[MS-FSDQE]:
Distributed Query Execution Protocol Specification

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	Minor	Updated 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	1.05	Minor	Clarified the meaning of the technical content.
03/18/2011	1.05	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.05	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Distributed Query Execution Protocol. This protocol enables a protocol client to perform search queries against a protocol server that manages a search engine in a distributed environment.

1.1 Glossary

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

big-endian
Coordinated Universal Time (UTC)
little-endian

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

aggregation specification
aggregator
arity
datetime
field collapsing
field importance level
freshness boost
hit highlighting
index partition
item
query hit
rank log
recall
result set
search index
stemming
summary class
TCP/IP
term frequency

The following terms are specific to this document:

chunk: A sequence of words that are treated as a single unit by a module that checks spelling.

internal property region: A subfield of a managed property that is used in query evaluations.

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.

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://ieeexplore.ieee.org/servlet/opac?punumber=2355

[MS-FSCX] Microsoft Corporation, "Configuration (XML-RPC) Protocol Specification"

[MS-FSFIXML] Microsoft Corporation, "FIXML Data Structure"

[MS-FSIN] Microsoft Corporation, "Input Normalization Data Structure"

[MS-FSIXDS] Microsoft Corporation, "Index Data Structures"

[MS-FSSADM] Microsoft Corporation, "Search Administration and Status Protocol Specification"

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

[RFC1950] Deutsch, P., and Gailly, J-L., "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996, http://www.ietf.org/rfc/rfc1950.txt

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

[RFC2279] Yergeau, F., "UTF-8, A Transformation Format of ISO10646", RFC 2279, January 1998, http://www.ietf.org/rfc/rfc2279.txt

[XML10] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Third Edition)", February 2004, http://www.w3.org/TR/REC-xml

1.2.2 Informative References

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

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

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

1.3 Protocol Overview (Synopsis)

This protocol issues search requests against a search server and retrieves information about **items** that match the requested search criteria. This is accomplished by sending a set of messages from a protocol client to a protocol server, where each message represents one stage of a complete operation. The stages are search and retrieval, which again are split into separate request and response messages.

The search stage consists of a message from a protocol client to a protocol server, where the message contains the query to perform. The protocol server then evaluates this query against its search index. The query can contain additional criteria on how the results are to be sorted, how many **query hits** to return and details about how to merge query hits that have the same values for a given property in the results, for example, to show only one item per category specified in a property. This protocol supports searching with both strings and numeric values, combined with Boolean operators such as "AND" and "OR". The query language also supports range searches, wildcards, and searching on proximity.

When the search stage finishes, the protocol server sends a sorted list of query hits to the protocol client. This list consists of item identifiers and the rank score for each item. The query result also contains metadata used to sort or merge the query hits. The protocol client can use this metadata to merge results from multiple protocol servers into one list of items. This merging feature enables

the protocol client to distribute searches over multiple protocol servers, either for increasing the number of simultaneous searches supported, or to divide the search index over multiple protocol servers to support larger search indexes.

Then the retrieval stage begins, where the protocol client selects specific items from the query hits, and requests the associated item summaries. The item summary contains the content and meta information about the document. The protocol server can highlight properties of the item summary, based on the query terms, to create condensed item content that shows where the search terms were found in an item. The protocol server then sends the query hit details or the condensed item content to the protocol client. Displaying the condensed item content enables the end user to determine whether the query hit is an interesting result.

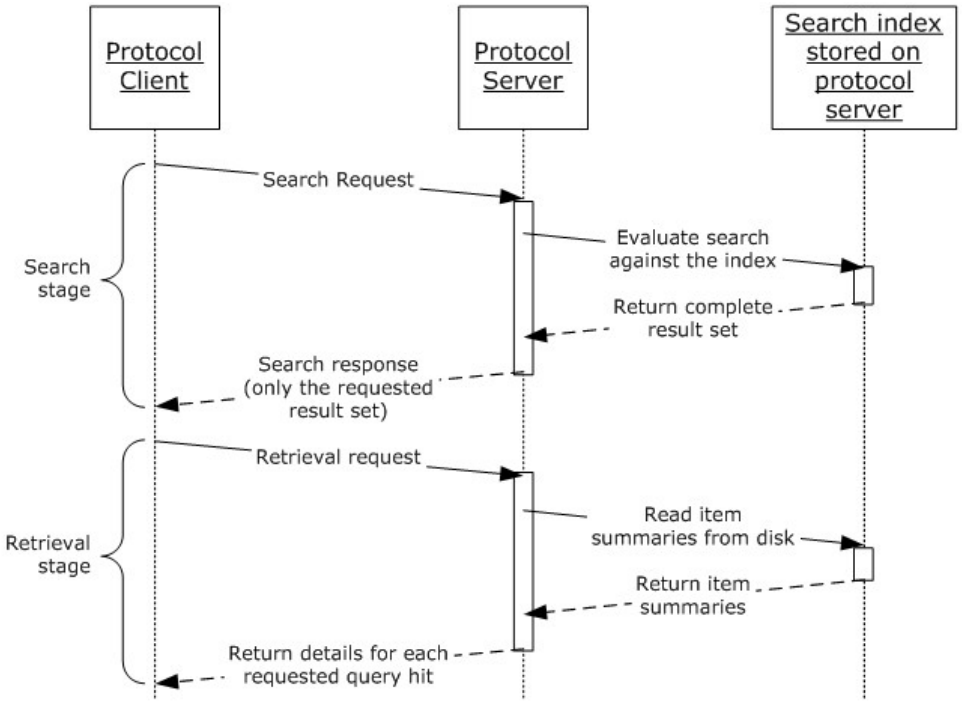


Figure 1: Overview of DQE protocol stages

The separation of the search and retrieval stages enables the protocol client to request details only for the exact items it is interested in. This reduces the resources needed for receiving and parsing in the protocol client. This also reduces the resource usage on the protocol server by avoiding unnecessary disk reads to retrieve all the details about every item that matches the query, and lowers the amount of resources used to send the result to the protocol client.

This protocol includes support for returning aggregated data for the full **result set** of a search, dynamically calculated based on the query terms, without having to return the complete result set to the protocol client. This is done by performing the aggregation as part of the search stage, and independent of the retrieval stage, and a protocol client can use this information to further refine a search before requesting the item summaries. An example of aggregate data could be the language in which the item is written, which enables the protocol client to request only items that are written in the specified language if multiple languages are present. Another example is using the length of documents to request only items of a specific size. Sorting is performed at the same early stage to make sure the order of the query hits is correct, even when a search generates more query hits than the protocol server supports returning or more than the protocol client requests.

The protocol relies on **TCP/IP** ([RFC793]) as its transport protocol for passing the messages between the client and the server. To avoid having to constantly open and close connections between the client and the server, the protocol supports multiplexing multiple messages over the same single connection. This is accomplished using virtual channels on the same connection between the server and the client. Each of the messages that are part of the search traffic are marked with a channel identifier that enables a protocol client to determine which message is the response to which request when performing multiple searches in parallel.

1.4 Relationship to Other Protocols

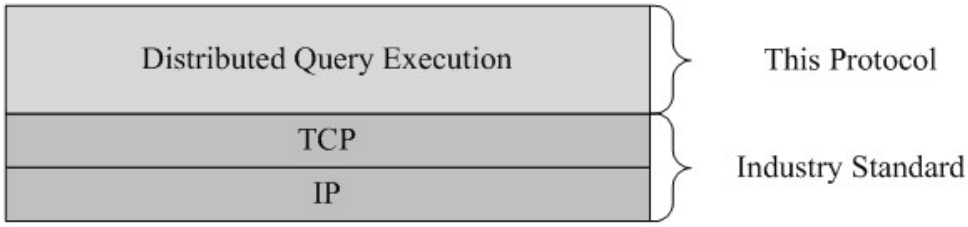


Figure 2: This protocol in relation to other protocols

No other protocol depends directly on this protocol. IPsec can be used at the network layer to add security to implementations that use the protocol, but because IPsec operates on the lower network level, it does not impact this protocol.

Protocol initialization relies on configuration files requested as described in [MS-FSCX].

1.5 Prerequisites/Preconditions

A TCP/IP connection from the protocol client to at least one instance of the protocol server needs to exist before the protocol can be used. The protocol depends on pre-shared configuration information to decode data correctly. For more information, see section 3.

The query strings are character normalized before use, as described in [MS-FSIN] section 2.1. For a complete **recall(2)** when querying for lemmatized words, an implementation uses **stemming**. If using a search index type different from the one described in [MS-FSIXDS], the index needs to be compatible when it comes to how tokenization is done. There are also rules that describe how to construct parts of the index content, including the anchor text and the associated query context catalogs as described in [MS-FSFIXML] section 2.5.2. It is a prerequisite that the query strings also follow these rules when querying these specific catalogs in the index.

For substring searches, the query string operators are tokenized as described in the file named "maptransform.xml" in [MS-FSSCFG] section 2.3.

If IPsec is used to secure connections between a protocol client and a protocol server, then authentication is performed by the underlying transport protocol.

1.6 Applicability Statement

The protocol is applicable for full-text search applications.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

All messages are transferred over TCP/IP. The protocol maintains at least one TCP/IP connection between the protocol client and protocol server for all traffic. Multiple connections can be used, but a protocol server MUST send the response on the same connection on which it received the associated request.

2.2 Message Syntax

2.2.1 Numeric Data Format Conventions

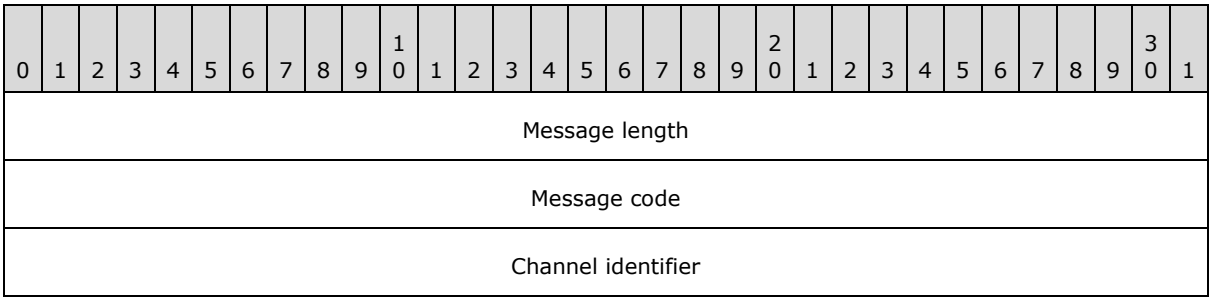
The following table specifies the data types used in this protocol. A protocol client or protocol server MUST support all of the data types in the table.

Data type	Format on the wire
float32_b	Single-precision 32-bit floating point, as specified in [IEEE754]. **Big-endian** data representation.
float32_l	Single-precision 32-bit floating point, as specified in [IEEE754]. **Little-endian** data representation.
double64_b	Double-precision 64-bit floating point, as specified in [IEEE754]. Big-endian data representation.
double64_l	Double-precision 64-bit floating point, as specified in [IEEE754]. Little-endian data representation.
int32_b	32-bit signed integer data type with two-complement signed number representation. Big-endian data representation.
int32_l	32-bit signed integer data type with two-complement signed number representation. Little-endian data representation.
uint32_b	32-bit unsigned integer data type. Big-endian data representation.
uint32_l	32-bit unsigned integer data type. Little-endian data representation.
int16_b	16-bit signed integer data type with two-complement signed number representation. Big-endian data representation.
int16_l	16-bit signed integer data type with two-complement signed number representation. Little-endian data representation.
uint16_b	16-bit unsigned integer data type. Big-endian data representation.
uint16_l	16-bit unsigned integer data type. Little-endian data representation.
int8	8-bit signed integer data type with two-complement signed number representation.
uint8	8-bit unsigned integer data type.
int64_b	64-bit signed integer data type with two-complement signed number representation. Big-endian data representation.
int64_l	64-bit signed integer data type with two-complement signed number representation.

Data type	Format on the wire
	Little-endian data representation.
uint64_b	64-bit unsigned integer data type. Big-endian data representation.
uint64_l	64-bit unsigned integer data type. Little-endian data representation.
int96_l	96-bit signed integer data type with two-complement signed number representation. Little-endian data representation.
uint96_l	96-bit unsigned integer data type. Big-endian data representation.
int160_l	160-bit signed integer data type with two-complement signed number representation. Little-endian data representation.
uint160_l	160-bit unsigned integer data type. Big-endian data representation.
datetime64_b	64 bit unsigned integer data type that contains a **Datetime** value. Big-endian data representation. Each step corresponds to 100 nanoseconds. The integer represents the time from -29000-01-01T00:00:00,000 to 29000-12-31T23:59:59,999 The value is specified in **UTC**.

2.2.2 Multi-part Message End

This message signals the end of a multi-part sequence of result details responses. The message format is specified in the following table.



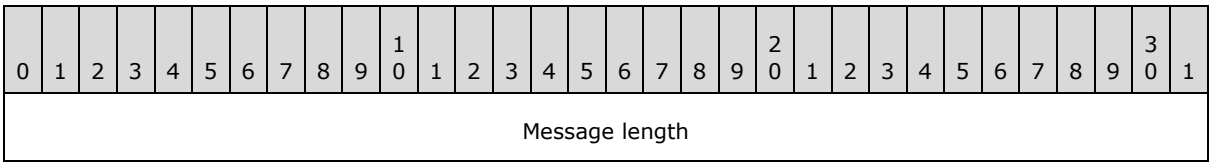
Message Length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. It contains the value 8.

Message Code (4 bytes): This is of type **uint32_b** and contains the value 200.

Channel Identifier (4 bytes): This is of type **uint32_b** and contains the identifier that associates the request and the response (section 3.2.1).

2.2.3 PING Request Message

The protocol client sends this message to determine the status of the protocol server and receive information about search service availability. The message format is specified in the following table.



Message code

Message length (4 bytes): This is of type **uint32_b** and contains the length of the message in bytes, excluding the length of this field. It contains the value 4.

Message Code (4 bytes): This is of type **uint32_b** and contains the value 206.

2.2.4 PING Request Answer Message

This is the response to a PING request. The message format is specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Message length																															
Message code																															
Index column identifier																															
Timestamp																															
Total number of search processes																															
Active number of search processes																															
Total partitions																															
Active partitions																															

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. It contains the value 32.

Message Code (4 bytes): This field is of type **uint32_b** and contains the value 210.

Index Column Identifier (4 bytes): This is of type **uint32_b** and specifies the column that the protocol server represents. It is configured at install time and MUST be greater than or equal to 0.

Timestamp (4 bytes): This is of type **uint32_b** and specifies the time that the protocol server was initialized. The age of the protocol server can assist in system diagnosis and tuning. The field contains the number of seconds that elapsed after 1970-01-01 UTC at the time it was initialized. This information is also used in the **datestamp** field when the protocol client sends a result details request (section 2.2.8).

