Document Number: DSP0212

Date: 2013-08-22

Version: 1.0.1

Filter Query Language

Document Type: Specification

Document Status: DMTF Standard

Document Language: en-US

CONTENTS

Tables

Foreword

The *Filter Query Language* (DSP0212) was prepared by the DMTF Architecture Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Jim Davis – WS, Inc. (Editor)
- George Ericson – EMC
- Andreas Maier – IBM
- Karl Schopmeyer – Inova Development

Introduction

The information in this specification should be sufficient for a provider or consumer to be able to utilize the Filter Query Language to filter CIM instances.

The target audience for this specification is implementers of the Filter Query Language.

Document conventions

Typographical conventions

The following typographical conventions are used in this document:

- Document titles are marked in *italics*.
- Important terms that are used for the first time are marked in *italics*.
- ABNF rules and FQL filter queries are in `monospaced font`.

ABNF usage conventions

Format definitions in this document are specified using ABNF (see RFC5234), with the following deviations:

- Literal strings are to be interpreted as case-sensitive Unicode characters, as opposed to the definition in RFC5234 that interprets literal strings as case-insensitive US-ASCII characters, unless otherwise specified.

Experimental material

Experimental material has yet to receive sufficient review to satisfy the adoption requirements set forth by the DMTF. Experimental material is included in this document as an aid to implementers who are interested in likely future developments. Experimental material may change as implementation experience is gained. It is likely that experimental material will be included in an upcoming revision of the specification. Until that time, experimental material is purely informational.

The following typographical convention indicates experimental material:

EXPERIMENTAL

Experimental material appears here.

EXPERIMENTAL

In places where this typographical convention cannot be used (for example, tables or figures), the "EXPERIMENTAL" label is used alone

Filter Query Language

1 Scope

The *Filter Query Language* provides a simple query language for filtering CIM instances.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

DMTF DSP0004, *CIM Infrastructure Specification 2.7,*
http://www.dmtf.org/standards/published_documents/DSP0004_2.7.pdf

DMTF DSP0207, *WBEM URI Mapping 1.0,*
http://www.dmtf.org/standards/published_documents/DSP0207_1.0.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.1,*
http://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf

IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, Jan. 2008,
http://www.ietf.org/rfc/rfc5234.txt

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
http://isotc.iso.org

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Annex H. The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The terms defined in DSP0004 apply to this document. The following additional terms are used in this document.

3.1

filter query

an expression that can be applied to a CIM instance. See 5.2 for details.

4 Symbols and abbreviated terms

The abbreviations defined in DSP0004 apply to this document. The following additional abbreviations are used in this document.

4.1

CQL

CIM Query Language

4.2

FQL

Filter Query Language

4.3

URI

Uniform Resource Identifier

4.4

WBEM

Web Based Enterprise Management

5 Filter Query Language

The Filter Query Language (FQL) is designed to filter a set of CIM instances of a CIM class (including subclasses) based on one or more property values of the class.

FQL has the following goals:

- Leverage the CIM Query Language (CQL) defined in DSP0202 wherever possible.
- The FQL was designed to be simple so that it can quickly be adopted by both implementers and consumers.
- The FQL is not a fully functional query language; use the CIM Query Language defined in DSP0202 if you need a full query language.
- No optional components, everything defined shall be supported.

5.1 Identifying the Filter Query Language

The Filter Query Language shall be identified by the string

```
"DMTF:FQL"
```

following the convention used for other query languages defined by DMTF.

5.2 Filter queries

This subclause describes the FQL filter queries.

5.2.1 General

A *filter query* is an expression that can be evaluated on a CIM instance. The evaluation of a filter query on an instance shall either succeed or fail. The evaluation of invalid filter queries shall fail.

If the evaluation of a filter query on an instance succeeds, the filter query shall evaluate to a boolean value indicating that the instance is either included (if True) or excluded (if False). Note that filter queries that succeed cannot evaluate to Null.

