action> resource that monitors a flexContainer> specialisation with attributes *a* and *b*. If both attributes in that resource rises above a certain value, the <action> resource shall assign a value to an attribute in another resource.

Similar requirements can be deducted for use of the <dependency> and <state> resource types.

**Requirement(s)**

* Be able to evaluate one or more attributes in one evaluation criteria.
* Be able to evaluate complex type attributes.
* Support the same comparison operators as in other queries (e.g. “in”)

TODO: Use case for parent- or child-resource attributes

## 6.3 Requirements

The following requirements are derived from the comparison above. In addition, some non-functional requirements are formulated.

### 6.3.1 Functional Requirements

* Being able to compare custom and resource specific attributes
* Being able to use comparison operators other than only equality, such as lesser and greater than
* Support other operators such as “in” for lists and strings
* Provide sub-queries
* Being able to mix boolean operations for individual or sub-queries
* Allow to reference resource’s parent and child resources and user their attributes in comparisons as well.
* Provide the same functionality for <action> and <dependency> Resources
* TBC

### 6.3.2 Non-Functional Requirements

* Ease-of-use
* Simplicity
* Size of implementation
* Be backward compatible with the existing ***Filter Criteria***, and ideally can be combined
* TBC

# 7 Query Languages

This section provides an overview about some existing query languages.

## 7.1 Query Languages Comparison

Data query languages are syntaxes designed specifically for querying and retrieving data from databases or data repositories. These query languages provide a way to interact with data, making it easier for applications to retrieve the information they need. Here is a short summary of relevant data query languages:

1. **SQL (Structured Query Language)**: SQL is the most widely used query language for relational databases. It allows users to retrieve, manipulate, and manage data stored in tables. SQL uses a declarative approach, where applications specify what data they want to retrieve rather than how to retrieve it. SQL supports various operations like SELECT, INSERT, UPDATE, and DELETE, enabling users to perform complex queries and aggregations on structured data. Relevant for filtering is the WHERE clause where filter expressions respective query statements can be formulated.
2. **XQuery**: XQuery is a query language specifically designed for querying XML (eXtensible Markup Language) data. It allows users to extract and manipulate data from XML documents using a variety of query and transformation capabilities. XQuery supports XPath, a language for navigating XML structures, making it suitable for querying and transforming complex XML data.
3. **JSON Query** (also known as JSONPath) is a syntax used to query and extract specific data from JSON documents provides a way to navigate through the hierarchical structure of JSON data and select elements based on specific criteria. It uses a path-like syntax to traverse the JSON structure and select elements or values.

### 7.2 Query Languages Examples

📝 Could also be moved to annex

The following examples show queries for the query languages introduced before.

#### 7.2.1 SQL

The following example queries data records from an addresses table filtered by a specific street name.

SELECT \*   
FROM addresses   
WHERE street\_name = 'Main Street';

The following example queries data records from an addresses table filtered by a specific street name and street number.

SELECT \*  
FROM addresses  
WHERE street\_name = 'Main Street' AND street\_number < 10;

The following example queries structured data records from an addresses table by a specific street name.

SELECT address.street  
FROM addresses  
WHERE addresses.address.street\_number < 10;

#### 7.2.2 XQuery

In general, XML and XQuery support organization of and access to complex structured data.

The following example queries elements from an XML document filtered by a specific street name.

for $address in /addresses/address  
where $address/street\_name = 'Main Street'  
return $address

The following example queries elements from an XML document filtered by a specific street name and street number.

for $address in /addresses/address  
where $address/street\_name = 'Main Street' and $address/street\_number < 10  
return $address

#### 7.2.3 JSON Query

In general, JSON and JSON Query support organization of and access to complex structured data.

The following example queries elements

$.address[?(@.streetName == "Main Street")].streetName

$.address[?(@.streetNumber < 10 && @.streetName == "Main Street")].streetName

## 7.3 Alternatives to Improve oneM2M’s Filters and Queries

The data query languages in section 7.1 provide efficient and straightforward methods for retrieving and manipulating data from databases or data sources. They offer different syntaxes and support the specific data formats and use cases and simplifying data retrieval.