Total Number of Search Processes (4 bytes): This is of type **uint32_b** and specifies how many search processes are running on the protocol server that sent this message. The protocol server can use this field and the **active number of search processes** field to specify monitoring information for the search service currently offered by the protocol server. The protocol client can ignore this diagnostic information.

Active Number of Search Processes (4 bytes): This is of type **uint32_b** and specifies how many search processes are currently active on the protocol server which sent this message.

The protocol server can use this field, in combination with the **total number of search processes** field, to specify monitoring information for the search service on the protocol server. The protocol client can ignore this diagnostic information.

Total Partitions (4 bytes): This is of type **uint32_b** and specifies the number of **index partitions** that need to be available on this protocol server for the complete index to be searchable.

Active Partitions (4 bytes): This is of type **uint32_b** and specifies the number of partitions available in the protocol server when this message was generated. A value less than the contents of the **total partitions** field SHOULD specify that the protocol server can only send partial query results, as specified in section 2.2.6. The protocol server uses this information only for diagnosis and tuning. It can be queried even if some of the partitions are down.

2.2.5 Error Message

This specifies the error that occurred. The channel identifier in the message is the channel identifier from the failed request. The structure of the error message is specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Message length																															
Message code																															
Channel identifier																															
Error code																															
Error message length																															
Error message																															

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. The length MUST be less than 1,000,008 bytes.

Message Code (4 bytes): This is of type **uint32_b** and contains the value 203.

Channel Identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response (section 3.2.1).

Error Code (4 bytes): This is of type **uint32_b** and specifies the error code that represents an error message. Error codes are specified in the **Error Message** field.

Error message length (4 bytes): This is of type **uint32_b** and specifies the number of bytes in the error message.

Error Message (variable): This is a UTF-8 encoded, variable-length string that specifies the error. The length of the message is contained in the **error message length** field.

The following table specifies error messages and associated error codes. The "Root cause origin" column specifies whether the error occurred on the protocol server or on the protocol client. The "Permanent / Temporary" column also specifies whether the error is temporary and can be corrected by resubmission.

Error Code	Error message	Further explanation	Permanent/ Temporary	Root cause origin
1	General error <varying extra details>.	Generic error message class for unexpected errors.	Can be both temporary and permanent, but mostly used for temporary errors.	Protocol server.
2	Message varies based on the type of parse error. Defaults to "Error parsing query" if the subcomponent associated with the error is not more specific.	Cannot parse query, syntax error. The input query from the protocol client was malformed or not supported.	Permanent.	Protocol client.
3	All partitions are down.	All partitions on a protocol server are down.	Permanent or Temporary.	Protocol server. Verify the state of the system.
6	The requested functionality is not implemented.	The requested functionality is not implemented.	Permanent.	Protocol client.
7	Query not permitted to run	Resource limitations do not permit the query to run.	Temporary	Protocol server. Resubmit query.
8	Lost connection to sub-node.	A protocol server lost connection to its underlying search node or process.	Temporary	Protocol server. Resubmit query.
9	Multiple errors have occurred.	Multiple errors occurred. A newline character separates the error messages. Error codes other than the first are prefixed to the error message in UTF-8 format.	Unknown or error-dependent.	Protocol server.
10	Query could not be evaluated.	Implementation-specific, protocol server error.	Unknown.	Protocol server.
11	Query timeout.	A timeout occurred on the protocol server.	Temporary.	Protocol server. Resubmit query.
12	Resource limit exceeded.	Not enough resources to perform the query.	Temporary.	Protocol server or protocol client. This is a temporary resource shortage on the protocol server, or

Error Code	Error message	Further explanation	Permanent/ Temporary	Root cause origin
				because of a resource intensive query from the protocol client.
13	Resource limit temporarily exceeded.	Not enough protocol server resources to perform the query.	Temporary.	Protocol server. Query can be resubmitted. For repeated failures, perform query analysis to reduce the complexity.
14	The requested functionality is not supported.	Functionality not supported.	Permanent.	Protocol client Change query syntax to functionality that is supported.
16	Requested generation no longer available for query.	The revision of the **search index** that is being queried is no longer available or cannot be reached.	Permanent.	Protocol client could have old values for index generation fields, as specified in section 2.2.6. Resubmit the query to search against search indexes when available.
17	Wildcard term count threshold exceeded.	Expansion of wildcard term exceeded the configured maximum limit of terms for expansion.	Permanent.	Protocol client Narrow the query and resubmit.
18	No engine available for partition <number>.	Specifies that a protocol server component was stopped.	Permanent.	Protocol server Some search indexes are unavailable.
20	Document summary timestamp does not match request.	Specifies that the **datestamp** field in the result details request was not equal to the startup **timestamp** field in the PING response.	Permanent.	Protocol client. The protocol client MUST use the **timestamp** field of the PING response.
21	Document summary could not be extracted.	Item summary error. Could also be related to connectivity issue or search nodes out of operation.	Permanent or Temporary.	Protocol server. Verify the installation.
22	Document summary timeout.	Item summary error. Could also be related to connectivity issue or search nodes out of operation.	Permanent or Temporary.	Protocol server. Verify the installation.

The protocol server can override the content of the error message. The error message column specifies the default error message text used. A protocol client implementation MUST base error processing on the error code rather than the error message.

2.2.6 Query Request

This message contains the query to perform on the search nodes. The structure is specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Message length																															
Message code																															
Channel Identifier																															
Enabled features																															
Query type																															
Offset																															
Max hits																															
Query flags																															
Generation Specification … …																															
Rank Profile Specification …																															
Random Seed (optional)																															
Current Date and Time (optional) …																															
User Cache Lines (optional)																															
Max Offset (optional)																															
Field Collapsing (optional)																															

Sort Specification length (optional)
Sort Specification (optional)
Aggregation Specification (optional)
Collapse Field Specification Length (optional)
Collapse Field Specification (optional)
Parsed Query (optional)

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. The length MUST be less than 60,000,008 bytes.

Message Code (4 bytes): This is of type **uint32_b** and contains the value 218.

Channel Identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response (section 3.2.1).

Enabled Features (4 bytes): This is of type **uint32_b** and specifies the request features that the protocol client enabled. The bit flags that are associated with each enabled feature MUST have a corresponding data field in the request. For example, if the **rank profile** bit flag is set, then the **Rank Profile Specification** field MUST be present in the message.

Bit patterns are specified in the following table. If any other combination of bits is specified, then the behavior is undefined. Whether the protocol server sends an error message is implementation-dependent. If a feature is not enabled in the bitmask, then there is no corresponding field data in the message. See the sections for each feature for the corresponding field data formats.

Value	Feature name	Meaning
0x00000002	Parsed Query	MUST be set for a request if there is a query. For more information about parsed query payload data, see section 2.2.6.
0x00000004	Rank profile specification	Specifies that the query is to be ranked with a specific rank profile.
0x00000080	Sort specification present	Specifies whether a sort specification is included in the message.
0x00000100	Aggregation specification	Specifies that the query contains an **aggregation specification**.
0x00000200	Random seed	Specifies that the query includes a random seed.
0x00000400	Current date and time	Specifies that the Current Date and Time field is present in the message. This MUST be set if **freshness boost** bit flag is set in the query flags field.
0x00000800	Generation Specification	MUST be set for a request. Specifies that processing of revisions of search indexes is enabled.
0x00002000	Field collapsing	Specifies that **field collapsing** is enabled and the corresponding field

Value	Feature name	Meaning
		is present in the payload.
0x00004000	Collapse field specification	Specifies that the message contains a collapse field specification.
0x00010000	User cache lines	MUST be set if the corresponding field is present in the message. Specifies that the protocol server SHOULD allocate a specific number of cache lines for the result of the current query to increase the speed of re-queries.
0x00020000	Max offset	MUST be set if a corresponding field is present in the message. This specifies the maximum offset for caching when specifying user cache lines. Normally set to the maximum offset limit of the protocol server.

The features are specified in the following fields. The order of the information in the payload is not the same as in the **Enabled features** table.

Query Type (4 bytes): This is of type **uint32_b** and specifies whether the original query from the user defaulted to match with an **OR** or an **AND** between query terms. This field does not impact how query evaluation is performed by the protocol server, because that is already taken into account when generating the **parsed query** stack. This SHOULD be set to 0 if it is unknown whether the default combination of terms was performed with an **OR** or an **AND** operator. It is also set to 0 if multiple search terms were by default combined with an **AND** operator. Otherwise, if this is set to 1, and multiple search terms were specified, they were evaluated as an **OR** expression.

Offset (4 bytes): This is of type **uint32_b** and specifies the offset in the result set from which the protocol server MUST returns query hits. If the offset is greater than the number of documents in the result set, or if the offset is greater than the maximum supported by the protocol server, then the protocol server MUST return 0 hits.

Max Hits (4 bytes): This is of type **uint32_b** and specifies the maximum number of query hits to return to the protocol client. The number of returned query hits is relative to the search result offset, as specified in the **Offset** field.

Query Flags (4 bytes): This is of type **uint32_b** and specifies bits that represent features to enable or disable. All other bits are set to 0. Values are specified in the following table.

Value	Feature name	Meaning
0x00000004	Enable error message	Specifies that the protocol server SHOULD send error messages.
0x00000008	Report queue length	The protocol server MUST sends a queue length message (section 2.2.10) before the query response.
0x00000100	Include ranking information	Specifies that the protocol server sends a rank log in the response. The rank log data is formatted as text in the predefined **ranklog** internal property field. The interpretation of the rank log data implementation specific.
0x00000800	Random rank	Specifies that random ranking is enabled in the query.
0x00002000	Freshness boost	When set, the protocol server calculates a freshness boost value. Implementations that do not use the freshness boost as part of their rank evaluation SHOULD ignore this.

Value	Feature name	Meaning
0x00008000	Report search coverage	Specifies that the protocol server sends information about search coverage, and partial result status, in the return query.
0x00020000	Allow partial results	If this is set, a protocol server sends a response, even if response content is incomplete because of lack of search index availability or other reasons. If the Report search coverage flag is set, then the protocol server sends partial results in the response. For more information, see the section 2.2.7. If this is not set, the protocol server does not send a query response if the response is incomplete. If the Enable Error message bit flag is set, the protocol server SHOULD send an error message.
0x00040000	Field collapsing	Enables field collapsing.
0x00080000	Top Level search	MUST be set by the protocol client.

GenerationSpecification 1 (12 bytes): This field consists of three numbers of the type **uint32_b**. The first MUST be set to 8. The second number MUST be set to 1, and the third MUST be set to 0.

Rank Profile Specification Field 1 (8 bytes): This optional field consists of two numbers of type **uint32_b,** where the first specifies which rank profile to use. If the protocol server does not support rank profiles, this value SHOULD be ignored. If the specified rank profile does not exist, the protocol uses the default. A rank profile configures the relevancy computation to apply to an index for a specific query. A protocol server implements rank profiles, where a configuration is mapped to a specific numeric value. The file named "rank.cf" contains rank profiles, as specified in [MS-FSSCFG] section 2.18. This file is retrieved using the protocol specified in [MS-FSCX]. The second number is not in use and MUST be set to 0.

Random Seed (4 bytes): This optional field is of type **uint32_b** and it specifies the seed for the **random** function.

Current Date and Time (8 bytes): This optional field is of type **datetime64_b**, and specifies the current date and time for the rank freshness boost. The protocol client SHOULD send the same value to all protocol servers, so that the protocol servers operate with the same value for current time for the same query.

User Cache Lines (4 bytes): This optional field is of type **uint32_b**, and specifies how many cache lines SHOULD be used to cache the results of the query.

Max Offset (4 bytes): This is an optional field of type **uint32_b** that specifies the maximum offset to cache when using user cache lines. A protocol server uses this field's value for optimizing searches. For an implementation where the protocol server specifies maximum limits for query offsets, this is set to less than or equal to the maximum limit for user specified cache lines. The default maximum offset is 100000.

Field Collapsing (4 bytes): This optional field is of type **uint32_b**, and enables the protocol server to combine results that contain identical values for a specific managed property. This field specifies the maximum number of collapsed results to keep in the result per managed property. If the number of collapsed entries for the response exceeds this maximum, the protocol server removes them.

Sort Specification Length (4 bytes): This is optional field is of type **uint32_b** and specifies the length of the **sort specification** field.

Sort Specification (variable): This is an optional, UTF-8 encoded, variable-length string that contains the sort specification. The sort specification syntax is specified in the following table.

Value	Description
+ or -	Specifies sort direction for this level. Specified as prefix to the managed property or rank profile name. "+" ascending order " –"descending order.
<sortby field>	This is one level or more of the following: ▪ <managed property>: A managed property for which sorting is enabled. This specifies to sort the results according to this managed property's value. ▪ [rank]: This specifies that the protocol server calculates this sort level according to the current rank profile. A rank profile is applied as only one level in the sort specification. Multiple rank profiles are not supported for one query. If specified, sorting by rank MUST be the last entry in the sort specification. ▪ [docid]: Sort the items by the docid field.
Random specification (full random search)	Full randomization runs on the complete result set, and is specified on the form "[random:seed=<seed>:hashfield=<managed property with sorting enabled>:addtorankmax=<max random value>]". The following parameters can be specified: ▪ seed: Seed value for the randomization. Re-use of a seed produces identical results (if the search index content is the same). MUST be present. ▪ hashfield: Optional managed property name with sorting is enabled. If specified, then it is the basis for the random values to select. ▪ addtorankmax: Optional number between 0 and this value is added to the item rank to increase randomness of results.
Sort formula	A formula to sort the result set, which is dynamically calculated for the current query. It is specified as "[formula:<expression>]". See the following table for a description of the available operators.

Square brackets ([...]) are a part of the sort syntax. They do not specify optional parameters.

Sort Formula Specification

The sort formula feature is an extension of the single- and multi-level sorting functionality. The sort formula accepts as input one or more numeric managed properties. The sort formula is specified as "[formula:<expression>]", and is dynamically evaluated on the result set of the search query. Supported operators are specified in the following table.

Operator	Description
+	Addition.
-	Subtraction.

Operator	Description
*	Multiplication.
/	Division.
rank	Rank for the item. For example, abs(rank-100) sorts on the distance from rank value 100.
[0-9.]+	Numbers are integer or double precision floating point numbers.
[a-z0-9]+	Any character strings that are not parsed as functions are processed as managed property names. Managed properties for which sorting is enabled are valid, for example, if sort is enabled for the managed property named "height", then the property can be used as an item-specific value in the sort formula.
( )	Used to group calculations ensuring correct precedence, for example, 4*(3+2)
sqrt(x)	The square root. For example, sqrt(9) returns 3.
pow(x,y)	Calculate x to the power of y. For example, pow(2,3) returns 8.
exp(x)	Calculates the natural logarithm exponent function.
log(x)	Natural logarithm function.
abs(x)	Absolute value. For example, abs(4-10) returns 6.
ceil(x)	Ceiling function. Rounds up to the closest integer. For example, ceil(4.5) returns 5.
floor(x)	Floor function. Rounds down to the closest integer. For example, floor(4.5) returns 4.
round(x)	Round to even function. Rounds to the closest integer. For example, round(4.5) returns 5, but round(4.4) returns 4.
bucket(x,a,b,c,…)	The first parameter MUST be the values to sort by, either a managed property name or a sub-expression. The remaining parameters specify the numeric bucket thresholds. The number of bucket thresholds is arbitrary. For example, if the parameter contains "bucket(size,5,15,50,100)", then the values associated with the managed property named "size" are aggregated into buckets, and the values are rounded down to the closest threshold. Values lower than the value of the first parameter are rounded to zero. Values higher than the highest parameter are rounded to value of the last parameter.