If the evaluation of a filter query on an instance fails, the filter query shall not have an evaluation result. Referencing specifications may define rules for the error handling of filter queries whose evaluation fails.

If a property does not exist in an instance that is being evaluated, the property shall be assumed to be null.

5.2.2 Encoding

FQL filter queries may contain (unescaped) UCS characters (see `UNICODE-CHAR` rule in 5.3.2). The encoding of FQL filter queries is not mandated in this specification.

For example, when an FQL filter query is transported in a communication protocol, the specification defining the protocol will specify acceptable encodings; similarly for APIs.

5.2.3 Whitespace

In FQL, the following characters shall be considered whitespace:

• TAB	(U+0009)
• CR	(U+000D)
• LF	(U+000A)
• SPACE	(U+0020)

For the use of whitespace characters in FQL, see 5.3.2.

5.2.4 Property comparison overview (informative)

At its core, FQL filter queries specify property comparisons. Property comparisons result in a boolean value and can be combined into the (boolean) evaluation result using boolean expressions, possibly overriding precedence of the boolean operators using parenthesis. Expressions in FQL filter queries are limited to combining the boolean results of property comparisons; there are no expressions in the property comparisons. The property comparisons are simple operations such as equality, ordering, pattern-matching or array related operations. For details, see the following subclauses.

5.2.5 Scalar value comparison

A scalar value comparison in a filter query compares two scalar values using equality operators (`"="` and `"<>"`), or ordering operators (`"<"`, `">"`, `"<="` and `">="`).

For example, `Started = True` or `Metric.Threshold > 25`.

Table 1 defines the comparison operators that shall be supported for each data type of the property involved in the scalar value comparison. Filter queries that specify operators other than those listed shall be considered invalid.

The column "Literal syntax" defines the allowable literal syntax for each datatype, referring to the ABNF rules defined in 5.3.2. Filter queries that specify literals that do not conform to these rules shall be considered invalid.

Table 1 - Comparison operators for scalar values

Property data type	Literal syntax	Comparison operators	Remarks
boolean	`boolean-literal`	equality	
integer (uint8 … uint64, sint8 … sint64)	`integer-literal`	equality, ordering	
real (real32, real64)	`real-literal`	equality, ordering	
string (string, char16)	`string-literal`	equality	
string and uint8[] qualified as octet string (OctetString qualifier)	`octetstring-literal`	equality	
string qualified as embedded object (EmbeddedInstance or EmbeddedObject qualifier)	N/A	equality	Not supported for comparison with literals
datetime	`datetime-literal`	equality, ordering	
reference	`reference-literal`	equality	

The semantic of the equality and ordering operators shall conform to DSP0004 subclause 5.2.6 "Comparison of Values" and for datetime typed properties in addition to DSP0004 subclause 5.2.4 "Datetime Type".

Note that DSP0004 permits the ordering operator on more data types than FQL does.

Only datatypes from the same row of Table 1 shall be compatible for scalar value comparison. A filter query shall be considered invalid if the data types used in a scalar value comparison are not compatible (that is, if they are from different rows of Table 1).

For example, comparing a boolean typed property to a string literal will be considered invalid.

5.2.6 Array value comparison

An array value comparison in a filter query compares two array values using equality operators (`"="` and `"<>"`).

For example, `OperationalStates = {2,5}.`

Array value comparison shall conform to the rules in DSP0004 subclause 5.2.6 "Comparison of Values".

5.2.7 Array operators (ANY and EVERY)

The array operators `ANY` and `EVERY` can be applied to array properties and the result is part of a scalar value comparison. The `ANY` operator is used to determine if any of the elements of an array satisfies the comparison. The `EVERY` operator is used to determine if all of the elements of an array satisfy the comparison. The `NOT` operator can be used before an `ANY` or `EVERY` operator and reverses the semantics of the following array operator.

