



1

2

3

4

Document Number: DSP0212

Date: 2013-08-22

Version: 1.0.1

5 **Filter Query Language**

6 **Document Type: Specification**

7 **Document Status: DMTF Standard**

8 **Document Language: en-US**

9

10 Copyright notice

11 Copyright © 2012-2013 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

12 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
13 management and interoperability. Members and non-members may reproduce DMTF specifications and
14 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to
15 time, the particular version and release date should always be noted.

16 Implementation of certain elements of this standard or proposed standard may be subject to third party
17 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
18 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
19 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
20 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
21 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
22 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
23 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
24 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
25 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
26 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
27 implementing the standard from any and all claims of infringement by a patent owner for such
28 implementations.

29 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
30 such patent may relate to or impact implementations of DMTF standards, visit
31 <http://www.dmtf.org/about/policies/disclosures.php>.

32

33

CONTENTS

34	Foreword	4
35	Introduction.....	5
36	1 Scope	7
37	2 Normative references	7
38	3 Terms and definitions	7
39	4 Symbols and abbreviated terms.....	8
40	5 Filter Query Language.....	8
41	5.1 Identifying the Filter Query Language.....	8
42	5.2 Filter queries	8
43	5.2.1 General	8
44	5.2.2 Encoding.....	9
45	5.2.3 Whitespace	9
46	5.2.4 Property comparison overview (informative)	9
47	5.2.5 Scalar value comparison	9
48	5.2.6 Array value comparison	10
49	5.2.7 Array operators (ANY and EVERY)	10
50	5.2.8 Pattern matching operator (LIKE)	10
51	5.2.9 Operator precedence	11
52	5.3 Grammar.....	11
53	5.3.1 Reserved words	11
54	5.3.2 FQL grammar.....	11
55	5.4 Examples	13
56	ANNEX A (informative) Change log.....	15
57	Bibliography	16
58		
59	Tables	
60	Table 1 - Comparison operators for scalar values.....	10
61		

62

Foreword

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

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

66 Acknowledgments

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

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

72

Introduction

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

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

76 Document conventions

77 Typographical conventions

78 The following typographical conventions are used in this document:

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

82 ABNF usage conventions

83 Format definitions in this document are specified using ABNF (see [RFC5234](#)), with the following
84 deviations:

- 85 • Literal strings are to be interpreted as case-sensitive Unicode characters, as opposed to the
86 definition in [RFC5234](#) that interprets literal strings as case-insensitive US-ASCII characters,
87 unless otherwise specified.

88 Experimental material

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

94 The following typographical convention indicates experimental material:

95 **EXPERIMENTAL**

96 Experimental material appears here.

97 **EXPERIMENTAL**

98 In places where this typographical convention cannot be used (for example, tables or figures), the
99 "EXPERIMENTAL" label is used alone
100

102

Filter Query Language

103 1 Scope

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

105 2 Normative references

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

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

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

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

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

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

120 3 Terms and definitions

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

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

129 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
130 described in [ISO/IEC Directives, Part 2](#), Clause 5.

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

134 The terms defined in [DSP0004](#) apply to this document. The following additional terms are used in this
135 document.

136 3.1

137 filter query

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

139 4 Symbols and abbreviated terms

140 The abbreviations defined in [DSP0004](#) apply to this document. The following additional abbreviations are
141 used in this document.

142 4.1

143 CQL

144 CIM Query Language

145 4.2

146 FQL

147 Filter Query Language

148 4.3

149 URI

150 Uniform Resource Identifier

151 4.4

152 WBEM

153 Web Based Enterprise Management

154 5 Filter Query Language

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

157 FQL has the following goals:

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

164 5.1 Identifying the Filter Query Language

165 The Filter Query Language shall be identified by the string

166 "DMTF:FQL"

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

168 5.2 Filter queries

169 This subclause describes the FQL filter queries.

170 5.2.1 General

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

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

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

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

180 5.2.2 Encoding

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

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

185 5.2.3 Whitespace

186 In FQL, the following characters shall be considered whitespace:

- 187 • TAB (U+0009)
- 188 • CR (U+000D)
- 189 • LF (U+000A)
- 190 • SPACE (U+0020)

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

192 5.2.4 Property comparison overview (informative)

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

199 5.2.5 Scalar value comparison

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

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

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

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

209

Table 1 - Comparison operators for scalar values

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

210 The semantic of the equality and ordering operators shall conform to [DSP0004](#) subclause 5.2.6
 211 "Comparison of Values" and for datetime typed properties in addition to [DSP0004](#) subclause 5.2.4
 212 "Datetime Type".