All operations support managed property names and floating point or integer numeric values. In addition, the special managed property named "rank" MUST be available, which represents the rank for an item.

Example 1: Sorting the items by the square root of the rank is specified with the sort formula "[formula:sqrt(rank)]".

Example 2: If there is a managed property in the search index named "height" that contains the height of a building in meters, then a formula that sorts the results by which buildings are closest to a height of 20 meters, and that uses the abs() operator, is written as "[formula:abs(20-height)]".

Aggregation Specification (variable): This is an optional UTF-8 encoded string field that specifies the type of aggregation data that is requested. The protocol server dynamically compiles aggregate statistics on the data in the query result set. These statistics can contain histogram data whose

aggregate results are bucketed based on managed property values, in addition to minimum, maximum, and mean values for numeric managed properties. This field contains the following fields:

Uint32_b field: The length of the UTF-8 encoded string that occurs immediately after this field.

UTF-8 encoded string: A variable sized **chunk** that contains an aggregation request.

The aggregation request MUST be according to the following ABNF grammar for all requests other than refines:

```
aggregationrequest = "(" ( singlevaluefunc / noargumentfunc / hist ) ")"
singlevaluefunc = ( "max" / "min" / "sum" / "count" / "countnz" SP ":" ) top SP
internalmanagedpropertyname
noargumentfunc = "hitcount" SP
hist = "hist" SP histtype SP *keyvalue SP internalmanagedpropertyname
histtype = ":" ( specifiedbuckets / "buckets :unique" / fixedbuckets / fixedwidth )
specifiedbuckets = ":buckets" SP "'(" 1*DIGIT / bucketspecification ")"
bucketspecification = (1*DIGIT SP 1*DIGIT) / 1*DIGIT
fixedbuckets = ":buckets" SP 1*DIGIT
fixedwidth = ":width" SP 1*DIGIT
keyvalue = ":" ( top / sorder / cutfreq / cutminbuckets / cutmaxbuckets / prefix )
top = "top" SP 1*DIGIT
sorder = "sorder" SP ( "lexasc" / "lexdesc" )
cutfreq = "cutfreq" SP 1*DIGIT
cutminbuckets = "cutminbuckets" SP 1*DIGIT
cutmaxbuckets = "cutmaxbuckets" SP 1*DIGIT
prefix = "prefix" SP 1*UTF-8-octets
internalmanagedpropertyname = 1*ALPHA
```

UTF-8-octets are characters that are present in the index after tokenization and normalization, as specified in [MS-FSIN] section 2.

All references to aggregated managed properties specify the attribute vector names by which the managed properties are known to the protocol server. The file named "index.cf" associates the managed property names with the internal attribute vector names, as specified in [MS-FSSCFG] section 2.10. This file can be retrieved with the protocol specified in [MS-FSCX]. For more information about the internal naming of fields and attribute vectors, see [MS-FSSCFG] section 2.1.2.

The **refine** aggregation request differs in that it uses the prefixed name of the managed property as the first parameter. For example, "(refine bavnstring1 2 3'w01 3'w02)" is a refinement request for the managed property named bavnstring1 to recalculate the occurrence count on the 2 buckets "w01" and "w02". For more information, see the **Refine Function** section.

The protocol server supports both numeric- and string aggregation requests. Numeric managed properties use numeric **aggregators**, while string managed properties use string aggregators.

Aggregation requests are case-sensitive and contain calls to at least one aggregation function. The main function call is the **hist** function, which generates the histogram of aggregation buckets in the query response. The contents of the aggregation buckets correspond to the entries in the managed properties in the query results. For example, if a collection of items contains an managed property, which supports aggregation, that is associated with size, then the result of a **hist** function performed on this property is a series of aggregation buckets that contains sizes. The exact distribution of aggregation buckets is specified in the parameters sent to the **hist** function. The distribution is one aggregation bucket per value, or it is aggregation buckets that contain multiple

size values from different result hits. The histogram function supports the parameters specified in the following table.

Operator	Description
:buckets <optional bucket specification>	Histogram with variable width buckets. The bucket is specified as follows: '(<bucket limit> <bucket limit> <bucket limit>)
:buckets :unique	Histogram with one bucket per unique value
:buckets <n>	Histogram with <n> buckets of equal size.
:width <n>	Histogram with fixed width buckets. N is the width of the buckets.

The aggregation buckets are either specified manually or they are specified by appending the unique parameter. For example, an aggregation bucket specification is specified as "hist :buckets '(5 10 15) <name or internal managed property attribute vector>", while the appended parameter version is "hist :buckets :unique". An aggregation buckets parameter with only one integer parameter is a special case of the manual bucket specification, and specifies exactly how many buckets in which to store the results. If "unique" is not present, a manual aggregation bucket size MUST be specified. The protocol server supports the aggregation functions specified in the following table.

Function	Description
Min	Minimum value for the managed property associated with the aggregator in the current result set. If the aggregator is not numeric, this is not applicable.
Max	Maximum value for the managed property associated with the aggregator in the current result set. If the aggregator is not numeric, this is not applicable.
Sum	Sum of all the values for the managed property associated with the aggregator in the current result set. If the aggregator is not numeric, this is not applicable.
Refine	Recalculate aggregation buckets for the specified query. Used by the protocol client if the query results specify that some aggregation buckets were truncated. The refine function operates on a specific aggregator element to make the protocol server recalculate the result to insure correct occurrence counts for each bucket.
Hitcount	Total number of query hits on which aggregated values are based.
Count	The number of sample values used as basis for the functions result. An item can contain multiple values for a specific aggregation.
Countnz	Returns the number of entries in the result set that contain at least one value for this aggregation result.

An example numeric aggregation specification is as follows:

```
(max bavnnumeric2)(min bavnnumeric2)(sum bavnnumeric2) (count bavnnumeric2)(countnz
bavnnumeric2)(hitcount )(hist :width 1 bavnnumeric2)
```

The **hitcount** function takes no parameters, except for the standard :top parameter to limit the result set to work on.

Multiple aggregation requests are combined into one aggregation specification in the request. In this case the sequence of the aggregation data in the response MUST be identical to the sequence they were requested.

The refine function is specified in the **Refine Function** section. All other supported functions are specified in the following table.

Function	Parameters
min	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
Max	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
Sum	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
hitcount	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
count	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
countnz	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result.
hist	:top <n> - Optional parameter that specifies the number of query hits to use as basis for the result. :sorder <lexasc|lexdesc> - Optional parameter that specifies sort order as lexically ascending or lexically descending, respectively. :cutfreq <n> – Optional parameter that specifies the minimum frequency for an aggregation bucket to return. For example would n=2, return only aggregation buckets with more than 2 entries. :cutminbuckets <n> - Optional parameter that specifies the minimum number of aggregation buckets to return before frequency cutting (:cutfreq) takes effect. Use this to avoid returning too few aggregation buckets. :cutmaxbuckets <n> - Optional parameter that specifies the maximum number of aggregation buckets to return. :prefix <string> - Optional parameter that specifies to return only aggregation buckets that contain this string prefix. If the aggregator is not a string aggregator, this is not applicable. The :width, :buckets and :unique parameters are specified elsewhere in this specification.

Refine Function

Refine differs from the aggregation functions covered in the previous section in that the specification does not follow the same ABNF grammar. Refine functions are specified by the following ABNF grammar instead:

```
refinerequest = "(" "refine" internalmanagedpropertyname SP bucketcount bucketspecifications
")"
internalmanagedpropertyname = 1*ALPHA
bucketcount = 1*DIGIT
bucketspecifications = 1*bucket ;The number of buckets MUST be equal bucketcount
bucket = namelength "'" bucketname ; bucketname MUST be namelength length.
namelenght = 1*DIGIT
bucketname = 1*UTF-8-octets
```

UTF-8-octets are the characters that are present in the index after tokenization and normalization, as specified in [MS-FSIN] section 2.

Example: (refine bavnstring1 2 3'w01 3'w02)

In this example, the aggregation buckets to recalculate are w01 and w02. In this case w01 and w02 represent two values in the response for the managed property "string1". The refine method does not take optional parameters.

Collapse Field Specification Length (4 bytes): This is an optional field of type **uint32_b** that specifies the length of the **collapse field specification** field.

Collapse Field Specification (variable): This is an optional UTF-8 encoded, variable-length **string** that specifies the name of the managed property associated with the **collapse** function. Collapsing occurs only if both this field and the **Field Collapsing** bit flag are specified. Sorting MUST be enabled for the managed property on which collapsing is performed, and the property MUST be numeric or of type **datetime**.

Parsed Query (variable): This is an optional variable-length string that specifies the parsed query, which is always the last information in a request. This field contains one **uint32_b** field that specifies the approximate number of entries in the query stack, followed by one variable length chunk that contains the serialized query stack. Because the number of entries is approximate, the protocol client / protocol server MUST not depend on that the number of entries is exact to unpack the query stack.

The query stack contains operators with optional features and information about the origin of the operator. The length of the serialized parsed query stack is the length of the request minus the lengths of the fields that were already read. The length of each operator on the stack is variable. The origin information specifies which feature that added the operator to the query, such as whether the operator was added automatically by the protocol client or specified in an end user request. The operators take varying numbers of parameters, and some operators also take operands.

For example, the expression "AND (term1, term2)" is represented as one **AND** operator. A separate parameter that occurs after the **AND** expression on the serialized stack specifies the **arity** of the **AND** expression. In this example, there are two operands, term1 and term2, that are the next string term operators. The operands are themselves separate operators.

The operators are serialized to the stack in a depth first order. For the expression "AND(term1,OR(term2,term3),term4)", the order of the operators on the stack is "**AND**", "term1", "**OR**", "term2","term3", "term4". The query is shown as a tree in the following figure, and the resulting stack is shown in the figure after the following figure.

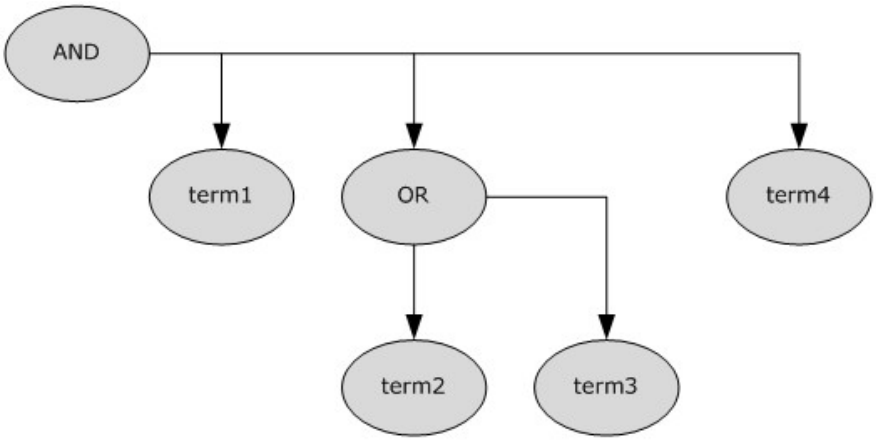


Figure 3: Query node tree

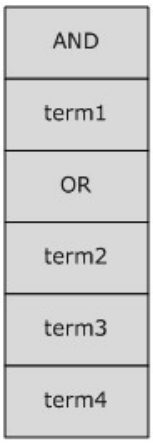


Figure 4: Serialized stack

An operator is represented as a **uint32_b** field, where the leftmost 12 bits specify the operator type. Bits 12-19 specify the origin if origin information is available. Bits 20-31 specify the operator features enabled. Operands that are associated with the operator are specified next. These operands have their own feature fields and parameters, depending on the operator type. Operand parameters are not associated with the original operator. The syntax is specified in the following table

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Operator type												Origin								Feature											
Feature fields (optional, length depending on the feature specification)																															
Operator parameters (variable, depending on operator type)																															

Operator type (12 bit): The available operator types are specified in the following table. See section 2.2.6.1 for specifications for each operator type.

Value	Operator
0	OR operator.
1	AND operator.
2	AND NOT operator.
3	RANK operator.
4	String term operator.
5	Numeric term and numeric ranges.
6	PHRASE operator.
8	Prefix wildcard. The operand string term MUST NOT contain wildcard characters, and only the prefix wildcard is supported. This SHOULD be an optimized wildcard operator, rather than a normal wildcard operator.

Value	Operator
9	Wildcard operator that MUST include wildcard tokens.
11	ANY operator. This matches any of the items that are in the query.
12	NEAR operator.
13	ORDERED NEAR operator.
14	IN operator.
15	Internal property region operator.
16	Complete internal property region operator.
17	Internal property region RANGE operator.
18	COUNT operator
19	Internal property region EQUALS operator.
20	Internal property region STARTS WITH operator.
21	Internal property region ENDS WITH operator.
22	XRANK operator.
23	EVERYTHING operator.

Operator Origin (8 bit): This SHOULD be used to specify which feature that added the operator to the query stack, or to specify that the behavior of the operator was modified. For example, the protocol client sets a bit to specify that a string term was added as a consequence of stemming. Another example is that the protocol client can use origin information to comment operators so that the protocol server can do hit highlighting differently if hit highlighting is enabled. It can also use the original origin to determine what the real query from the user was when calculating rank. If this field is not used, it is set to 0. Values for the field are specified in the following table.

Value	Feature name	Meaning
0	Original	The query operator came from the user. No modification to operator
1	Automatic filter	Automatically applied filter on operator. Operators with filtered origins SHOULD NOT influence rank.
3	Approximate match	Operator is an approximation. Related to spell-check features.
5	Lemmatization	Added because of stemming.
6	Proper name	The operator was identified as being the name of a company or other official entity.
7	Similar	Operator added as a consequence of searching for similar documents
8	Query boost	Operator impacted by some type of query boost.

Feature (12 bit): If a feature is enabled in the **feature** field of the operator, the corresponding payload occurs immediately after the **uint32_b** field that specifies the operator. The amount of data

in this payload depends on the enabled features. Not all features increase the payload that the protocol server sends to the protocol client. The features are specified in the following table.

Value	Feature name	Meaning
0x00100000	Weight value	An operator SHOULD be weighted differently than default when calculating rank. Weight is applicable only for string terms.
0x00400000	Dictionary normalization	Used to override the values for word term frequencies.
0x00800000	Exact hit	This operator is enclosed in a filter. Result rank SHOULD NOT be impacted by the operator when this is set. This feature does not add any data to the payload.
0x01000000	Return internal property region	This feature applies only to internal property region operators. It specifies that the current internal property region, and all sub-regions SHOULD be returned. This enables searching in a deeper internal property region without limiting the returned data to the deeper internal property region. This feature does not add any data to the payload.