For example, the scalar value comparison `NOT EVERY Temperatures < MaxTemperature` is True if not every array entry of the Temperatures array property is less than the value of the MaxTemperature scalar property.

5.2.8 Pattern matching operator (LIKE)

The `LIKE` operator can be used to match regular expression patterns. The regular expression syntax is defined in DSP1001 Annex B.

5.2.9 Operator precedence

The FQL operators shall have the following precedence, from highest to lowest:

1) NOT
2) array operators (ANY and EVERY)
3) equality and ordering operators and LIKE
4) AND
5) OR

5.3 Grammar

5.3.1 Reserved words

The following words are reserved for FQL. These reserved words shall be treated case insensitively.

```
AND = "AND"
ANY = "ANY"
EVERY = "EVERY"
FALSE = "FALSE"
LIKE = "LIKE"
NOT = "NOT"
NULL = "NULL"
OR = "OR"
TRUE = "TRUE"
```

5.3.2 FQL grammar

Valid FQL filter queries shall conform to the ABNF rule `fql` defined in this subclause and to all constraints defined in this subclause (including constraints defined in ABNF comments). As a consequence, FQL filter queries that do not satisfy these rules need to be considered invalid and need to fail.

The following ABNF rules shall be interpreted to combine their terminals by implicitly inserting zero or more (or between adjacent reserved words, one or more) of the whitespace characters defined in 5.2.3.

```
fql = fql-expr / "(" fql-expr ")" *( bool-op "(" fql-expr ")" )

fql-expr = property-comp *( bool-op property-comp )

property-comp =
    array-property                array-comp-op   array-literal /
    array-property                array-comp-op   array-property /
    scalar-property               scalar-comp-op  scalar-literal /
    scalar-property               scalar-comp-op  scalar-property /
    array-property "[" index "]"  scalar-comp-op  scalar-literal /
    array-property "[" index "]"  scalar-comp-op  scalar-property /
    array-property "[" index "]"  scalar-comp-op  array-property "[" index "]" /
    array-op array-property       scalar-comp-op  scalar-literal /
    array-op array-property       scalar-comp-op  scalar-property /
    array-op array-property       scalar-comp-op  array-property "[" index "]" /
    scalar-property               like-op         like-pattern /
    array-property "[" index "]"  like-op         like-pattern /
    array-op array-property       like-op         like-pattern

scalar-property = property      ; property shall identify a scalar property
```

```
array-property = property       ; property shall identify an array property

index = unsigned-integer        ; the array on which the index is used may be of
                                ; any array type (Bag, Ordered, Indexed)

like-pattern = like-literal

property = property-name *( "." property-name )

; property-name is the name of a property in the CIM instance that is evaluated

scalar-comp-op = "=" / "<>" / "<" / ">" / "<=" / ">="

array-comp-op = "=" / "<>"

like-op = [NOT] LIKE

bool-op = AND / OR

array-op = [NOT] ( ANY / EVERY )

array-literal  = "{" [scalar-literal *( "," scalar-literal ) ] "}"

scalar-literal = boolean-literal / string-literal / integer-literal /
                 real-literal / datetime-literal / reference-literal / NULL
```

The following ABNF rules shall be interpreted to combine their terminals as stated, without implicitly inserting any whitespace characters.