However, they are tailored to the specific data formats they are designed to support. oneM2M uses XSD, the XML Schema Definition language, to formally define its specification’s artefacts (for example the resources and request structures). It would seem that *XQuery* would be a natural choice for a query language in oneM2M. However, it shares some drawbacks with the other presented query languages:

* The presented query languages are more suited for the environments for which they have been designed. oneM2M defines its own data model which is then mapped to either XML, JSON or CBOR when exchanging data with an Application Entity or another CSE during a request/response exchange.
* Though CSE implementation may choose to support any of these encoding formats, they usually only support one or two of them. Adding any of the frameworks to an implementation would probably increase the size of an implementation considerably. This would especially be the case for smaller implementations that would run as an ASN on a constraint device. Those implementations may choose only to support one content encoding (XML, JSON or cbor). Adding a query frameworks that originates from any of the the other, unsupported encodings would add even more size to a deployment.

## 7.3.1 Enhancing the Existing *Filter Criteria* Method

The main limitation of the current filtering in oneM2M is the lack of recursivity, which forbids to express complex requests, and the fact that only the OR logical operator can be used between matching conditions of the same type (i.e. attributes).

#### 7.3.1.1 Filter Groups

This first proposal therefore focuses on adding a new *filterGroup (fg)* matching condition that allows nesting expressions and enforces operations pointed to by the *filterOperation (fo)* tag also between Filter Matching Conditions of the same type. Making the operation type defined in *filterOperation* tag valid for all the Filter Matching Conditions both makes the syntax more clear/readable and allows using all AND/OR/XOR/NOT operations also for attributes.

With this approach it is not only possible to apply different operations between attribute tags in general but also to use different operations for each subsequent *filterGroup* when needed.

Example:

<filterCriteria>  
 <fo>2</fo>  
 <attribute> … </attribute>  
 <fg>  
 <fo>1</fo>  
 <labels> … </labels>  
 <childLabels > … </childLabels>  
 </fg>  
 <fg>  
 <fo>1</fo>  
 <attribute> … </attribute>  
 <fg>  
 <fo>2</fo>  
 <labels> … </labels>  
 <childLabels> … </childLabels>  
 </fg>  
 </fg>  
</filterCriteria>

This example displays new possibilities to express complex requests: - Multiple instances of *filterGroup* matching condition allowed. - *filterGroup* matching conditions can be nested. - Separate filtering operation defined for each *filterGroup* matching condition.

**Note** It is also proposed to remove a quite confusing rule in TS-0004 7.3.3.17.0, saying that: “If multiple matching conditions of the same type (i.e. same condition tag) are present in the Filter Criteria parameter, these shall be combined by applying logical OR operation”.

With the proposed extension,

* to send a disjunction request with multiple matching conditions of the same type, the *filterOperation* must be explicitely set to OR. Though this brings a backward incompatibility, this is limited to matching conditions that originally had 0..n multiplicity, that is: *contentType*, attributes, childAttributes, parentAttributes, *semanticsFilter*.
* it is allowed to use conjonctions (and XOR operations) requests with multiple matching conditions of the same type, for instance on attributes (e.g. *attr1=val1 and attr2=val2*), which was not possible before.

#### 7.3.1.5 Advantages and Disadvantages

* Major increase in semantic expressiveness with minimal syntactic change: the syntax is the same as the existing one, with only one new filter criterium; basically, it just allows nesting expressions of the existing request language.
* Easy to implement, for the same reason.
* A logical *NOT* operation is added inside the *filterGroup*.

* It does not allow non-equality operations on attributes (it could be feasible, but complex to implement and costly at runtime).
* It does not extend path-based requests à la XPath/JSONPath (except current child/parent equality on labels and attributes).
* Possible break of backward compatibility if the rule on the default value of *fo* is removed (see the note above).