The fields are read in the same order as they are specified in the following sections. The "Exact hit" and "Return internal property region" features are represented by flags that specify processing options rather than payload content, therefore they are not associated with separate subsections.

Weight

A **uint32_b** field is present if the **weight** field is set for an operator. The normal weight for a term is 100, and the number specified is relative to this weight. For example, if the **weight** field contains a value of 200, then the operator is twice as important in a relevancy calculation.

Dictionary Normalization

If this feature is enabled on a string term operator, local and global term frequencies are specified as parameters. This is not applicable to operators other than term operators. The structure of the payload is specified in the following table.

Parameter type	Description
uint8 stored as uint32_b	Number of **field importance levels** in search indexes for which normalized result occurrences are specified. A 32-bit space is reserved for this field in the message, but only the 8 first bits are used. The remaining 24 bits MUST be ignored. Indexes with multiple levels is supported; see [MS-FSSCFG] section 2.8.5.3.1 for details. For an implementation without levels, the term frequency count for each level can be summed up into a single count by the protocol server. This value specifies the number of **uint32_b** fields that are serialized sequentially in the next part of the payload.
uint32_b * X	The number of uint32_b variables, where X is the numeric value in the initial 8 message bytes. These variables represent the number of times the term is present in the query result set per level of the search index.
uint8 stored as uint32_b	Number of field importance levels in the search indexes for which normalized global occurrences are specified. A 32-bit space is reserved for this field in the message, but only the 8 first bits are used. The remaining 24 bits MUST be ignored. An index with multiple levels is supported as specified in [MS-FSSCFG] section 2.8.5.3.1. For an

Parameter type	Description
	implementation without levels, the normalized global occurrence count for each level can be summed up into one single count by the protocol server. This value specifies the number of fields stored sequentially in the next part of the payload.
uint32_b*X	Specifies the number of times the term is present in the whole search index. One value is specified per level of the search indexes. The number of fields is specified as the numeric value in the leftmost 8 bytes of the field.

These values SHOULD override the frequency values of the protocol server to specify consistent **term frequency** when calculating rank, so that the same word has the same frequency on different search indexes and different protocol servers.

These values MUST be normalized against an imaginary search index of 10 million items. An example is that if the frequency of a term is set to 5, and the search index contains 1 million items, the size of the search index is ignored and the normalized value for this message is 50.

For protocol server implementations that do not use term frequency as part of calculating rank, they MAY be ignored.

2.2.6.1 Query operators

The protocol server MUST support all operators specified in this section, but some operators MAY be treated as dummy operators and have no effect if it does not fit with the protocol server implementation. An example would be that a protocol server that does not implement ranking, would ignore the RANK and XRANK operators.

OR operator

This operator takes one parameter, which is the arity, or the number of operator nodes on the stack on which this operator operates. For example, an **OR** operator with an arity of three operates on 3 operands from the stack. This maps to an expression such as "or (operand1, operand2, operand3)".

The arity is represented as a **uint32_b** field. Only items that match at least one **OR** operand MUST be returned. Matching items SHOULD receive a higher rank when more **OR** operands match; however, this is implementation-dependent.

AND operator

This operator takes one parameter, a **uint32_b** field that specifies the arity of the operator. Only items matching all **AND** operands MUST be returned.

AND NOT operator

This takes one parameter, the arity of the operation. The arity is represented by a **uint32_b** field. Only items that match the first operand from the stack without matching any of the rest of the operands MUST be returned.

RANK operator

This SHOULD increases the rank on results based on whether an operation hits in a result already present in the result set. This is not an absolute requirement for the protocol because rank calculations are implementation-dependent. Its use does not impact the recall of the query. The

amount of rank increase is implementation specific. The **RANK** operator takes two parameters, which MUST be present, as specified in the following table.

For example, a query is searching for the term "hello". The query ranks results higher if they also contain the term "world". This is accomplished with a **RANK** operator that uses "hello" as its first operand, and "world" as the second. The **RANK** operation supports multiple occurrences of operands for increasing rank.

Parameter type	Description
uint32_b	Arity
uint32_b	MUST be set to 0.

String Term

This represents a query token for which to search for. A string term operator can specify only the query token, or it can also specify a specific index to search in. A **uint32_b** field that specifies the length of the index name occurs immediately after the operator. If the length is greater than 0, then a UTF-8 encoded index name that contains that number of bytes occurs after the length field. A **uint32_b** field that specifies the length of the query token in bytes occurs next followed by a query token whose length is the one that was specified. The string term token parameter is encoded as UTF-8, as specified in [RFC2279].

There is no end-of-term marker, so an implementation processes the token string by reading the number of bytes specified as length.

All string term values are lowercase. Any uppercase letters in a string term specify metadata about the string term. All string terms are suffixed by an uppercase letter that specifies the type of the term. The suffix letters are specified in the following table.

Letter	Type of term
T	Token. The String term is a non-lemmatized term
L	Lemmatized. The string term is a lemmatized form.

Lemmatized string term operators are suffixed by an "L". All string term operators that are not lemmatized are suffixed by a "T". For example, a query that does not use stemming represents the original non-lemmatized form of the string term "car" as "carT".

This enables matching only one form of a word. The protocol server stores the original term suffixed both with a "T" and an "L", so queries for which lemmatization is enabled will match the term suffixed by "L", while queries that are not lemmatized will still match only the original term. By having both in the search index, a query does not explicitly search for variations.

A protocol client normalizes query token characters, as specified in section 1.5, before serializing them to a string term operator. Failure to do so correctly could lead to reduced recall.

Numeric Term and Ranges

This represents a numeric query token on the query stack. A numeric term operator is followed on the stack by a **uint32_b** field that specifies the length of the name of the index to search. If the length is zero, the name of the index is not specified. If the length is greater than 0, then the next item on the stack is an UTF-8 encoded index name whose length is the number of bytes specified. A

uint32_b field that specifies the length of the numeric term in bytes occurs next, followed by an UTF-8 encoded string version of the numeric term of that length.

Integer values are encoded as 9223372036854775808 + the original integer value . The value 9223372036854775808 corresponds to the hexadecimal value "0x8000000000000000". Date/time values MUST be represented according to their integer value representation.

When a numeric range is used, this MUST specified by a numeric term string on the form [<integer1>;<integer2>]. The calculation MUST be done greater than or equal integer1 and less than integer2. To achieve greater than integer1 instead, add 1 to integer1 before submitting the query, while less than or equal can be implemented by adding 1 to integer2.

Internal property region searches specify integers and datetime values as subpart strings in the format "subpart|<numeric string representation>". For example, searching for the integer 30 in the subpart "number" of an internal property region, is specified as a search for "number|09223372036854775838" in the internal property region. A search for the string "30" is still processed as a normal string term in this case. There is a special **RANGE** operator specified for range searches in internal property regions, which does not use this encoding. See the Internal Property Region **RANGE** operator section for a specification of this operator.

PHRASE operator

Operands of this operator are processed as a phrase when evaluating a query. An operand is followed by a **uint32_b** field that specifies the arity of the operator. The next **uint32_b** field specifies the length of the name of the index. If this length is greater than 0, then an UTF-8 encoded index name that contains that number of bytes occurs after the length field. This index name specifies the part of the search index on which to process the phrase.

If an index name is specified for a **PHRASE** operator, this index name MUST also be specified for each operand. The operands do not inherit the index from the **PHRASE** operator. If the same index is not specified for all of the operands, the behavior is undefined.

Only string term operators are valid operands for the **PHRASE** operator. The **PHRASE** operator is not supported for region searches, and therefore is emulated by using the **ONEAR** operator with a distance of 0. A protocol server can choose to emulate PHRASE operators using AND operators if this is needed for performance reasons, sacrificing exactness for performance. For example, searching for the phrase "The big new bottle", this could internally be evaluated by the protocol server as the phrase "big new bottle" combined with an AND operator and the search term "The", if the term "The" was found to be too common for the protocol server to evaluate as part of a phrase.

Prefix Wildcard

This SHOULD be an optimized form of the standard wildcard feature, as specified in the next section. It only supports prefix-matching. For example, "pref*" matches preferences, while "*erences" is not valid. The ordinary wildcard operator MUST be used for other wildcard searches than prefix-matching. The prefix wildcard operator returns only items that match the prefix term. It takes the parameters specified in the following table. All parameters MUST be present, but the length of the byte arrays can be 0.

Parameter type	Description
Uint32_b	Length of optional index name to match against
Byte array	Index name. Length is specified by the previous parameter.

Parameter type	Description
Uint32_b	Length of term for which to perform a prefix search.
Byte array	Prefix term for which to search. Length is specified by the previous parameter, and does not include the asterisk character.

Wildcard Operator

This operator returns only the items that match the parameters of this operator. The wildcard characters that are supported are "?", which matches one character, and "*", which matches zero or more characters.

The operator supports the parameters specified in the following table. All parameters MUST be present, except for byte arrays whose length can be 0.

Parameter type	Description
uint8	Wildcard flags to enable/disable features. MUST be set to 0.
uint32_b	Minimum number of characters in expansion
uint32_b	Maximum number of characters in expansion.
uint32_b	Length of optional index name
Byte array	Index name. Length is specified by the previous parameter. MUST be UTF-8 encoded.
uint32_b	Length of wildcard term, including any wildcard tokens.
Byte array	Wildcard term. MUST be UTF-8 encoded.

ANY Operator

This takes one parameter, that specifies the arity of the operator. The arity MUST be represented by a **uint32_b** field. The **ANY** operator is similar to **OR** except that rank calculations are not affected by the number of operands that match in an item. Instead, a rank calculation uses the highest rank of the operands that match. Only items matching at least one **ANY** operand MUST be returned.

NEAR Operator

This takes two parameters. The first parameter is arity of the operator. The second parameter is the maximum number of positions (distance) between the operands. If this distance between operands is exceeded, they are not returned. Both fields are of type **uint32_b**.

For example, the **NEAR** operator with the arity of 3 and distance of 2 specifies that all three operator nodes are 2 or less positions away from each other altogether.

Another example, "operator1 string1 string2 operator2" has a distance of two from operator1 to operator2. If **NEAR** has an arity greater than 2, the maximum number of words between the terms is counted within the entire expression.

Complex operators such as **OR** and **RANK** can be operands for the **NEAR** operator. The **AND** operator, the **AND NOT** operator, and **RANGE** operators are not allowed as operands.

The **NEAR** operator returns only results where the distance is less than or equal to the specified distance.

If the **NEAR** operator is used with more than 4 operands, or if the **NEAR** operator is used with the **RANK** operator, the wildcard operator, the **OR** operator, the **ANY** operator, or the **XRANK** operator, then the **NEAR** operator MUST be preceded by an IN operator that has been associated with a complete internal property region operand. The **NEAR** operator is the second operand associated with the **IN** operator.

ORDERED NEAR Operator

This operator is similar to the **NEAR** operator, but the order of the operands is relevant. An ordered **NEAR** operation on the two string operators "string1", "string2" with distance 1 MUST match "string1 string2", but not "string2 string1". An ordered **NEAR** operator with more than 4 parameters is the same as a multiple ordered **NEAR** of 4 parameters combined with a logical **AND** operator.

The ordered **NEAR** operator returns only results where the distance is less than or equal to the specified distance, and the order is the same.

If the ordered **NEAR** operator is used with more than 4 operands, or if the ordered **NEAR** operator is used with the **RANK** operator, the wildcard operator, the **OR** operator, the **ANY** operator, or the **XRANK** operator, then the ordered **NEAR** operator MUST be preceded by an **IN** operator that has been associated with a complete internal property region operand. This **IN** operator is associated with the **NEAR** operator as its second operand.

IN Operator

This specifies that the operands that follow it apply only to the internal property region that was specified as an operand, rather than the entire content of the managed property. The **IN** operator takes one parameter, which is the arity of the operator. This is a **uint32_b** field. The first operand of an **IN** operator is an **Internal property region operator**, as specified in the next section.

The **IN** operator changes the evaluation of the operands to return only results that contain hits that are in the internal property region specified by the first operand.

Internal Property Region Operator

The internal property region is a subfield of a managed property. This operator specifies the internal property region to which the current operator applies. An **IN** operator MUST precede an internal property region. The internal property region operator supports the parameters that are specified in the following table. All parameters MUST be present, except for the index parameter if its length is 0.

Parameter type	Description
uint32_b	Length of the name of the index
Byte array	Index to search in. Length specified by the previous parameter. MUST be UTF-8 encoded.
uint32_b	Length of internal property region specification
Byte array	Internal property region specification. Length specified by the previous parameter. MUST be UTF-8 encoded.

Complete Internal Property Region Operator

All **IN** operators MUST be followed by an operand that specifies the internal property region to which it limits operands that are associated with it. This operator specifies that the active internal property region is the entire managed property. An **IN** operator that is associated with a complete internal property region searches the entire managed property. This enables operators that take internal property regions as operands to process complete managed properties.

The arity of this operator is 0, and it does not have any parameters.

Internal Property Region RANGE Operator

This operator MUST be used only for internal property region searches. The numeric term operator MUST be used for operating on complete managed properties instead.

This operator implements the **RANGE** operator for subfield queries for integer and datetime fields. When used, both maximum and minimum values MUST be specified. The range is calculated as greater than or equal to the minimum value, and less than the maximum value. The operator parameters are specified in the following table. All parameters MUST be present, except for the index parameter when the length of the name of the index is 0.

In internal property region searches, integers and datetime fields are encoded as strings of the format "subfield|<numeric string representation>". For example, the search for the integer 30 in the subfield "number" of an internal property region, is specified as "number|09223372036854775837" in that property region.

Parameter type	Description
uint32_b	Index name length
Byte array	Index name. Length specified by the previous parameter. MUST be UTF-8 encoded.
uint32_b	Length of minimum value specification.
Byte array	Minimum value. Because this is sent as a serialized byte array, The endianness is the same as the underlying computing platform.
uint32_b	Length of maximum value specification.
Byte array	Maximum value. Because this is sent as a serialized byte array, The endianness is the same as the underlying computing platform.

COUNT Operator

This matches results where an operand exists a specific number of times. The operator takes two parameters as specified in the following table. Both parameters MUST be present.

Parameter type	Description
uint32_b	Minimum number of occurrences
uint32_b	Maximum number of occurrences

The **COUNT** operator is used on an internal property region. As a consequence, the arity of the **COUNT** operator is 2, which specifies the **complete internal property region** operator, and a string term operand. This operator can be used only for complete internal property regions. It supports only operands that are single query terms, phrases or wildcards. Phrases are emulated

using ordered **NEAR** with the distance 0, because this protocol does not support phrases in property regions.

The maximum field is exclusive, so to match when a term occurs more than 5 and less than or equal 10 times, then the minimum number of occurrences is 5 and the maximum is 11.

The **COUNT** operator returns results only for string terms that are within the specified minimum and maximum occurrence range.