213 Note that [DSP0004](#) permits the ordering operator on more data types than FQL does.

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

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

218 5.2.6 Array value comparison

219 An array value comparison in a filter query compares two array values using equality operators ("`=`" and
 220 "`<>`").

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

222 Array value comparison shall conform to the rules in [DSP0004](#) subclause 5.2.6 "Comparison of Values".

223 5.2.7 Array operators (ANY and EVERY)

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

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

232 5.2.8 Pattern matching operator (LIKE)

233 The `LIKE` operator can be used to match regular expression patterns. The regular expression syntax is
 234 defined in [DSP1001](#) Annex B.

235 5.2.9 Operator precedence

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

- 237 1) NOT
- 238 2) array operators (ANY and EVERY)
- 239 3) equality and ordering operators and LIKE
- 240 4) AND
- 241 5) OR

242 5.3 Grammar

243 5.3.1 Reserved words

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

```
245 AND = "AND"
246 ANY = "ANY"
247 EVERY = "EVERY"
248 FALSE = "FALSE"
249 LIKE = "LIKE"
250 NOT = "NOT"
251 NULL = "NULL"
252 OR = "OR"
253 TRUE = "TRUE"
```

254 5.3.2 FQL grammar

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

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

```
261 fql = fql-expr / "(" fql-expr ")" *( bool-op "(" fql-expr ")" )
262
263 fql-expr = property-comp *( bool-op property-comp )
264
265 property-comp =
266     array-property          array-comp-op  array-literal /
267     array-property          array-comp-op  array-property /
268     scalar-property         scalar-comp-op scalar-literal /
269     scalar-property         scalar-comp-op scalar-property /
270     array-property "[" index "]" scalar-comp-op scalar-literal /
271     array-property "[" index "]" scalar-comp-op scalar-property /
272     array-property "[" index "]" scalar-comp-op array-property "[" index "]" /
273     array-op array-property scalar-comp-op scalar-literal /
274     array-op array-property scalar-comp-op scalar-property /
275     array-op array-property scalar-comp-op array-property "[" index "]" /
276     scalar-property         like-op       like-pattern /
277     array-property "[" index "]" like-op   like-pattern /
278     array-op array-property like-op       like-pattern
279
280 scalar-property = property ; property shall identify a scalar property
```

```

281
282 array-property = property      ; property shall identify an array property
283
284 index = unsigned-integer      ; the array on which the index is used may be of
285                               ; any array type (Bag, Ordered, Indexed)
286
287 like-pattern = like-literal
288
289 property = property-name *( "." property-name )
290
291 ; property-name is the name of a property in the CIM instance that is evaluated
292
293 scalar-comp-op = "=" / "<>" / "<" / ">" / "<=" / ">="
294
295 array-comp-op = "=" / "<>"
296
297 like-op = [NOT] LIKE
298
299 bool-op = AND / OR
300
301 array-op = [NOT] ( ANY / EVERY )
302
303 array-literal = "{" [scalar-literal *( "," scalar-literal ) ] "}"
304
305 scalar-literal = boolean-literal / string-literal / integer-literal /
306                 real-literal / datetime-literal / reference-literal / NULL

```

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

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

```

311 boolean-literal = TRUE / FALSE
312
313 like-literal = string-literal      ; the literal shall conform to the regular
314                               ; expression syntax defined in DSP1001, Annex B
315
316 datetime-literal = string-literal ; the literal shall conform to the datetime format
317                               ; defined in DSP0004
318
319 reference-literal = string-literal ; the literal shall conform to the untyped WBEM URI
320                               ; syntax defined in DSP0207
321
322 string-literal = single-quote *( UNICODE-CHAR / char-escape ) single-quote
323
324 single-quote = "'"
325
326 ; UNICODE-CHAR is any UCS character from the ranges:
327 ;   U+0020 .. U+D7FF
328 ;   U+E000 .. U+FFFD
329 ;   U+10000 .. U+10FFFF
330 ; Note that these UCS characters can be represented in XML without any escaping