Some alphabetic characters shall be treated case insensitively, as stated. All other alphabetic characters shall be treated case sensitively.

```
boolean-literal = TRUE / FALSE

like-literal = string-literal      ; the literal shall conform to the regular
                                   ; expression syntax defined in DSP1001, Annex B

datetime-literal = string-literal  ; the literal shall conform to the datetime format
                                   ; defined in DSP0004

reference-literal = string-literal ; the literal shall conform to the untyped WBEM URI
                                   ; syntax defined in DSP0207

string-literal = single-quote *( UNICODE-CHAR / char-escape ) single-quote

single-quote = "'"

; UNICODE-CHAR is any UCS character from the ranges:
;   U+0020 .. U+D7FF
;   U+E000 .. U+FFFD
;   U+10000 .. U+10FFFF
; Note that these UCS characters can be represented in XML without any escaping
; (see W3C XML).

char-escape = "\" ( "\" / single-quote / "b" / "t" / "n" / "f" / "r" /
                    "u" 4*6(hex-digit) )

integer-literal = decimal-literal / binary-literal / hex-literal

octetstring-literal = hex-literal

decimal-literal = [sign] unsigned-integer

unsigned-integer = 1*(decimal-digit)

binary-literal = [sign] 1*(binary-digit) "B"                     ; case insensitive

hex-literal = [sign] "0X" 1*( hex-digit hex-digit )              ; case insensitive

real-literal = [sign] exact-numeric [ "E" decimal-value ]        ; case insensitive

exact-numeric = unsigned-integer "." [unsigned-integer] /
                "." unsigned-integer

sign = "+" / "-"

binary-digit = "0" / "1"

decimal-digit = binary-digit / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

hex-digit = decimal-digit / "A" / "B" / "C" / "D" / "E" / "F"    ; case insensitive
```

5.4 Examples (Informative)

- `Started = TRUE`
  evaluates to true when an instance has a boolean property named Started with the value TRUE.

- `Started = TRUE AND StartMode = 'Manual'`
  evaluates to true when an instance has a boolean property named Started with the value TRUE and a string property named StartMode with a value of "Manual".

- `Threshold > 25`
  evaluates to true when an instance has a numeric property named Threshold that has a value greater than 25.

- `CreationClassName NOT LIKE 'CIM_.*'`
  evaluates to true when an instance has a string property named CreationClassName that has a value that does not start with "CIM_".

- `Dedicated = {3,14}`
  evaluates to true when an instance has a numeric array property named Dedicated that has the

values 3,14 (in order).

- `ANY Dedicated = 3 AND ANY Dedicated = 14`
  evaluates to true when an instance has a numeric array property named Dedicated that has the values 3 and14 (in any order) along with zero or more additional values.

- `ANY Dedicated = 3 AND NOT ANY Dedicated = 2`
  evaluates to true when an instance has a numeric array property named Dedicated that includes the value 3 and does not include the value 2.

- `NOT EVERY Dedicated = 5`
  evaluates to true when an instance has a numeric array property named Dedicated that does not have the value 5 for each value in the array.

- `(Started = true and startmode='manual') OR (Started=False and Startmode='Automatic')`
  evaluates to true when an instance has either of the comparisons in parentheses evaluate to true.

- `RequestedState = EnabledState`
  evaluates to true if the property value of EnabledState equals the property value of RequestedState.

- `SystemTime = "20051003112233.000000+000"`
  evaluates to true if the SystemTime property value is "20051003112233.000000+000"; otherwise, false.

- `InstallDate > "20051003112233.000000+000"`
  evaluates to true if the property InstallDate is later than "20051003112233.000000+000"; otherwise, false.

- `SourceInstance.RequestedState = 5`
  evaluates to true if the embedded instance referenced by the SourceInstance property has a property named RequestedState that has a value of 5.

ANNEX A
(informative)

Change log

Version	Date	Description
1.0.0	2012-12-13	
1.0.1	2013-08-22	Released as DMTF Standard with the following changes 1) Eliminate option to qualify a property by class name 2) Add option to do array compares with like 3) Clarified that property evaluation is against what is in the instance being compared. 4) Added informative next to examples 5) Fixed example text to match syntax 6) Added example for embedded instance

Bibliography

DMTF DSP0202, *CIM Query Language Specification 1.0*, http://www.dmtf.org/standards/published_documents/DSP0202_1.0.pdf

W3C XML, *Extensible Markup Language (XML) 1.0*, http://www.w3.org/TR/REC-xml/