## 7.3.2 Defining a Simplified New Filter and Query Language

The input contribution SDS-2022-0014R01 proposes to add a new attribute *advancedQuery* to ***Filter Criteria*** that contains a filter string with a syntax based on S-expressions [i.2]. The new attribute, *advancedQuey*, can be used together in combination with the already existing attributes in ***Filter Criteria*** or alone.

It is an extension of the proposal in clause 7.3.1; the *filterGroup* instances are move from a list attribute to S-expressions.

#### 7.3.2.1 S-expressions

S-expressions are a notation that is based on a nested list (or tree) structure. Those lists may contain either none, one or many elements. Each of the elements can either be value or another S-expression. The first element can also be a function which takes the other following elements of the list as input (after recursively evaluating them). S-expressions are the basis for lisp-like languages, like *Common Lisp*, *Scheme*, or *Closure*.

Examples:

* (1 2 3) : This S-expression is a list with the three values 1, 2, 3
* (== 1 2) : This S-expression evaluates the function == with the parameters 1 and 2. It evaluates to false
* (| (== 1 2) (< 3 4)) : This S-expression evaluates the function | (or) with the result of the two S-Expressions (== 1 2) (that evaluates to *false*) and (< 3 4 (that evaluates to *true*). The whole S-expression evaluates to true.

The advantage of S-expressions are that parsers and interpreters are fairly simple to implement in any programming language, and that implementations have a small footprint.

#### 7.3.2.2 Supported Comparisons

Since S-expressions are evaluated recursively no loop or expression functionalities need to be supported. The number of operations can be limited to a small set of logical and comparison operations. The following list can be extended if needed.

The S-expression for queries and filters in oneM2M has at least one operator followed by an attribute name or an S-expression, optionally followed by one or more values or S-expressions. Attribute names are replaced by their respective values from the evaluated resources during a query:

(<operator> <attribute name or S-expression> [<value or S-expression>]\*)

Table 7.3.2.2-1: Supported Logical Operators

| Operator | Description | Examples |
| --- | --- | --- |
| & | Logical *and*. All included expressions must evaluate to *true* for the expression to evaluate to *true*, otherwise the expression evaluates to *false*. | (& (== ty 1) (in lbl "aLabel")) |
| | | Logical *or*. At least one of the included expressions must evaluate to *true* for the expression to evaluate to *true*, otherwise the expression evaluates to *false*. | (\| (== ty 1) (== ty 2)) |
| ! | Logical *not*. Only one included expression is allowed, which boolean value is negated as a result of the evaluation. | (! (== ty 1)) |

Table 7.3.2.2-2: Supported comparison Operators

| Operator | Description | Examples |
| --- | --- | --- |
| == | Equality comparison. All included and evaluated expressions are compared to each other. Only if all included expressions are equal to each other the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (== ty 1) |
| != | Inequality comparison. All included and evaluated expressions are compared to each other. Only if all included expressions are not equal to each other the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (!= ty 1) |
| < | Less-than comparison. All included and evaluated expressions are compared to each other in order.Only if all included expressions are monotonically increasing the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (< 1 2 3) |
| <= | Less-than-or-equal comparison. All included and evaluated expressions are compared to each other in order. Only if all included expressions are less than or equal to the value of their right expression the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (<= 1 2 2 3) |
| > | Greater-than comparison. All included and evaluated expressions are compared to each other in order.Only if all included expressions are monotonically decreasing the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (> 3 2 1) |
| >= | Greater-than-or-equal comparison. All included and evaluated expressions are compared to each other in order. Only if all included expressions are greater than or equal to the value of their right expression the result of the comparison is *true*, otherwise the expression evaluates to *false*. | (>= 3 2 2 1) |
| in | Contains comparison. The first expression must evaluate to a string or list. All remaining expressions must be contained in that string or list for the expression to evaluate to *true*, , otherwise the expression evaluates to *false*. | (in (1 2 ) 2) |

The following is a list of operators or functions for further extensions that may be included later if needed:

* startsWith, endsWith: compare only the start or beginning of a string
* regular expression matches
* case-insensitive comparisons for strings
* presence: an attribute is present or not, regardless of its value