```

```

331 ; (see W3C XML).
332
333 char-escape = "\" ( "\" / single-quote / "b" / "t" / "n" / "f" / "r" /
334                "u" 4*6(hex-digit) )
335
336 integer-literal = decimal-literal / binary-literal / hex-literal
337
338 octetstring-literal = hex-literal
339
340 decimal-literal = [sign] unsigned-integer
341
342 unsigned-integer = 1*(decimal-digit)
343
344 binary-literal = [sign] 1*(binary-digit) "B" ; case insensitive
345
346 hex-literal = [sign] "0X" 1*( hex-digit hex-digit ) ; case insensitive
347
348 real-literal = [sign] exact-numeric [ "E" decimal-value ] ; case insensitive
349
350 exact-numeric = unsigned-integer "." [unsigned-integer] /
351                "." unsigned-integer
352
353 sign = "+" / "-"
354
355 binary-digit = "0" / "1"
356
357 decimal-digit = binary-digit / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
358
359 hex-digit = decimal-digit / "A" / "B" / "C" / "D" / "E" / "F" ; case insensitive
360

```

361 5.4 Examples (Informative)

- 362 • Started = TRUE
363 evaluates to true when an instance has a boolean property named Started with the value TRUE.
364
- 365 • Started = TRUE AND StartMode = 'Manual'
366 evaluates to true when an instance has a boolean property named Started with the value TRUE and
367 a string property named StartMode with a value of "Manual".
368
- 369 • Threshold > 25
370 evaluates to true when an instance has a numeric property named Threshold that has a value
371 greater than 25.
372
- 373 • CreationClassName NOT LIKE 'CIM_.*'
374 evaluates to true when an instance has a string property named CreationClassName that has a
375 value that does not start with "CIM_".
376
- 377 • Dedicated = {3,14}
378 evaluates to true when an instance has a numeric array property named Dedicated that has the

- 379 values 3,14 (in order).
380
- 381 • `ANY Dedicated = 3 AND ANY Dedicated = 14`
382 evaluates to true when an instance has a numeric array property named `Dedicated` that has the
383 values 3 and 14 (in any order) along with zero or more additional values.
384
 - 385 • `ANY Dedicated = 3 AND NOT ANY Dedicated = 2`
386 evaluates to true when an instance has a numeric array property named `Dedicated` that includes the
387 value 3 and does not include the value 2.
388
 - 389 • `NOT EVERY Dedicated = 5`
390 evaluates to true when an instance has a numeric array property named `Dedicated` that does not
391 have the value 5 for each value in the array.
392
 - 393 • `(Started = true and startmode='manual') OR (Started=False and`
394 `Startmode='Automatic')`
395 evaluates to true when an instance has either of the comparisons in parentheses evaluate to true.
396
 - 397 • `RequestedState = EnabledState`
398 evaluates to true if the property value of `EnabledState` equals the property value of `RequestedState`.
399
 - 400 • `SystemTime = "20051003112233.000000+000"`
401 evaluates to true if the `SystemTime` property value is "20051003112233.000000+000"; otherwise,
402 false.
403
 - 404 • `InstallDate > "20051003112233.000000+000"`
405 evaluates to true if the property `InstallDate` is later than "20051003112233.000000+000"; otherwise,
406 false.
407
 - 408 • `SourceInstance.RequestedState = 5`
409 evaluates to true if the embedded instance referenced by the `SourceInstance` property has a
410 property named `RequestedState` that has a value of 5.

411
412
413
414
415

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 <ol style="list-style-type: none">1) Eliminate option to qualify a property by class name2) Add option to do array compares with like3) Clarified that property evaluation is against what is in the instance being compared.4) Added informative next to examples5) Fixed example text to match syntax6) Added example for embedded instance

416

Bibliography

- 417 DMTF DSP0202, *CIM Query Language Specification 1.0*,
418 http://www.dmtf.org/standards/published_documents/DSP0202_1.0.pdf
- 419 W3C XML, *Extensible Markup Language (XML) 1.0*,
420 <http://www.w3.org/TR/REC-xml/>