Internal Property Region EQUALS Operator

The internal property region **EQUALS** operator has a fixed arity of 2 and does not take any parameters. The first operand is the internal property region specification, and the second operand is the comparison value. This operator returns results only where the content of the internal property region is equal to the second operand.

The protocol server emulates an internal property region **EQUALS** operator for non-property-region searches by searching for the internal region begin/end marker combined with the query terms in the form of a phrase. For **EQUALS**, the protocol server adds the marker at both sides of the query terms. The internal region marker is the UTF-8 character "c7 82" when using a search index as specified in [MS-FSIXDS]. For example: searching an internal property region with the **EQUALS** search for the term "CompleteString" is emulated as a phrase search for "0xC70x82CompleteString0xC70x82".

Internal Property Region STARTS WITH Operator

The internal property region **STARTS WITH** has a fixed arity of 2 and does not take any parameters. The first operand specifies an internal property region, and the second operand is the comparison value. It returns results only where the beginning of the internal property region is equal to the second operand.

The protocol server emulates an internal property region **STARTS WITH** operator for non-property-region searches by searching for the internal region begin/end marker combined with the query terms in the form of a phrase. For **STARTS WITH** the marker is added at the front of the query terms. The internal region marker is the UTF-8 character "c7 82" when using a search index as specified in [MS-FSIXDS]. For example, an internal property region starts with a search for the term StartofString is emulated as a phrase search for "0xC70x82StartofString".

Internal Property Region ENDS WITH Operator

The internal property region **ENDS WITH** operator has a fixed arity of 2 and does not take any parameters. The first operand MUST be an internal property region specification. The second operand contains the comparison value. It returns only results where the end of the internal property region is equal to the second operand.

The protocol server emulates an internal property region **ENDS WITH** operator for non-region searches by searching for the internal region begin/end marker combined with the query terms in the form of a phrase. For **ENDS WITH** the marker is added at the end of the query terms. The internal region marker is the UTF-8 character "c7 82" when using a search index as specified in [MS-FSIXDS]. For example, an internal property region **ENDS WITH** search for the term String is emulated as a phrase search for "String0xc70x82".

XRANK Operator

This SHOULD increase the rank on results based on whether an operator hits in a result already present in the result set. It MUST NOT impact the recall of the query. The **XRANK** operator differs

from the normal **RANK** operator, in that the protocol client specifies how much to increase/decrease the rank. The operator parameters are specified in the following table.

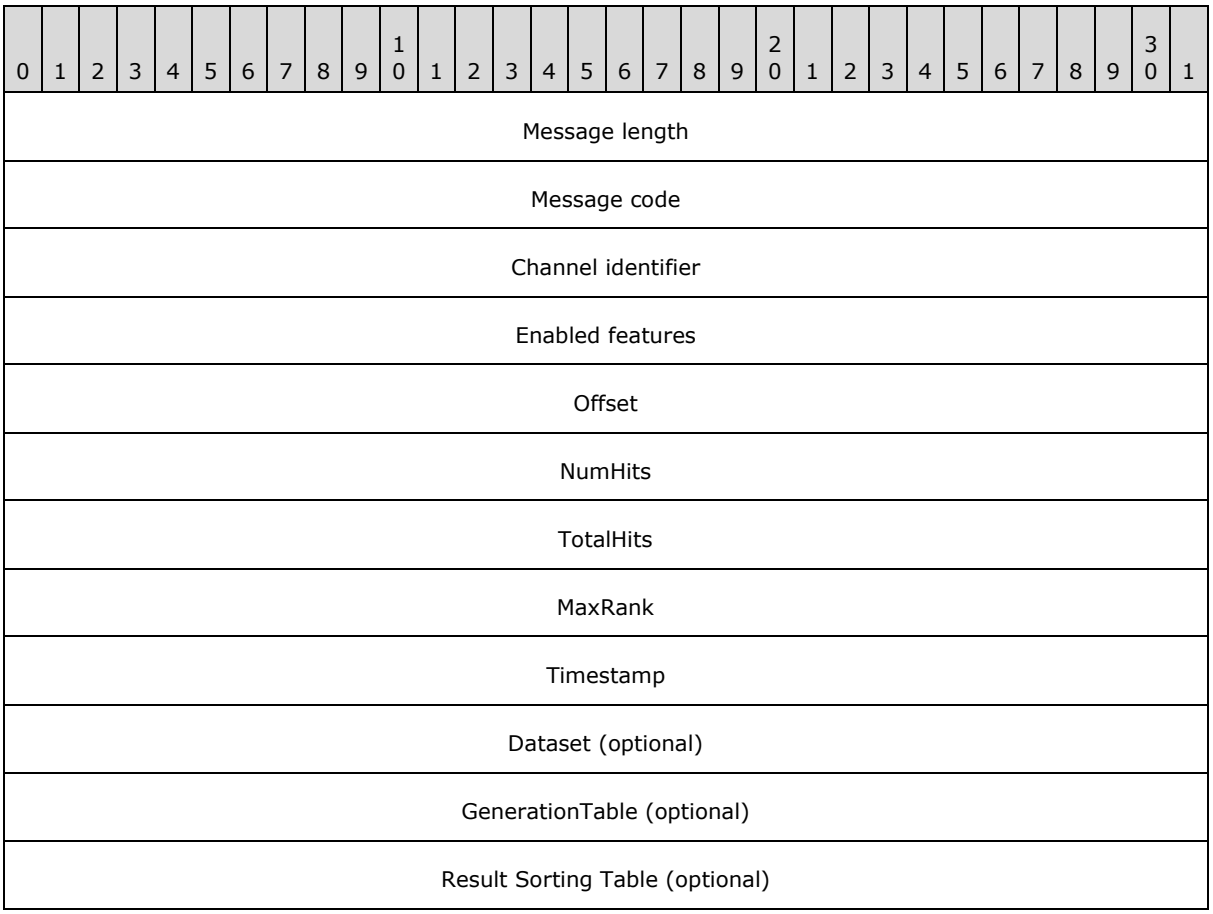
Parameter type	Description
uint32_b	Arity
uint32_b	Fixed integer to add to results in the result set if the results contain the operators on which to perform the XRANK operation. Negative values SHOULD reduce rank. Negative values MUST be stored in two's complement form.
uint32_b	Boost all hits in the result set. It is disabled if it is set to 0, which specifies that only hits that have a calculated rank are changed.

Everything operator

This matches all documents in the search index. It has an arity of 0, and takes no parameters.

2.2.7 Query Response

This response message from the protocol server to the protocol client contains the result of a query request sent from the protocol client to a protocol server. The message format is specified in the following table.



AggregationData (optional)
Field Collapsing with Collapsed Results Removed (optional)
Search Coverage (optional) … … …
Hit list (optional)

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. The length MUST be less than 500,000,008 bytes.

Message code (4 bytes): This is of type **uint32_b** and contains the value 217.

Channel identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request to the response (section 3.2.1).

Enabled Features (4 bytes): This is of type **uint32_b**, and it specifies the response features that the protocol server enabled. Values are specified in the following table. All other bits MUST be set to 0.

Bit mask	Referenced name	Meaning
0x00000001	Dummy flag.	Not used. MUST be set to 1.
0x00000002	DatasetPresent	When this is set, each result entry in the hit list field contains a **Dataset** field.
0x00000004	SiteIdPresent	When this is set, each result entry in the hit list field contains a **SiteID** field.
0x00000010	SortDataPresent	Specifies that the result sorting field is present in the message.
0x00000020	AggregationDataPresent	Specifies that the **aggregationdata** field is present in the message.
0x00000040	CoveragePresent	Specifies that search coverage information is present in the message in the **search coverage** field.
0x00000080	GenerationPresent	This flag MUST be set and the **GenerationTable** field MUST be present in all query responses.
0x00000100	CollapseRemovePresent	When this is set, the message MUST contain information associated with field collapsing in the collapsed field specification field. In addition, a count value MUST be included for each query hit in the query hit list as specified in the Hit List section.

Offset (4 bytes): This is of type **uint32_b** and specifies the offset from the first result item to the first returned result item.

NumHits (4 bytes): This field is of type **uint32_b** and specifies the number of item identifiers in this query response. If this value is greater than 0, then a query hit list that contains the same number of items is included in the message.

TotalHits (4 bytes): This is of type **uint32_b** and specifies the total number of items matching the query from the search index managed by this protocol server component.

MaxRank (4 bytes): This is of type **uint32_b** and specifies an estimate of the maximum rank that can be calculated for an item, based on its correspondence to or deviation from the query terms. This value is not precise, so the protocol client MUST NOT interpret the rank value for each item and the **maxrank** field as normalized values. Hence, the protocol client cannot use this field to determine how relevant an item match is on a percentage scale.

The protocol client and protocol server do not impose any restrictions on the range and values of rank fields, except that the rank values SHOULD specify how well a query matches an item that is contained in the same query hit list, and that it can be stored in a **uint32_b**. A higher value means an item is believed to be more relevant to a query.

Timestamp (4 bytes): This is of type **uint32_b** and is not used. It is set to 0, and the protocol client MUST ignore it.

Dataset (4 bytes): This is an optional field of type **uint32_b** and is contained in the message payload if the **DatasetPresent** flag is set. It is set to 0, and the protocol client MUST ignore it.

GenerationTable (variable): This optional field is a variable-length byte array, and MUST be included in the response if the **GenerationPresent** flag is set, and identifies which versions of the search indexes the query hits are from. The format of this field is specified in the following table.

Field	Type / Length	Value	Meaning
Length	Uint32_b	This value MUST be set to 8.	Number of bytes the generation table consists of, not counting this field.
Leaf	uint32_b	This value MUST be set to 1.	Indicates that protocol server that responded was a leaf node.
Generation identifier	uint32_b	Greater than or equal to 0.	Numeric value representing the generation of indexes used when evaluating the search query.

Result Sorting Data (variable): This optional field is a variable-length string that specifies the results of sorting the response from the protocol server if the **SortDataPresent** flag is set in the **enabled features** field. The field consists of two data chunks, **SortIndex** and **SortData**, which are specified in the following sections. The protocol client receives this data so it can merge sorted lists of items from multiple protocol servers into one list of items. These items can then be retrieved in the result details request stage. For example, to sort results from multiple protocol servers on the title of an item, the protocol client needs to know the titles of all items returned from all protocol servers.

The **SortIndex** and **SortData** chunks appear in the message in the order they are specified here.

SortIndex

This is a series of **numhits** fields that specify the byte offset of the next **SortData** field from the beginning of the previous **SortData** field. The first **SortData** field occurs at offset 0 relative to the beginning of the **SortData** buffer. The first **SortIndex** entry specifies the offset to add to the beginning of the **SortData** field to locate the second element in the **SortData** buffer. The next

SortIndex entry specifies the offset to add to the previous computation to locate the third element in the **SortData** buffer, and so on. The last entry specifies the address for the beginning of the **SortData** buffer. This is specified in the following table.

Field	Type / Length	Value	Meaning
SortIndex[1]	uint32_b	Greater than or equal to 0	Byte offset relative to the beginning of data area for item number 2.
SortIndex[2]	uint32_b	Greater than or equal to 0	Byte offset relative to the beginning of the data area for item number 3
...	uint32_b	Greater than or equal to 0	...
SortIndex[Number of Items returned]	uint32_b	Greater than or equal to 0	Byte offset relative to the first byte of the SortData field.

SortData

The **numhits** field specifies the number of item data areas. The first element begins at offset 0 relative to the beginning of the **SortData** buffer. The second element begins at the offset specified in the **SortIndex[1]** field and so on. The following table shows the scheme of the **SortData** data structure.

Item number	Beginning of data area	End of data area	Meaning
1	0	SortIndex[1]-1	Byte array that contains values of managed property sorted for item 1
2	SortIndex[1]	SortIndex[2]-1	Byte array that contains values of managed property sorted for item 2
...			...
NumberOfDocs	SortIndex[NumberOfDocs-1]	SortIndex[NumberOfDocs]-1	Byte array that contains values of managed property sorted for item NumberOfDocs

Each item data area contains the values on which to sort the managed properties. The protocol server concatenates the managed properties to perform multi-level sorting, but only after each managed property is encoded. For example, for a multi-level sort on a managed property of type int64_b and one of type datetime64_b, the managed properties are encoded separately, and then concatenated. Encoding and decoding of the internal property types in an item data area, and the value for each field is specified in the following table.

Type	Value decoding/encoding for ascending sort order	Value decoding/encoding for descending sort order
int64_b	Value **XOR** 0x8000000000000000	Value **XOR** 0x7fffffffffffffff
uint64_b	None	Value **XOR** 0xffffffffffffffff

Type	Value decoding/encoding for ascending sort order	Value decoding/encoding for descending sort order
datetime64_b	None	Value **XOR** 0xffffffffffffffff
float32_b	Encode: if value >= 0 **XOR** 0x80000000 else **XOR** 0xffffffff Decode: if value <0 **XOR** 0x80000000 else **XOR** 0xffffffff	If value >=0 **XOR** 0x7fffffff
Byte array that contains an UTF-8 encoded string.	None	0xFF - value for each byte in the byte array

AggregationData (variable): This optional field is a variable-length byte array that contains aggregation information requested in the query request. It MUST appear in the message if the **AggregationDataPresent** flag is set. The field format is specified as follows:

AggrDataLen (4 bytes): This field is of type **uint32_b** and specifies the number of subsequent bytes in the **aggregationdata** field.

Version (4 bytes): This field is of type **uint32_b** and specifies the version of the aggregation specification. It MUST be set to 0x01000001.

AggrElement[N] (variable): An aggregation information element that is contained in the aggregation data. A set of aggregation information elements is returned for each aggregator. The total number of elements (n) is related to the aggregator type and the aggregation data that was requested in the query request. Format and values are specified in the following sections.

AggrElement Generic Field Format

This section specifies the generic field format for each aggregation information element in the aggregation data **AggrElement** field.

Field	Type / Length	Value	Meaning
signature	uint32_l	A 32-bit bitmask that contains multiple data elements	See the following table for a specification of the contents and how to interpret it.
internal	uint32_l	Any unsigned 32-bit value	This field MUST be set to 0.
Aggregation Element Data	Variable-length field		The length and encoding of this field is specified in the **AggrElement** format for the specified aggregator. It is based on the Data Type (T) as specified in the Aggregation Data types section, and the Aggregator Type (A) of the signature, as specified in the Aggregation Request Types section. The syntax for each aggregation element type is specified in separate sections following the Aggregation Data Types section.

The signature is a bitmask that is specified as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
I	B	M	A																	T							D				P

Fields are specified as follows.

P (1 bit): MUST be set if a subset of the search index partitions did not return any aggregation data. As a consequence the aggregation result could be incomplete.

D (6 bit): Data Type of the indexed content data the aggregation is based on. Data types are specified in the Aggregation Data types section.

T (7 bit): Data Type of the aggregation data, as specified in the Aggregation Data types section.

A (15 bit): Aggregator type, as specified in the Aggregation Request Types section.