#### 7.3.2.3 Attribute Paths

To support complex attributes and also parent-child relationships for queries the following syntax options for attribute paths are proposed:

Table 7.3.2.3-1: Attribute paths syntax

| Syntax | Description | Example |
| --- | --- | --- |
| attribute | A single top-level attribute of an evaluate resource | rn: ResourceName |
| attribute.sub-attributeattribute.sub-attribute.sub-sub-attribute… | A single attribute that is part of a complex structure. It can be further down the path of complex attributes | loc.typ: type of m2m:geoCoordinates |
| attribute{}sub-attributeattribute.sub-attribute{}sub-sub-attributeattribute{}sub-attribute{}sub-sub-attribute… | A single attribute that is a member of a list attribute. During evaluation, all the members of a list are evaluated.For a successful comparison of list attributes, only one occurrence in a list needs to successfully fulfil the comparison evaluation. | pc.acr{}acod{}chty: The *chty* attribute in *accessControlObjectDetails*. |
| cattr.”attribute path” | Apply an attribute path to all the direct child-resources of the currently tested resource. All the direct child-resources are |  |
| ^^.”attribute path” | Apply an attribute path to the parent resource of a currently tested resource. | ^^.lbl: The *labels*\*\* attribute of the parent resource. |

**Note**

It is common in lisp-like languages to use the - character in function and attribute names.

If an attribute is not present in a resource, for example in case of an optional attribute, the expression returns NULL, which for a boolean expression evaluates to False.

#### 7.3.2.4 Functions and Iterators

Some functionality is needed to support iterations over direct child resources and comparing their attributes against a comparison. The following table lists those functions.

Table 7.3.2.4-1: Functions and iterators

| Function | Description | Example |
| --- | --- | --- |
| any-child | Iterate over every direct child resource of a currently tested resource and individually evaluate the argument S-expression. The function will evaluate to true if the S-expression evaluates to true for any of the direct child resources of the currently tested resource. This function can be applied recursively to include sub-child resources as well. | (any-child (& (== ty 4) (> con "40")) : Test if any <contentInstance> has a value that is greater than “40”. |
| all-children | Iterate over every direct child resource of a currently tested resource and individually evaluate the argument S-expression. The function will evaluate to true if the S-expresson evaluates to true for all the direct child resources of the currently tested resource. This function can be applied recursively to include sub-child resources as well. | (all-children (== ty 3)) : Test if all the child resources are of type <container>. |

#### 7.3.2.5 Advantages and Disadvantages

* An implementation for an S-expression parser and interpreter could be fairly small and would fit in smaller ASN and MN CSE implementations.
* The size of filter queries formulated as S-expressions is also small. This reduces the size of requests.
* The syntax is simple and easy to understand. The operators and functions can be defined specifically for by oneM2M purposes.
* There is no implementation, framework or library available.
* The specification of the syntax and the operators and functions needs to be maintained by oneM2M.

# 8 Conclusion

TODO

# Annexes

# Annex <A>:Title of annex

<Text>

## First clause of the annex

<Text>

### A.1.1 First subdivided clause of the annex

<Text>

<PAGE BREAK>

# Annex <y>:Bibliography

The annex entitled “Bibliography” is optional.

It shall contain a list of standards, books, articles, or other sources on a particular subject which are not mentioned in the document itself.

It shall not include references mentioned in the document.

* <Publication>: “<Title>”.

OR

<Publication>: “<Title>”.

<PAGE BREAK>

# History

This clause shall be the last one in the document and list the main phases (all additional information will be removed at the publication stage).

| Publication history | Publication history | Publication history |
| --- | --- | --- |
| V1.1.1 | <yyyy-mm-dd> | <Milestone> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

| Draft history (to be removed on publication) | Draft history (to be removed on publication) | Draft history (to be removed on publication) |
| --- | --- | --- |
| V5.0.0 | 2024-09-02 | SDS-2024-0077R01 applied |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |