[MS-FSFQL]:
Fast Query Language Structure

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Editorial	Revised and edited the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the structure of the FAST Query Language (FQL), which the Query and Result Protocol uses as described in [MS-FSQR]. FQL is a language for expressing search criteria.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Augmented Backus-Naur Form (ABNF)
UTF-8

The following terms are defined in [MS-OFCGLOS]:

default index
dynamic rank
dynamic teaser
internal property
managed property
query processing
search service application
stemming
token

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-FSSCFG] Microsoft Corporation, "Search Configuration File Format Specification", November 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.ietf.org/rfc/rfc5234.txt

1.2.2 Informative References

[MS-FSQR] Microsoft Corporation, "Query and Result Protocol Specification", November 2009.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary", June 2008.

[MS-SEARCH] Microsoft Corporation, "Search Protocol Specification", June 2008.

1.3 Structure Overview (Synopsis)

Application implementers use FQL to express criteria for searching. A typical scenario for using FQL is an application that enables users to search for items and browse through results.

An FQL expression consists of search **tokens** and operators. A search token consists of a value or range of values to search for, and an operator specifies how to include, exclude, and rank the search results. Examples of operators include **and**, **andnot**, **or**, **not**, and **near**.

The **and** operator applies when the user wants items that match all operands.

A search query that uses the **andnot** operator returns items that match only the first operand, and it excludes items that match subsequent operands.

An **or** operator expression returns items that match any of the operands.

The **not** operator excludes items that match any of the operands.

The **near** operator matches items based on the proximity of indexed tokens that match the operands.

An FQL expression consists of either a single search token or a single operator expression. Many operators accept FQL expressions as parameters, which permits FQL expressions to be nested.

1.4 Relationship to Protocols and Other Structures

The Query and Result protocol uses FQL as described in [MS-FSQR].

1.5 Applicability Statement

Application implementers use FQL for searches when they use the Query and Result protocol as described in [MS-FSQR]—if the SharePoint Search keyword syntax as described in [MS-SEARCH] section 2.2.10 does not provide the capabilities that they need. FQL is not a search language for end users.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

None.

2 Structures

An FQL expression consists of search tokens and operators. A search token consists of a value or range of values to search for, and an operator specifies how to include, exclude, or rank the search results.

The **query processing** component evaluates each token according to its type, which is expressed either implicitly or explicitly.

An operator MUST precede its operands. The operands MUST be comma-delimited and contained within parentheses. Where noted in the following subsections, valid operands include named parameters that consist of a name and value separated by an equal sign.

Although FQL keywords are not case sensitive, lowercase is suggested for future compatibility. To be interpreted as a search token, a keyword MUST be contained within double quotation marks. Any word that is not a keyword MUST be interpreted as a search token.

The following words are FQL operators and keywords:

- **and**
- **andnot**
- **any**
- **count**
- **datetime**
- **decimal**
- **double**
- **ends-with**
- **equals**
- **filter**
- **float**
- **int**
- **int32**
- **max**
- **min**
- **near**
- **not**
- **onear**
- **or**
- **phrase**

- **range**
- **rank**
- **scope**
- **starts-with**
- **string**
- **uint32**

Unless an FQL expression is qualified with the **:** operator as specified in section 2.1, the **search service application** MUST search the **default index** as specified in [MS-FSSCFG] section 2.8.3.8.

The structure of an FQL expression corresponds to the following rules, which themselves conform to **Augmented Backus-Naur Form (ABNF)** as specified in [RFC5234].

```
fql-expression  = (operator-expression / paren-expression / token)

operator-expression = [in-expression] (and / andnot / any / or / rank
    / xrank / near / onear / not / equals / filter / starts-with
    / ends-with / count)

paren-expression = [in-expression] "(" fql-expression ")"

token = [in-expression] (datetime-token / float-token / int-token
    / phrase-token / range-token / string-token)

; Operator expressions
and = "and" "(" multiple-fql-params ")"
andnot = "andnot" "(" multiple-fql-params ")"
any = "any" "(" multiple-fql-params ")"
or = "or" "(" multiple-fql-params ")"

rank = "rank" "(" rank-param *("," rank-param) ")"
rank-param = token / fql-expression

xrank = "xrank" "(" xrank-param *("," xrank-param) ")"
xrank-param = ("boost" "=" integer-value)
    / ("boostall" "=" yesno-value)
    / fql-expression

near = "near" "(" near-param *("," near-param) ")"
near-param = ("N" "=" token-distance) / fql-expression

onear = "onear" "(" onear-param *("," onear-param) ")"
onear-param = ("N" "=" token-distance) / fql-expression

not = "not" "(" fql-expression ")"

count = ("count" "(" token
    1*("," (("from" "=" int-token) / ("to" "=" int-token))) ")")

equals = "equals" "("
    [in-expression] (string-token / phrase-token) ")"
starts-with = "starts-with" "("
    [in-expression] (string-token / phrase-token) ")"
ends-with = "ends-with" "("
```

```
    [in-expression] (string-token / phrase-token) ")"
filter = "filter" "(" fql-expression ")"

; Token operator expressions
phrase-token = "phrase" "(" phrase-token-param
    *("," phrase-token-param) ")"
phrase-token-param = ("weight" "=" unsigned-integer-value)
    / ("linguistics" "=" onoff-value)
    / ("wildcard" "=" onoff-value)
    / token

string-token = explicit-string-token / implicit-string-token
explicit-string-token = "string" "(" string-token-param
    *("," string-token-param) ")"
string-token-param = ("mode" "=" mode-value)
    / ("N" "=" token-distance)
    / ("weight" "=" integer-value)
    / ("linguistics" "=" onoff-value)
    / ("wildcard" "=" onoff-value)
    / token
implicit-string-token = string-value

float-token = explicit-float-token / implicit-float-token
explicit-float-token = "float" "(" (float-value
    / (DQUOTE float-value DQUOTE)) ")"
implicit-float-token = *DIGIT "." 1*DIGIT

int-token = explicit-int-token / implicit-int-token
explicit-int-token = "int" "(" (integer-value
    / (DQUOTE integer-value DQUOTE)
    / (DQUOTE integer-value *(SP integer-value) DQUOTE ","
    numeric-or-mode)
    / (numeric-or-mode "," DQUOTE 1*integer-value *(SP integer-value)
    DQUOTE))
    ")"
implicit-int-token = integer-value

datetime-token = explicit-datetime-token / implicit-datetime-token
explicit-datetime-token = "datetime" "(" (datetime-value
    / (DQUOTE datetime-value DQUOTE)) ")"
implicit-datetime-token = datetime-value

range-token = "range" "(" range-token-param *("," range-token-param)
    ")"
range-token-param = ("from" "=" from-condition)
    / ("to" "=" to-condition)
    / range-limit
range-limit = datetime-token / float-token / int-token
    / "min" / "max"
from-condition = unquoted-from-condition
    / (DQUOTE unquoted-from-condition DQUOTE)
unquoted-from-condition = "GE" / "GT"
to-condition = unquoted-to-condition
    / (DQUOTE unquoted-to-condition DQUOTE)
unquoted-to-condition = "LE" / "LT"

; Data types
string-value = quoted-string-value / unquoted-string-value
```

```
; <quoted-string-value> can contain any characters
; (including wide characters) that are not control
; characters, except for double quotation marks.
quoted-string-value = DQUOTE 1*(quoted-escaped-character
    / %x20-21 / %x23-ffffffff) DQUOTE
quoted-escaped-character =
    quoted-escaped-backslash
    / quoted-escaped-newline
    / quoted-escaped-carriage-return
    / quoted-escaped-tab
    / quoted-escaped-backspace
    / quoted-escaped-form-feed
    / quoted-escaped-double-quote
    / quoted-escaped-single-quote

quoted-escaped-backslash = "\\"
quoted-escaped-newline = "\n"
quoted-escaped-carriage-return = "\r"
quoted-escaped-tab = "\t"
quoted-escaped-backspace = "\b"
quoted-escaped-form-feed = "\f"
quoted-escaped-double-quote = "\" DQUOTE
quoted-escaped-single-quote = "'"

; <unquoted-string-value> can contain any characters (including wide
; characters) that are not control characters, except for spaces,
; commas, double quotation marks, parentheses,colons, and equals
; signs.
unquoted-string-value =
    1*(%x21 / %x23-27 / %x2a-2b / %x2d-39 / %x3b-3c / %x3e-ffffffff)
integer-value = ["-" / "+"] 1*DIGIT
unsigned-integer-value = 1*DIGIT
float-value = ["-" / "+"] (*DIGIT "." 1*DIGIT) / 1*DIGIT

datetime-value = year "-" month "-" day
    ["T" hour ":" minute ":" second ["Z"]]
year = 4*DIGIT            ; four-digit or longer year (0000-infinity)
month = ("0" DIGIT)       ; two-digit month (00-09)
    / ("1" %x30-32)       ; two-digit month (10-12)
day = (%x30-32 DIGIT)     ; two-digit day (00-29)
    / ("3" %x30-31)       ; two-digit day (30-31)
hour = (%x30-31 DIGIT)    ; two-digit hour (00-19)
    / ("2" %x30-33)       ; two-digit hour (20-23)
minute = (%x30-35 DIGIT)  ; two-digit minute (00-59)
second = (%x30-35 DIGIT)  ; two-digit second (00-59)

yesno-value = quoted-yesno-value / unquoted-yesno-value
quoted-yesno-value = DQUOTE unquoted-yesno-value DQUOTE
unquoted-yesno-value = "YES" / "NO"

onoff-value = quoted-onoff-value / unquoted-onoff-value
quoted-onoff-value = DQUOTE unquoted-onoff-value DQUOTE
unquoted-onoff-value = "ON" / "OFF"

; <mode-value> MUST be inside double quotation marks.
mode-value = DQUOTE ("PHRASE" / "AND" / "OR" / "ANY" / "NEAR"
    / "ONEAR" / "SIMPLEANY" / "SIMPLEALL") DQUOTE

; General syntax elements
```

```
in-expression = ((internal-property-name / property-name) ":")
    / (DQUOTE (internal-property-name / property-name) DQUOTE ":")
numeric-or-mode = "mode" "=" DQUOTE "OR" DQUOTE
token-distance = unsigned-integer-value
internal-property-name = property-name "." property-name
property-name = 1*(ALPHA / DIGIT)
multiple-fql-params = fql-expression 1*("," fql-expression)
```

For readability, the preceding rules assume that no extra white space exists in the FQL expression. However, FQL does permit white space to immediately precede and follow parentheses, commas, operators, keywords, and tokens.

Also, although ABNF as specified in [RFC5234] does not explicitly support any encoding other than US-ASCII, the **quoted-string-value** and **unquoted-string-value** data types support wide character values that have **UTF-8** encoding.

2.1 Operators

2.1.1 : Operator

The **:** operator functions as an "in" operator. The name of a **managed property** or an **internal property** MUST precede the **:** operator, and a Boolean operator, a token, or a parenthetical expression MUST follow the **:** operator. The **:** operator specifies that the subsequent operator, token, or parenthetical expression MUST match the specified managed property or internal property (unless another **:** operator overrides that **:** operator).

2.1.2 and Operator

The **and** operator MUST specify two or more FQL expression operands. To be returned as a match, an item MUST match all of the operands.

2.1.3 andnot Operator

The **andnot** operator MUST specify two or more FQL expression operands. To be returned as a match, an item MUST match the first operand but MUST NOT match any of the subsequent operands.

2.1.4 any Operator

The **any** operator MUST specify two or more FQL expression operands. To be returned as a match, an item MUST match at least one of the operands. The **dynamic rank** SHOULD be based on the single best-matching operand.

The method that is used to identify the best match is implementation-dependent. It can involve such factors as frequency and position of tokens that match the operand.

2.1.5 count Operator

The **count** operator MUST specify exactly one operand, which in turn MUST specify a string token or phrase token to be matched.

The value of the *from* named parameter MUST be a positive integer that specifies the minimum number of times that the specified string token or phrase token MUST be matched. If the *from* parameter is not specified, no lower limit will exist.

The value of the *to* named parameter MUST be a positive integer that specifies the non-inclusive maximum number of times that the specified string token or phrase token MUST be matched. (For example, a *to* value of 11 specifies 10 times or fewer). If the *to* parameter is not specified, no upper limit will exist.

2.1.6 ends-with Operator

The **ends-with** operator MUST specify exactly one operand, which in turn MUST specify a string token or a phrase token. The **ends-with** operator MUST match only managed properties that end with the specified string token or phrase token.

2.1.7 equals Operator

The **equals** operator MUST specify exactly one operand, which in turn MUST specify a string token or a phrase token. The **equals** operator MUST match only managed properties that contain the specified string token or phrase token and that do not contain any extra indexed tokens.

2.1.8 filter Operator

The **filter** operator MUST specify exactly one operand. The **filter** operator is for querying metadata or parametric content that is not expressed in a natural language.

When a query processing component evaluates the **filter** operator, the linguistic features MUST be off by default, ranking MUST be disabled, and highlighting MUST NOT be applied to the **dynamic teaser**.

2.1.9 near Operator

The **near** operator MUST specify two or more operands, which in turn MUST each specify an expression to be matched.

If it is specified, the *N* named parameter specifies the maximum number of interspersed, unmatched, indexed tokens.

To match the operands of the **near** operator, the managed property MUST match all of the specified expressions, with no more than the specified number of interspersed, unmatched, indexed tokens.

The operands of the **near** operator MUST NOT be **and**, **andnot**, **not**, or **range** operators.

If two operands match the same indexed token, the matches MUST be considered near each other.

2.1.10 not Operator

The **not** operator MUST specify exactly one FQL expression operand. To be returned as a match, an item MUST NOT match the operand.

2.1.11 onear Operator

The **onear** (ordered near) operator functions in the same way that the **near** operator does (as specified in section 2.1.9), except that each operand MUST match the searched items in the specified order.

For example, an **onear** operation on the string tokens "string1" and "string2" with a **token-distance** value of 1 MUST match "string1 string2", but MUST NOT match "string2 string1".

2.1.12 or Operator

The **or** operator MUST specify two or more FQL expression operands. To be returned as a match, an item MUST match at least one of the operands. Each matching operand SHOULD increase the item's dynamic rank. The degree of increase is implementation specific.

2.1.13 rank Operator

The **rank** operator MUST specify exactly one FQL expression operand in addition to one or more string or phrase operands for which to boost the dynamic rank.

The FQL expression operand MUST contribute to the dynamic rank in the same way that it would without the **rank** operator.

The **rank** operator MUST match only those items that match the FQL expression operand.

The **rank** operator SHOULD increase the dynamic rank of any items that match both the operand and the specified tokens. The degree of increase is implementation specific. The tokens MUST NOT affect which items are returned—only their dynamic ranks.

2.1.14 starts-with Operator

The **starts-with** operator MUST specify exactly one operand, which in turn MUST specify a string token or phrase token to be matched. The **starts-with** operator MUST match only managed properties that start with the specified string token or phrase token.

2.1.15 xrank Operator

The **xrank** operator MUST specify one or more FQL expression operands. The first operand MUST contribute to the dynamic rank in the same way that it would without the **xrank** operator. Subsequent operands MUST NOT affect which items are returned as matches.

The **xrank** operator MUST match only those items that match the first operand. If a matched item also matches subsequent operands, that item's dynamic rank SHOULD increase by the value that the *boost* parameter specifies.

The named parameters in the following table are valid with the **xrank** operator.

Named parameter	Default value	Description
boost	100	Specifies that the dynamic rank of matching items SHOULD increase by the specified number. A negative value MUST set the rank of each matching item to 0.
boostall	"no"	Specifies whether the increase in dynamic rank SHOULD apply to items that do not already have a dynamic rank.

The values in the following table are valid for the *boostall* named parameter.

Value	Description
yes	Specifies that the increase in dynamic rank SHOULD apply to all the matching items, even if they do not already have dynamic rank values.
no	Specifies that the increase in dynamic rank SHOULD apply only to those matching items that

Value	Description
	already have dynamic rank values.

2.1.16 Token Operators

2.1.16.1 datetime Token Operator

The **datetime** token operator MUST specify exactly one operand, which in turn MUST specify a token value. The token value MUST be a valid **datetime-value** as specified by the ABNF rules in section 2.

The **datetime** token operator MUST be assumed for any valid **datetime-value** that is not enclosed in double quotation marks.

Every **datetime-value** MUST be specified according to Coordinated Universal Time.

2.1.16.2 float Token Operator

The **float** token operator MUST specify exactly one operand, which in turn MUST specify a token value.

The **float** token operator MUST be assumed for numeric text that contains a decimal point, unless that text is enclosed in double quotation marks.

2.1.16.3 int Token Operator

The **int** token operator MUST specify exactly one operand, which in turn MUST specify a token value.

If the mode named parameter is specified to contain a value of "OR", the token value MUST be a space-delimited list of token values that are enclosed in double quotation marks and MUST be evaluated as if the values were operands specified by the **or** operator.

The **int** token operator MUST be assumed for numeric text that is not enclosed in double quotation marks, unless that text contains a decimal point.

2.1.16.4 phrase Token Operator

The **phrase** token operator MUST specify one or more operands, which in turn MUST each specify either a string token or an **or** operator.

The **phrase** operator specifies that the managed property MUST contain indexed tokens that match the operands, uninterrupted and in the exact order in which they are specified.

The **phrase** operator supports the *weight*, *linguistics*, and *wildcard* named parameters as specified in section 2.1.15.

2.1.16.5 range Token Operator

The **range** token operator MUST specify two numeric operands of the same type (**float**, **int**, or **datetime**), and the managed property being searched MUST be of a compatible type. The first operand specifies the range start, and the second operand specifies the range end.

The named parameters in the following table are valid with the **range** operator.

Named parameter	Default value	Description
from	"GE"	Specifies the condition for evaluating the **start** operand.
to	"LT"	Specifies the condition for evaluating the **end** operand.

The values in the following table are valid for the *from* named parameter.

Value	Description
"GE"	Specifies that matching values MUST be greater than or equal to the value of the **start** operand.
"GT"	Specifies that matching values MUST be greater than the value of the **start** operand.

The values in the following table are valid for the *to* named parameter.

Value	Description
"LE"	Specifies that matching values MUST be less than or equal to the value of the **end** operand.
"LT"	Specifies that matching values MUST be less than the value of the **end** operand.

2.1.16.6 string Token Operator

The **string** token operator MUST specify exactly one operand, which in turn MUST specify a token value. The operand is case insensitive. That is, a query processing component MUST ignore case when it compares the operand to the searched items.

If the operand is numeric, it MUST be converted to a string and evaluated as such.

The **string** token operator MUST be assumed for text that is not enclosed in double quotation marks, unless that text is a keyword or contains another explicit token operator. The **string** token operator MUST be assumed for all text that is enclosed in double quotation marks.

The named parameters in the following table are valid with the **string** token operator.

Named parameter	Default value	Description
mode	"PHRASE"	Specifies how the text operand MUST be evaluated. The value of the *mode* named parameter MUST be enclosed within double quotation marks.
N	4	Specifies the maximum token distance. This named parameter applies only if the value of *mode* is "NEAR" or "ONEAR".
weight	100	Specifies a positive integer, which in turn specifies the relative weight of the dynamic rank of this string token.
linguistics	"ON"	Specifies whether linguistic features will be enabled when a query processing component evaluates the string.
wildcard	"ON"	Specifies whether to support wildcards in the string.

The values in the following table are valid for the *mode* named parameter.

Value	Description
"PHRASE"	Specifies that the text MUST be evaluated as a phrase. Using this value is equivalent to using the **phrase** operator.
"AND"	Specifies that the text MUST be evaluated as a list of tokens provided to the **and** operator.
"OR"	Specifies that the text MUST be evaluated as a list of tokens provided to the **or** operator.
"ANY"	Specifies that the text MUST be evaluated as a list of tokens provided to the **any** operator.
"NEAR"	Specifies that the text MUST be evaluated as a list of tokens provided to the **near** operator.
"ONEAR"	Specifies that the text MUST be evaluated as a list of tokens provided to the **onear** operator.

The values in the following table are valid for the *linguistics* named parameter.

Value	Description
"ON"	Specifies that linguistic features MUST be applied.
"OFF"	Specifies that linguistic features MUST NOT be applied.

The values in the following table are valid for the *wildcard* named parameter.

Value	Description
"ON"	Specifies that the characters "?" and "*" MUST be evaluated as wildcards. A "?" character matches any single character, and a "*" character matches zero or more characters.
"OFF"	Specifies that the characters "?" and "*" MUST NOT be evaluated as wildcards.

The escaped strings in the following table are valid within quoted strings to represent reserved characters.

Escaped string	Hexadecimal character code	Description
\\	5C	Backslash.
\n	0A	Line feed, or newline.
\r	0D	Carriage return.
\t	09	Tab.
\b	08	Backspace.
\f	0C	Form feed.
\"	22	Double quotation mark.
\'	27	Single quotation mark, or apostrophe.

2.2 Keywords

2.2.1 max Keyword

When specified in place of a numeric value, the **max** keyword MUST represent the maximum value for the expected type.

2.2.2 min Keyword

When specified in place of a numeric value, the **min** keyword MUST represent the minimum value for the expected type.

3 Structure Examples

For readability, the FQL examples in the following subsections are not escaped as described in [MS-FSQR] section 3.1.4.1.2.1.1. An example of an escaped FQL expression is available. For more details, see [MS-FSQR] section 4.1.2.

3.1 Operator Examples

3.1.1 : Operator Examples

Each of the following expressions matches items that have both "much" and "nothing" in the **title** managed property.

```
title:and(much, nothing)
and(title:much, title:nothing)
title:string("much nothing", mode="and")
```

3.1.2 and Operator Example

The following expression matches items for which the default index contains "cat", "dog", and "fox".

```
and(cat, dog, fox)
```

3.1.3 andnot Operator Examples

The following expression matches items for which the default index contains "cat" but not "dog".

```
andnot(cat, dog)
```

The following expression matches items for which the default index contains "dog" but neither "beagle" nor "chihuahua".

```
andnot(dog, beagle, chihuahua)
```

3.1.4 any Operator Examples

The following expression matches items for which the default index contains "cat" or "dog".

```
any(cat, dog)
```

Based on the preceding expression, if an item’s default index contains both "cat" and "dog" but "cat" is considered a better match, the item’s dynamic rank will be based on "cat" with no consideration given to "dog".

3.1.5 count Operator Examples

The following expression matches at least 5 occurrences of the word "cat".

```
count(cat, from=5)
```

The following expression matches at least 5 but not 10 or more occurrences of the word "cat".

```
count(cat, from=5, to=10)
```

Each of the following expressions matches at least 3 occurrences of a certain word—and that word can be either "cat" or "dog".

```
count(or(cat, dog), from=3)
count(string("cat dog", mode="or"), from=3)
```

The following table contains examples of managed property string values and states whether they match the three preceding expressions.

Match?	Text
Yes	My cat likes my dog, but my dog hates my cat.
No	My bird likes my newt, but my dog hates my cat.

3.1.6 ends-with Operator Example

The following expression matches all the items for which the **title** managed property ends with "Odyssey".

```
title:ends-with("Odyssey")
```

3.1.7 equals Operator Example

The following expression matches all the items for which the **title** managed property is "The Iliad" and for which no extra indexed tokens exist.

```
title:equals("The Iliad")
```

3.1.8 filter Operator Example

The following expression matches items that have a **title** managed property that contains "sonata" and a **doctype** managed property that contains only the token, "audio".

```
and(title:sonata, filter(doctype:equals("audio")))
```

For the preceding expression, no linguistic matching will be performed on "audio". And because the **filter** keyword will be used to match "audio", that text will not be highlighted in the dynamic teaser.

3.1.9 near Operator Examples

The following expression matches strings that contain both "cat" and "dog" as long as no more than four indexed tokens (which is the default number) separate them.

```
near(cat, dog)
```

The following expression matches strings that contain "cat", "dog", "fox", and "wolf" as long as no more than four indexed tokens separate them.

```
near(cat, dog, fox, wolf)
```

The following table contains examples of managed property string values and states whether they match the preceding expression.

Match?	Text
Yes	The picture shows a cat, a dog, a fox, and a wolf.
Yes (with **stemming**)	Dogs, foxes, and wolves are canines, but cats are felines.
No	The picture shows a cat with a dog, a fox, and a wolf.

The following expression matches all the strings in the preceding table.

```
near(cat, dog, fox, wolf, N=5)
```

If multiple operands of the **near** operator match the same indexed token, they are considered near each other. For example, the following expression matches a managed property that contains only the indexed token "clarinet" because both "c*t" and "clarinet" match and are considered near each other, even though both search tokens match the same indexed token. The search token "c*t" is evaluated through wildcards as specified in section 2.1.16.6.

```
near("c*t", "clarinet")
```

3.1.10 not Operator Example

The following expression matches items that do not contain "aardvark".

```
not(aardvark)
```

3.1.11 onear Operator Examples

The following expression matches every occurrence of the word "cat" that appears before the word "dog", as long as no more than four indexed tokens separate them.

```
onear(cat, dog)
```

The following expression matches all the occurrences of the words "cat", "dog", "fox", and "wolf" that appear in order, as long as no more than four indexed tokens separate them.

```
onear(cat, dog, fox, wolf)
```

The following table contains examples of managed property string values and states whether they match the preceding expression.

Match?	Text
Yes	The picture shows a cat, a dog, a fox, and a wolf.
No	Dogs, foxes, and wolves are canines, but cats are felines.
No	The picture shows a cat with a dog, a fox, and a wolf.

The following expression matches (with stemming) the text in the second row of the preceding table.

```
onear(dog, fox, wolf, cat, N=5)
```

The following expression matches the text in the first and third rows of the preceding table.

```
onear(cat, dog, fox, wolf, N=5)
```

3.1.12 or Operator Example

The following expression matches all the items for which the default index contains either "cat" or "dog".

```
or(cat, dog)
```

If an item’s default index contains both "cat" and "dog", it will match and have a higher dynamic rank than it would if it contained only one of the tokens.

3.1.13 rank Operator Examples

The following expression matches items for which the default index contains "dog". The expression will increase an item’s dynamic rank if its default index also contains "cat".

```
rank(dog, cat)
```

For the preceding expression, note that if an item’s default index contains "cat" but not "dog", the item will not match the expression.

The following expression matches items for which the default index contains "dog". The expression will increase an item’s dynamic rank if its default index also contains "boxer" or "pointer".

```
rank(dog, boxer, pointer)
```

The following expression matches the same items as the preceding expression but will increase an item’s dynamic rank if its default index also contains the phrase "thoroughbred beagle".

```
rank(dog, "thoroughbred beagle")
```

The following expression matches items for which the **title** managed property contains both "dog" and "beagle". The expression increases the dynamic rank of items for which the **title** managed property also contains the indexed token "thoroughbred".

```
and(title:dog, rank(title:beagle, title:thoroughbred))
```

3.1.14 starts-with Operator Example

The following expression matches items for which the **title** managed property begins with "Yet another".

```
title:starts-with("Yet another")
```

3.1.15 xrank Operator Examples

The following expression matches items for which the default index contains either "cat" or "dog". The expression boosts the dynamic rank of those items that also contain "thoroughbred".

```
xrank(or(cat, dog), thoroughbred)
```

The following expression matches items for which the default index contains either "cat" or "dog". The expression boosts the dynamic rank of those items that contain "thoroughbred" by 500 rather than by the default of 100, even for items that previously had no rank.

```
xrank(or(cat, dog), thoroughbred, boost=500, boostall=yes)
```

3.1.16 Token Operator Examples

3.1.16.1 datetime Token Operator Examples

Each of the following expressions consists of an implicit **datetime** token.

```
2008-01-29
2008-01-29T03:37:19
2008-01-29T03:37:19Z
```

Each of the following expressions consists of an explicit **datetime** token.

```
datetime(2008-01-29)
datetime("2008-01-29T03:37:19")
datetime(2008-01-29T03:37:19Z)
```

3.1.16.2 float Token Operator Examples

The following expression consists of an implicit **float** token.

```
2.71828182846
```

The following expression consists of an explicit **float** token.

```
float("3.14159265358979")
```

3.1.16.3 int Token Operator Examples

Each of the following expressions consists of an implicit **int** token.

```
360
-25
```

Each of the following expressions consists of an explicit **int** token.

```
int(360)
int(-25)
```

The following expression matches items that have an **authorid** managed property of type **integer** equal to 1, 3, 5, 7, or 9.

```
int("1 3 5 7 9", mode="or")
```

3.1.16.4 phrase Token Operator Example

The following expression matches documents that contain the phrase "to sleep, perchance to dream".

```
phrase(to, sleep, perchance, to, dream)
```

3.1.16.5 range Token Operator Examples

The following expression matches items for which the **size** managed property is greater than or equal to 0 and less than 100 (note that a value of 100 will not match).

```
size:range(0, 100)
```

The following expression matches managed properties for which the **size** managed property is greater than 0 and less than or equal 25 (note that a value of 0 will not match).

```
size:range(0, 25, from="GT", to="LE")
```

The following expression matches items for which the **size** managed property is less than 500.

```
size:range(min, 500, to="LT")
```

3.1.16.6 string Token Operator Examples

Each of the following expressions consists of an implicit string token.

```
potato
"to be or not to be"
"and"
"100"
"3.14159265358979"
"2005-12-31"
```

The following expression consists of an explicit string token.

```
string("sigh no more")
```

Because the default string mode is "PHRASE", each of the following expressions yields the same results.

```
"what light through yonder window breaks"
string("what light through yonder window breaks")
string("what light through yonder window breaks", mode="phrase")
phrase(what, light, through, yonder, window, breaks)
```

The following string token expression and **and** operator expression yield the same results.

```
string("cat dog fox", mode="and")
and(cat, dog, fox)
```

The following string token expression and **or** operator expression yield the same results.

```
string("coyote saguaro", mode="or")
or(coyote, saguaro)
```

The following string token expression and **any** operator expression yield the same results.

```
string("coyote saguaro", mode="any")
any(coyote, saguaro)
```

The following string token expression and **near** operator expression yield the same results.

```
string("coyote saguaro", mode="near")
near(coyote, saguaro)
```

The following string token expression and **near** operator expression yield the same results.

```
string("cat dog fox wolf", mode="near", N=4)
near(cat, dog, fox, wolf, N=4)
```

The following string token expression and **onear** operator expression yield the same results.

```
string("cat dog fox wolf", mode="onear")
onear(cat, dog, fox, wolf)
```

The following string token expression matches "cat", "cot", and any other three-letter indexed token that begins with "c" and ends with "t" because the "?" character is evaluated as a wildcard as specified in section 2.1.16.6.

```
string("c?t")
```

The following string token expression matches "cot", "cat", "carrot ", and any other indexed token that begins with "c" and ends with "t" because the "*" character is evaluated as a wildcard as specified in section 2.1.16.6.

```
string("c*t")
```

The following string token expression matches "c?t" without the evaluation of "?" as a wildcard character.

```
string("c?t", wildcard="off")
```

The following string token expression matches the word "nobler" with linguistic features disabled, so other forms of the word (such as "ennobling") are not matched by means of stemming.

```
string("nobler", linguistics="off")
```

The following expression matches items that contain either "cat" or "dog ", but the expression increases the dynamic rank of items that contain "dog" more than items that contain "cat".

```
or(string("cat", weight="200"), string("dog", weight="500"))
```

3.2 Keyword Examples

3.2.1 max Keyword Example

The following expression matches numeric values that are greater than or equal to 100 but less than the maximum value.

```
size:range(100, max)
```

3.2.2 min Keyword Example

The following expression matches numeric values that are less than 10.

```
size:range(min, 10)
```

4 Security Considerations

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® FAST™ Search Server 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

6 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

7 Index