M (1 bit): MUST be set if aggregation cut-off occurred. If set, the **maxerror** field MUST be present in the **AggrElement**.

B (1 bit): MUST be set if the element contains histogram aggregation buckets.

I (1 bit): MUST be set if the aggregation buckets are indexed and the Aggregation Element Data contains the number of aggregation buckets present.

For example, if the protocol server used the **sum** function on an aggregated data set where the indexed data type was int32_l, and the resulting aggregated data type was int64_l, then **D** is equal to 10, **T** is equal to 11 and **A** is equal to 2. See the Aggregation Data Types section for a list of data types.

D and **T** can be different types in cases where the result of the aggregation function on the indexed content data cannot be represented in the same data type. For example, a sum of many uint32_l values can overflow the data type and needs to be represented as a uint64_l in the aggregation data.

Aggregator Request Types

Aggregation results are specified in several formats. For example, aggregation function results for **min** or **max functions** are specified differently from aggregation buckets results for **hist** requests. The type of aggregation is contained in the **signature** field, and it specifies aggregator types from the following table. The syntax for the **AggrElement** associated with the aggregator type is specified in the **AggrElement** Syntax column of the table.

Aggregator Type Value	Aggregation function	Description	AggrElement Syntax
0	Max	Maximum value contained in the aggregated data set	Specified in the Aggregation Function Result Syntax section.
1	Min	Minimum value contained in the aggregated data set	Specified in the Aggregation Function Result Syntax section.
2	Sum	Aggregated sum across the aggregated data set	Specified in the Aggregation Function Result Syntax

Aggregator Type Value	Aggregation function	Description	AggrElement Syntax
			section.
5	hist :buckets <n>	Count for each aggregation bucket when a fixed number <n> buckets are requested.	Aggregation bucket index, as specified in the Aggregation Bucket Syntax section.
100	Hitcount	Query hits used for generating the histogram. Total number of query hits on which the aggregated values and histogram are based.	Specified in the Aggregation Function Result Syntax section.
101	Count	Sample Count. An aggregator might take on multiple values for an item. This is reflected in the Sample Count value, which denotes the number of samples from which the histogram was computed.	Specified in the Aggregation Function Result Syntax section.
102	Countnz	The number of query hits or unique items that contain at least one value for this aggregator.	Specified in the Aggregation Function Result Syntax section.
103	hist :buckets	Count for each aggregation bucket when the buckets are of variable width	Aggregation bucket index, as specified in the section Aggregation Bucket Syntax.
104	hist :buckets :unique	Count for each aggregation bucket when individual buckets are requested for each unique value in the result set	Individual buckets that contain unique values, as specified in the Unique Value Individual Bucket Syntax section.
105	hist :width	Count for each aggregation bucket when the buckets are of fixed width	Aggregation bucket, as specified in the Content Aggregation Bucket Syntax section.
106	Refine	Count for each aggregation bucket when the request is to refine the previous aggregation	Aggregation bucket as specified in Refined Aggregation Buckets section.

Aggregation Data Types

The data type derived from the signature field is specified in the following table. For each data type, the corresponding field encoding is specified in the Encoding column.

Data type identifier	Description	Encoding
1	String	UTF-8
2	8-bit unsigned integer	8-bit value
3	16-bit unsigned integer	uint16_l
4	32-bit unsigned integer	uint32_l

Data type identifier	Description	Encoding
5	64-bit unsigned integer	uint64_l
6	96-bit unsigned integer	uint96_l
7	160-bit unsigned integer	uint160_l
8	8-bit signed integer	8-bit value
9	16-bit signed integer	int16_l
10	32-bit signed integer	int32_l
11	64-bit signed integer	int64_l
12	96-bit signed integer	int96_l
13	160-bit signed integer	int160_l
14	32-bit floating point value as specified in [IEEE754]	float32_l

Aggregation Function Result Syntax

This specifies the syntax for the serialized **AggrElement** field that represents the result of an aggregation function that returns one value.

signature (4 bytes): This field is of type **uint32_l**, and it specifies the signature of the aggregation request, as specified in the AggrElement Generic Field Format section.

internal (4 bytes): This field is of type **uint32_l**, and it MUST be set to 0.

value (integer or float variable): The type and length of this field is specified by the aggregated data type (T) in the **signature** field.

Aggregation Bucket Syntax

This specifies a serialized aggregation bucket:

signature (4 bytes): This field is of type **uint32_l**, and it specifies the signature of the aggregation request, as specified in the **AggrElement** Generic Field Format section.

internal (4 bytes): This field is of type **uint32_l**, and it MUST be set to 0.

aggregation buckets (4 bytes): This field is of type **uint32_l**, and it specifies the number of aggregation buckets in this aggregation element. It is greater than or equal to 0.

aggregation bucket[N] (variable): Represents the aggregation buckets. The aggregation buckets are specified as an array that contains the following fields.

index (4 bytes): This field is of type **uint32_l**, and it specifies the index for the aggregation buckets. Aggregation buckets are specified as an array where the index corresponds to the fixed length buckets in the request. This field is greater than or equal to 0.

count (4 bytes): This field is of type **uint32_l**, and it specifies the number of aggregation buckets. This is the number of samples that occur in the specified range in the aggregated result set. It is greater than or equal to 0.

Content Aggregation Bucket Syntax

This specifies a serialized aggregation bucket:

signature (4 bytes): This field is of type **uint32_l**, and it specifies the signature of the aggregation request, as specified in the AggrElement Generic Field Format section.

internal (4 bytes): This field is of type **uint32_l**, and it MUST be set to 0.

aggregation buckets (4 bytes): This field is of type **uint32_l**, and it specifies the number of buckets in this aggregation element. It is greater than or equal to 0.

aggregation bucket[N] (variable): Represents the aggregation bucket format as specified in the following table. The aggregation buckets are specified as a pair of fields. The two fields are specified as follows:

value (Data Type (T)): Data value associated with this aggregation bucket. The protocol server sends one bucket per value, which is greater than or equal to 0. The type and length of the data type are contained in the **signature** field.

count (4 bytes): This field is of type **uint32_l**, and it specifies the number of samples in the aggregated result set that contain the value specified in the **value** field of the aggregation bucket. MUST be greater than or equal to 0.

Unique Value Individual Bucket Syntax

This specifies the aggregation bucket when the individual buckets for each unique value were contained in a previously retrieved result set.

signature (4 bytes): This is of type uint32_l, and it specifies the signature of the aggregation request, as specified in the AggrElement Generic Field Format section.

internal (4 bytes): This field is of type uint32_l, and it MUST be set to 0.

maxerror (4 bytes): This field is of type uint32_l, and it specifies the maximum count for aggregation buckets that are not displayed in the result set if a cutoff occurs. Otherwise, it is set to 0.

buckets (4 bytes): This field is of type uint32_l, and it specifies the number of aggregation buckets in this aggregation element. It MUST be a positive integer.

bytes (4 bytes): This field is of type uint32_l, and it specifies the total number of bytes in the buckets of this aggregation element, which are the all of the data elements that occur after this field. It MUST be a positive integer.

bucket[buckets] (variable): Aggregation buckets as specified in the following table.

Field	Type / Length	Value	Meaning
strlen	uint32_l	Greater than or equal to 0.	Length of the string representing this aggregation bucket
str	Character array whose length is specified in the **strlen** field	UTF-8 characters	The text string representing this aggregation bucket
count	uint32_l	Greater than or equal to 0.	The number of samples in the bucket in the aggregated result set that contain the specified string

Refined Aggregation Bucket Syntax

This syntax specifies the format for aggregation buckets when the protocol client requests a refinement of a previously-retrieved aggregation. When more than one aggregator is requested in the query, the aggregation elements of a specific aggregator type appear in the message in the same sequence as requested in the query.

signature (4 bytes): This is of type **uint32_l**, and specifies the signature of the aggregation request, as specified in the **AggrElement** Generic Field Format section.

internal (4 bytes): This is of type **uint32_l**, and it MUST be set to 0.

buckets (4 bytes): This is of type **uint32_l**, and it specifies the number of aggregation buckets in this aggregation element. It MUST be a positive integer.

count[buckets] (4 bytes): A field of type **uint32_l** that specifies the number of samples in the aggregated result set that contain the specified string. This field MUST be greater than or equal to 0.

Field Collapsing with Collapsed Results Removed (variable): This optional field specifies how the managed properties were collapsed. It MUST appear in the message if **CollapseRemovePresent** flag is set in the **enabled features** field. Field collapsing is performed only on numeric fields. The field is formatted as specified in the following table.

Field	Type / Length	Meaning
NumberNotCollapsed	uint32_b	Number of items that had no value or that are undefined for the managed property on which the collapsing is performed
NumberCollapsed	uint32_b	Number of items that were collapsed.
CollapseValue[0]	uint64_b	Value of the collapsed managed property
CollapsedValueCount[0]	uint32_b	Number of items with the value specified in the **CollapseValue[1]** field in the collapsed managed property
…	…	…
CollapseValue[NumberCollapsed]	uint64_b	Value of the collapsed managed property
CollapsedValueCount[NumberCollapsed]	uint32_b	Number of items in the collapsed managed property that contain the value specified in Value[NumberPairs]

Search Coverage

This field appears in the message if **CoveragePresent** flag is set in the **enabled features** field. The field format is specified in the following 3 fields.

internal (8 bytes): This field is of type **uint64_b**, and contains information specific to a particular implementation of the protocol server. It is greater than or equal to 0, and the protocol client MUST ignore it.

nodes (4 bytes): This field is of type **uint32_b**, and specifies the number of index partition nodes associated with the evaluation of the query. It is associated with a specific protocol server implementation, and is greater than or equal to 0. The protocol client SHOULD ignore it.

fullResult (4 bytes): This field is of type **uint32_b**, and specifies whether the query result is complete. It MUST be set to 0 if a partial result is returned, and it MUST be set to 1 if the query result is complete.

Hit List (variable): This field appears in the message if the **numhits** field is greater than 0. The number of occurrences of the field is contained in the **numhits** field. The fields are specified as follows.

docid (4 bytes): This field is a positive integer of type **uint32_b**, and it specifies an item identifier that is unique to each protocol server, rather than globally unique. The most significant bit in the **docid** field specifies whether or not the protocol server performed field collapsing for this document. If it is set to 1, field collapsing has taken place. The most significant bit MUST be set to 0 to retrieve the actual **docid** field, which can be used in subsequent requests.

metric (4 bytes): This field is a positive integer of type **uint32_b**, and it specifies the rank for this result.

part_id (4 bytes): This field is a positive integer of type **uint32_b**, and it specifies the identifier of the index partition that contains the query hit. This is used when constructing the **docids** field in the corresponding result details request.

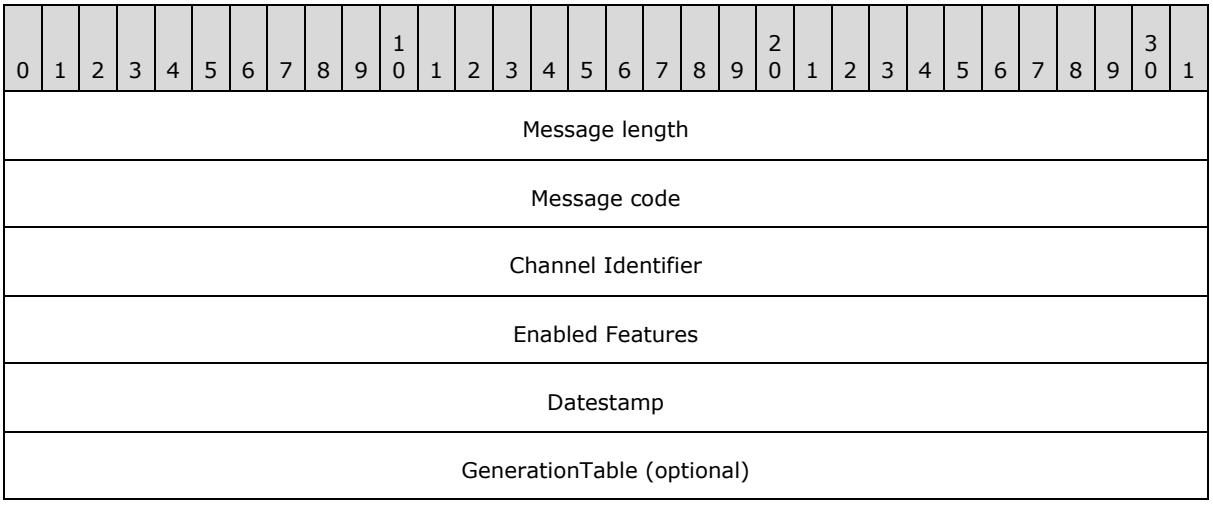
docstamp (4 bytes): This optional field is of type **uint32_b**, and it specifies when the item was indexed. The age is specified as the number of seconds elapsed after 1970-01-01 UTC.

siteid (4 bytes): This optional field is a positive integer of type **uint64_b**, and it MUST contain the value of the managed property used for field collapsing if the **SiteIdPresent** flag in the **enabled features** field is set.

count (4 bytes): This optional field is a positive integer of type **uint32_b**, and it specifies the number of items in this collapse group. MUST be present if the **CollapseRemovePresent** flag in the **enabled features** field is set.

2.2.8 Result Details Request

This message contains a result detail request that the protocol client sends to the protocol server. It is based on query result information previously sent to the protocol client in a query response. The format of the message is specified in the following table.



Ranking (optional)
Query flags (optional)
Wanted Summary Class (optional)
Query stack (optional)
Current Date and Time (optional) …
Docids (optional)

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. The length MUST be less than 20,000,008 bytes.

Message code (4 bytes): This is of type **uint32_b** and contains the value 219.

Channel identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response (section 3.2.1).

Enabled Features (4 bytes): This field is of type **uint32_b**, and it specifies enabled features by setting bits according to the following table. All other bits are set to 0.

The corresponding field data MUST be present in the message for each enabled feature. The fields that correspond to each feature are specified after the next table. If the protocol client requests result details for any items, a series of **Docids** sections MUST follow at the end of the payload.

Value	Feature name	Meaning
0x00000001	Item Identifier **Datestamp**	Each item identifier in the **docids** field MUST be followed by a date stamp. MUST be enabled. This is the only place in the message that content is changed for this bit flag, rather than appending a content section.
0x00000004	Query stack	Specifies that the message includes the query stack from the original query request. This enables the protocol server to highlight item summaries in the query results. This field SHOULD be set.
0x00000008	**WantedResClass**	Specifies that the protocol client is requesting a specific result view from the protocol server.
0x00000010	Ranking	MUST be set if **rank log** is enabled.
0x00000040	Current date and time	Current date and time on the protocol client.
0x00000080	**GenerationPresent**	MUST be set. Specifies the revision of the search index on which to perform queries.

Datestamp (4 bytes): This field is of type **uint32_b** and specifies the age of the protocol server that the protocol client is sending the requests to. It is specified in seconds that have elapsed since 1970-01-01 UTC.

GenerationTable (variable): This optional field MUST be present in the request if the **GenerationPresent** bit flag of the **enabled features** field of the request is set. It contains three **uint32_b** values. The first **uint32_b** field represents the length of the data chunk in bytes. If the protocol client does not request a specific revision of a search index, this **uint32_b** field MUST be equal to zero. To request a specific index generation or revision of a search index, a protocol client copies the leaf and generation identifier fields from the preceding query response to this field. If the initial **uint32_b** field is greater than 0, then the **GenerationTable** field MUST be according to the specification in the following table.

Field	Type / Length	Value	Meaning
Length	uint32_b	This value MUST be set to 8.	Number of bytes the generation table consists of, not counting this field.
Leaf	uint32_b	This value MUST be set to 1 if the length of the field is greater than 0.	Specifies that the protocol server that responded is a leaf node.
Generation identifier	uint32_b	Greater than or equal to 0.	Numeric value representing the generation of indexes used when evaluating the search query.

Ranking (4 bytes): If the **Ranking** flag is set in the **enabled features** field, this optional **uint32_b** field specifies the rank profile to use.

QueryFlags (4 bytes): This optional field is of type **uint32_b**. If the **Ranking** flag is set in the **enabled features** field, this specifies query flags that are contained in the request. This is the information that is copied from the original request if it is present. It is used to re-evaluate the query to generate extra debug information if the **include ranking information** bit flag is set in the copied **queryflags**.

Wanted Summary Class (4 bytes): This optional field is of type **uint32_b**, and it specifies the **summary class** to use in the response. It MUST be present if the **WantedResClass** bit flag in the **enabled features** field is set. A request for a specific class returns the managed properties that are contained in that class. The mapping between this field and the summary class is specified in the file named "summary.cf", as specified in [MS-FSSCFG] section 2.18.

Query Stack (variable): This optional field is a variable-length byte array that contains the serialized query stack, if the **query stack** flag is set in the **enabled features** field. The query stack consists of a **uint32_b** field that specifies the approximately number of operators that were serialized onto the query stack, followed by a **uint32_b** field that specifies the length of the serialized query stack, followed by the serialized query stack as specified in section 2.2.6. The length field is only the length of the query stack; it does not include the two **uint32_b** fields. The structure of the query stack field is specified in the following table.

Field	Type / Length	Value	Meaning
Number of stack entries	uint32_b	Unsigned integer	Specifies approximately how many operators exist on the stack. Not exact.
Length	uint32_b	Unsigned integer	Length of query stack data chunk.
Serialized query stack.	Variable-length byte array	Data array	Included for re-evaluation of the query by the protocol server.

Current Date and Time (8 bytes): This optional field is of type **datetime64_b**, and specifies the current date and time on the protocol client for freshness boost calculations. Reevaluation of rank only occurs if the **rank log** bit flag is enabled. It MUST be present if the **Current date and time** flag is set.

Docids (variable): This optional field is of variable length and contains document identifiers associated with items for which the protocol client is requesting result details. The 3 fields it contains are specified as follows:

Docid (4 bytes): This field is of type **uint32_b** and it contains the document identifier. The docid MUST be specified without the highest bit set. The highest bit is used to specify if collapsing has been done in query responses, see section 2.2.7.

part_id (4 bytes): This field is of type **uint32_b** and it contains the internal partition identifier. It MUST be the same as the **part_id** field that is contained in the query hit for this document, as specified in the hit list field in section 2.2.7.

docstamp (4 bytes): MUST be the same as the docstamp field that is contained in the hit list, from the query response, for this document. This field MUST be set.

This section MUST always be the last part of the Result Details request message. By counting the number of bytes that have been read, the protocol server can read the rest of the message payload and treat it as a list of item identifiers that are requested by the protocol client.

2.2.9 Result Details Response

This message contains the response to a result details request, including the managed properties of an item and highlighted managed properties. Each message contains result details for one query hit from the **docids** field in the corresponding result details request. The message syntax is specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Message length																															
Message code																															
Channel Identifier																															
Docid																															
Summaryclass																															
Item Content																															

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message in bytes, excluding the length of this field. The length MUST be less than 500,000,008 bytes.

Message Code (4 bytes): This is of type **uint32_b** and contains the value 205.

Channel Identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response flow (section 3.2.1).

Docid (4 bytes): This field is of type **uint32_b** and it contains the item identifier of the item being returned. It is the same as the item identifier contained in the result details request.

Summaryclass (4 bytes): This field specifies the summary class used for this result details response. A summary class represents a view of the search index that can be returned in a result details response. A protocol server SHOULD support multiple summary classes. The summary.cf and summary.map files are used to store the configuration on the content of each summary class. These are available through the [MS-FSCX] protocol. See [MS-FSSCFG] section 2.18 and [MS-FSSCFG] section 2.19 for more information about these files.

If a summary class was not specified in the corresponding result details request, the protocol server uses the default summary class specified in summary.cf.

Item Content (variable): This field contains the item summaries for each query hit in the query hit list. It contains **summaryfields** fields that are specified as follows.

summaryfield[N] (variable): Data for each managed property for the current item, as specified in the summary.cf file for the summary class in use. The number of managed properties returned is specified in the summary.cf file. The following table specifies how the different managed property field types are serialized.

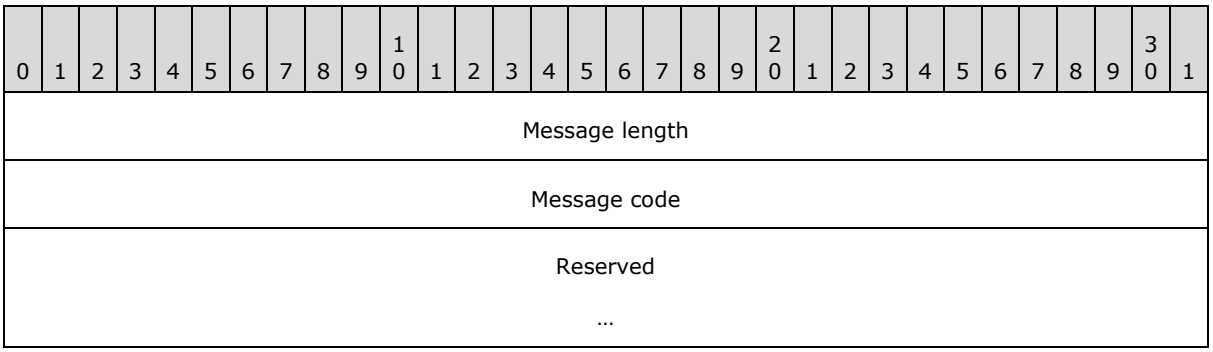
Field	Type / Length	Value	Meaning
length	uint16_l or uint32_l	Unsigned integer of the same size as the type	Specifies the length of the string field. The field is of type uint16_l for item summary fields of type string and data, and the field is of type uint32_l for item summary fields of type longstring. For 'longstring', the length MUST be masked with the bitmask 0x80000000 to find the length.
string	Variable length byte array	Data	Content of the item property

For an item summary of type string, the string field contains UTF-8 encoded characters. For an item summary of type **longstring**, the content can be compressed or uncompressed. The most significant bit specifies whether the field is compressed. A uncompressed field of type **longstring** contains UTF-8 encoded characters. The format of a compressed field is specified in the following table.

Field	Type / Length	Meaning
length	uint32_l	Length of the uncompressed data
data	Byte array	A byte array of the length of the previous field, containing compressed UTF-8 encoded text strings. The compression MUST use the zlib format, as specified in [RFC1950], and be compressed using the "deflate" compression method.

2.2.10 Queue Length Message

The Queue Length message content MUST be ignored. The message structure is specified in the following.



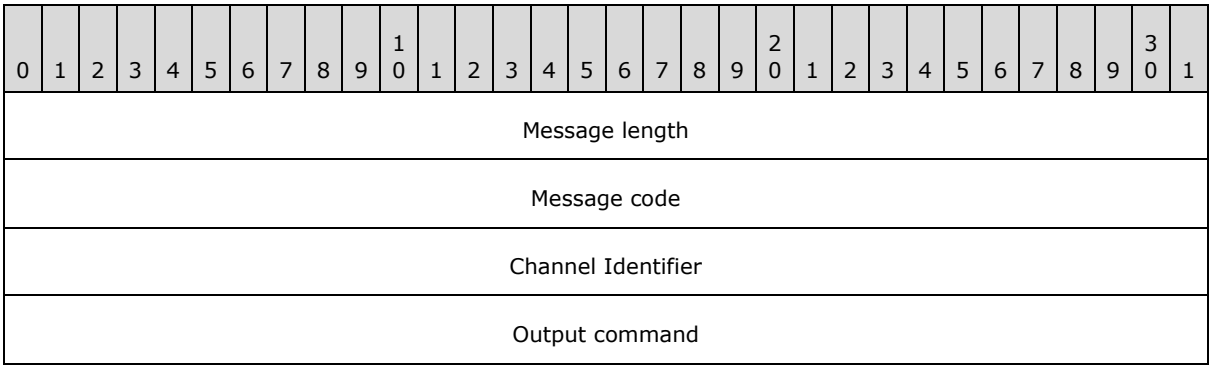
Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message excluding this field. It MUST be set to 12.

Message code (4 bytes): This is of type **uint32_b** and MUST contain the value 216**.**

Reserved (8 bytes): This MUST be ignored.

2.2.11 Statistics Query Request

This message requests a statistics query response from the protocol server. The message structure is specified in the following table.



Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message excluding this field. It MUST be set to 12.

Message code (4 bytes): This is of type **uint32_b** and contains the value 222.

Channel identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response (section 3.2.1).

Output command (4 bytes): This is of type **uint32_b** and it specifies how the protocol server formats the statistics XML (see [XML10] for more on XML). The values are specified in the following table.

Value	Meaning
1	Specifies that the protocol server formats the statistics XML as an XML fragment.
2	Specifies that the protocol server wraps the statistics XML fragment so that the output is a complete XML document.

The protocol server adds the following header to wrap the output, when the output command is equal '2':

```
<?xml version="1.0"?>
<!DOCTYPE search-stats SYSTEM "search-stats-1.0.dtd">
<search-stats>
```

The protocol server then adds the following footer to close the search-stats XML clause.

```
</search-stats>
```

2.2.12 Statistics Query Response

This message returns query and cache statistics for the protocol server. The message structure is specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Message length																															
Message code																															
Channel Identifier																															
Statistics payload length																															
Statistics payload																															

Message length (4 bytes): This is of type **uint32_b** and specifies the length of the message excluding this field. It MUST be greater than 0 and less than 70,000,008.

Message code (4 bytes): This is of type **uint32_b** and contains the value 223.

Channel identifier (4 bytes): This is of type **uint32_b** and specifies the identifier that associates the request and the response (section 3.2.1).

Statistics payload length (4 bytes): This is of type **uint32_b**, and it specifies the length of the statistics payload.

Statistics payload (variable): This is of type **string** and it contains a UTF-8 encoded XML fragment. This field response contains statistical information about the protocol server. The format is specified in [MS-FSSADM] section 7.

3 Protocol Details

3.1 Common Details

This section contains details that apply to both the protocol server and the protocol client.

The protocol architecture enables an M:N relation between protocol clients and protocol servers. The protocol does not impose any limitations on M or N. Communication in this protocol is specified as occurring between one protocol client and one protocol server.

Multiple instances of protocol clients and protocol servers serve several purposes:

- Support partitioning of the entire searchable index over several protocol servers
- Enable configurations with multiple instances of protocol servers to achieve increased performance and fault-tolerance
- Enable configurations with multiple instances of the protocol client for increased performance and fault-tolerance

3.1.1 Common Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The protocol performs queries against a searchable content index and returns results that match the query parameters. This MAY be an index in the format documented in [MS-FSIXDS].

The protocol client sends a request to the protocol server, and the protocol server responds with the matching result. An example of a full search and retrieve sequence that contains 5 types of messages is specified in the following figure. Messages are associated with responses as specified in the following table.

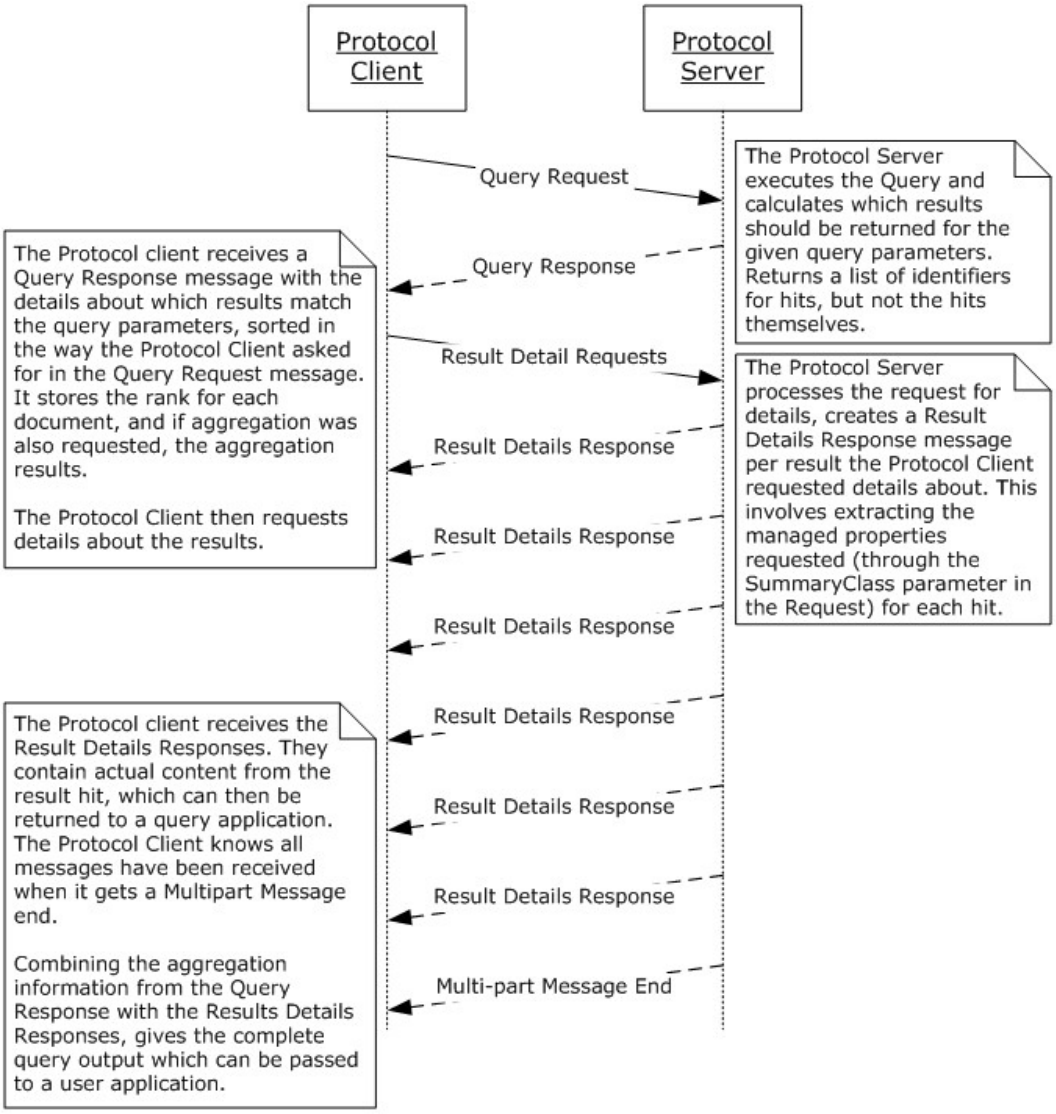


Figure 5: Full search and retrieve sequence

The messages, their origins, and associated responses are specified in the following table.

Messages originating from the protocol client	Responses originating from the protocol server
PING Request (section 2.2.3)	PING Request Answer (section 2.2.4)
Query Request (section 2.2.6)	Query Response (section 2.2.7)
	Queue Length message (section 2.2.10)
Result Details Request (section 2.2.8)	Result Details Response (section 2.2.9)
	Multi-part Message End (section 2.2.2)
Statistics Query Request (section 2.2.11)	Statistics Query Response (section 2.2.12)

Messages originating from the protocol client	Responses originating from the protocol server
Any message	Error (section 2.2.5)

How the protocol client and protocol server process these messages is specified in sections 3.2 and 3.3.

3.1.2 Timers

None.

3.1.3 Initialization

A TCP/IP connection from the protocol client to the protocol server MUST exist before the protocol can be used. The default protocol server port is 13052.

The protocol client and protocol server SHOULD use only one TCP/IP connection per protocol client/protocol server pair. Multiplexing of multiple queries on this connection is performed by using the message **channel identifier** field as specified in section 3.2.1.

The protocol client and protocol server use configuration files associated with the search index to encode and decode the protocol. These files are available from the Configuration Server, using the protocol specified in [MS-FSCX]. The paths to the files are specified in [MS-FSSCFG]. If the protocol server or the protocol client cannot read the files specified in the following table, they MUST have access to the same configuration information through some other method.

File Name	Description
summary.cf	Specifies the available summary classes in the system and what managed properties constitute those summary classes. For more information, see [MS-FSSCFG] section 2.18.
Index.cf	Specifies the structure of the search index. The protocol client requires this to determine whether named managed properties in the request are valid for the search index on the protocol server. For more information, see [MS-FSSCFG] section 2.1.2.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

The request sequences are documented in general in the following sections. For more information about messages sequences and handling, see sections 3.2.5 (protocol client) and 3.3.5 (protocol server).

3.1.5.1 PING

The protocol client uses PING messages to monitor the status of connections to at least one protocol server. The protocol client MUST send PING requests to monitor the protocol server (section 2.2.3) each second and the protocol server MUST send a PING response immediately (section 2.2.4).

3.1.5.2 Query

A query consists of two messages, a request (section 2.2.6) from the protocol client, and a response from the protocol server (section 2.2.7). The protocol client uses the response to request more

information about the items in the result set from the response. If the protocol client only wants to check if there are items that match the query terms instead of retrieving the specific item details, it can skip the result details stage.

3.1.5.3 Result Details

The protocol client requests information about query hits that were listed in the previous query response. The protocol server sends one response (section 2.2.9) for each query hit. After returning all requested responses, the protocol server sends a Multi-Part Message end response (section 2.2.2) to the protocol client to specify that the response is complete. The protocol client combines information from the initial and subsequent responses to create the full end search result.

3.1.5.4 Errors

If an error occurs on the protocol server during message processing, the protocol server SHOULD send an error message (section 2.2.5) if the protocol client enabled error reporting. If an error occurs because of lack of resources, the protocol server can close the connection immediately without sending an error message, even if the protocol client enabled error reporting. For more information about enabling error messages, see section 2.2.6.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The protocol client sends a request to the protocol server. The protocol client compares the channel identifiers in the request and the response to determine whether they can be paired. A request and the associated response contain the same channel identifier. If a request leads to multiple responses, all of the responses MUST contain the same channel identifier. A protocol client MUST NOT reuse a channel identifier if outstanding responses exist for a request/response pair, and any associated time limits have not expired. An example of the flow of a channel identifier is shown in the following figure.

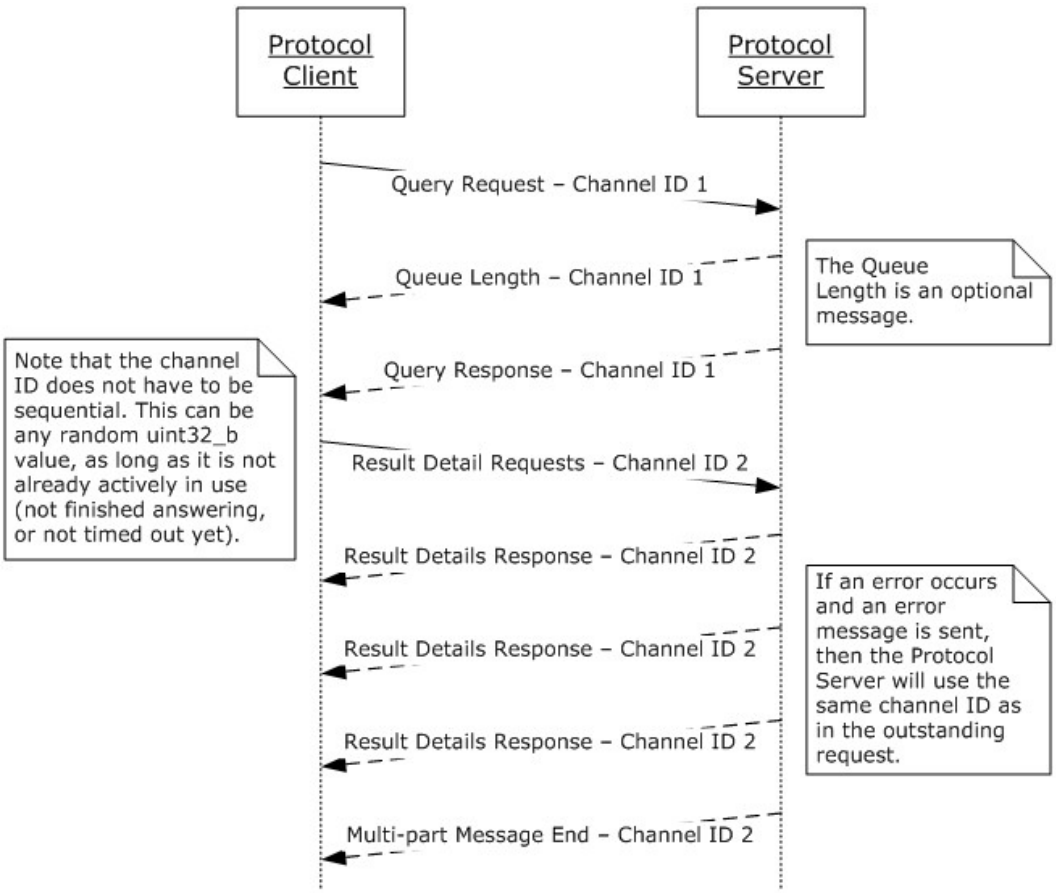


Figure 6: Channel identifier flow

Global states in the protocol client:

SentChannelIds: A list of the channel identifiers currently in use by the protocol client.

ServersAvailable: A list of available protocol servers.

ServersDown: A list of protocol servers which are not available.

LastStartOfServers: A list of the start time of each of the protocol server components.

A full query/result sequence consists of multiple messages. The protocol client needs to store some information during the query sequence, and the following is state held per query by the protocol client:

ChannelId: The current channel identifier used for this query.

QueryFlags: The query flags used in the original query.

ParsedQuery: The parsed query stack sent in the original query.

Index generation: The generational information returned from the protocol server in the query response.

HitList: The list of items from the query response hit list. Each entry contains the **docid**, **part_id** and **docstamp** fields, which are required when the protocol client requests further information about a query hit.

RankHitList: The rank of each entry in the query hit list (from the query response message).

SortData: The sort data returned for the query hits (From the query response message).

AggregationResults: The results of any aggregation specifications in the query request.

CollapseList: A data structure that specifies whether a query hit was collapsed, and if so, which other items in the result set are collapsed with it.

TotalCollapsed: The total number of collapsed entries.

TotalNotCollapsed: The total number of non-collapsed entries.

SentARefineRequest: A **Boolean** field that specifies whether the protocol server will send a refine response to match the refine request.

The protocol tries to avoid duplication of data sent from the protocol server to the protocol client. Because of this, the protocol client needs to assemble a number of these temporary query state variables to form a complete response. The protocol does not impose any restrictions on how the protocol client exposes results to an end-user or a higher-level application, but it SHOULD at least contain the following information:

AggregationResults

HitList (combined with RankHitList to expose the rank).

Item content (the managed properties for each query hit).

3.2.1.1 Handling Multiple Protocol Servers

The protocol architecture enables communication between a protocol client and multiple protocol servers. Typically, architectures that support multiple instances of protocol servers increase performance and increase fault-tolerance. The following two scaling configurations are supported by this protocol:

- Support partitioning of the entire searchable index over protocol servers. In this case each protocol server receives queries for a subset of the searchable index. The protocol client merges the results from the different protocol servers to form a complete result set for search and query applications.
- Enable configurations of redundant protocol servers for increased performance and fault-tolerance by duplicating parts of the searchable index. An application can implement load sharing mechanisms to enable separate protocol server components to support different queries. This protocol does not impose restrictions on the type of load sharing mechanism. The application can implement appropriate methods for handling situations where protocol servers cannot process queries, as long as one redundant protocol server is available.

The two scaling methods MUST be possible to combine. If an implementation supports the first scaling configuration, the protocol client MUST:

- Send all subsequent result detail requests to the protocol server that sent the preceding query response.

- Sort query result entries based on sort data returned in the query response message, depending on the sorting mechanism requested. The protocol client MUST be able to merge sorting data from several protocol servers. Implementations can use different methods to calculate the sorting results, but the data returned MUST follow the specification for the result sorting data field in the query response message (see section 2.2.7). Merging MUST also be possible based on rank.
- Merge aggregation data from multiple protocol server components into one histogram.
- Merge collapsed result sets from multiple protocol servers so that the number of documents per value collapsed is the same as before the merge.
- The channel identifier is not globally unique across several protocol client/servers. The protocol client MUST use the channel identifier and the protocol server address to associate a response with the message that requested it.

3.2.1.2 Error Handling

If the **error messages enabled** flag in the **query flags** field in the request is set (section 2.2.6), then any request except the PING request can cause the protocol server to send an error message if a request fails.

The error code in the error message (section 2.2.5) specifies whether this is a permanent error or a temporary one. Temporary errors are typically corrected when the protocol client automatically resends the query, while permanent errors specify that some corrective action is required. A protocol client MAY use the error code to decide whether or not to automatically handle the error. The default protocol client does not do this.

3.2.1.3 Issuing a Query

A query follows the pattern shown in section 3.1.1. For more information about what the protocol client stores for each part of the request, see section 3.2.5.

3.2.2 Timers

The protocol client MUST implement a timer to govern the timeout for a server response. The protocol server does not guarantee that an error message is sent if an error occurs (even if it is requested), and if a query takes too long, the protocol server SHOULD abort it to prevent excessive resource usage. The timeout in the protocol client SHOULD be configurable and the timeout on the client SHOULD be the same or lower than the timeout on the protocol server to avoid waiting longer than necessary.

The protocol client MUST also have a timer that triggers sending PING Request messages every second to check if the protocol servers are running.

3.2.3 Initialization

See section 3.1.3.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

The protocol client uses the **message code** field to determine the type of message it received. Each message is associated with a unique code; which is specified for each message in section 2.2. The messages that the protocol client uses to communicate with the protocol server are specified in section 3.1.1. If the protocol client receives a message that contains an unknown or non-valid message code, it MUST ignore the message.

3.2.5.1 Receiving an Error Message

If the protocol client receives an error message with a specific channel identifier, then it MUST ignore any other messages from the protocol server that contain this channel identifier until the protocol client reuses it. See section 3.2.1.2 for general handling of error messages.

3.2.5.2 PING Request and Response

3.2.5.2.1 Sending a PING Request and Receiving a PING Request Answer

The protocol client sends a PING request to verify that a protocol server is available. If the protocol client connects to several protocol server components, it sends one message to each component. The protocol client implements simple monitoring with PING requests, because it only determines whether the protocol server responds, rather than analyze the payload of the PING request. The protocol client MUST update the **ServersAvailable** and **ServersDown** lists based on whether it receives a response from the protocol server. The protocol client also stores the **timestamp** field of the PING response for each protocol server in the **LastStartOfServers** list. The protocol client uses this field when sending result details requests.

More advanced search coverage monitoring is specified in section 3.3.5.1.1.1. This determines whether the protocol server is down, even if it responds to the PING request and does not report that the complete search content is available. The default protocol server implementation marks a protocol server up based on whether or not it responds, but uses the content of the PING response to specify the health of the protocol server in more detail in its internal statistics output.

The PING request and response do not use a channel identifier in the message.

3.2.5.3 Query Request and Response

3.2.5.3.1 Sending a Query Request

Any query request a protocol client sends is formatted as specified in section 2.2.6. The query is serialized into the **parsed query** field of the request. The protocol client can implement any query language with this protocol if it can convert from that query language to the parsed query format of the protocol. The protocol client MUST set the **top level search** flag (section 2.2.6) to specify that it is an external top level protocol client.

The protocol client SHOULD comment operators with origin metadata, as specified in section 2.2.6 to improve **hit highlighting** and relevancy calculation, if the protocol server uses this. This is implementation specific, and the protocol server typically supports only a subset of the origin metadata values. If a protocol server does not support a specific value, it MUST ignore it.

The protocol client temporarily stores the query flags and parsed query from the request in the local query state (**QueryFlags** and **ParsedQuery** states). These are used when constructing a result details request later.

The protocol client temporarily stores the channel identifier for this request in **SentChannelIds**, so that it can determine whether a response is valid.

If the protocol client knows that the same query will be resubmitted shortly, but with a different offset (for example for splitting large search results up into several pages of items), the protocol client SHOULD set the **User Cache line** and **Max offset flags** in the **enabled features** field of the query request message to specify this and send the associated query request message fields (as specified in section 2.2.6). This makes it possible for a protocol server implementation to optimize caching of the results for reoccurring queries.

3.2.5.3.2 Receiving a Query Response

The protocol server sends a response to a query request as specified in section 2.2.7. The protocol client verifies that the channel identifier in the response is in the **SentChannelIds** list and the same as the original request. Then the protocol client stores the query hits, any associated ranks, the results of any aggregation, the **GenerationTable** field from the response, any field collapsing data, and the sort information in the temporary query state (as specified in the abstract data model, see section 3.2.1). If the protocol server supports search index revisions, then the protocol client uses the **GenerationTable** field from the response and the **hit list** field to specify the same revision of the items when assembling subsequent requests. If the result set is empty, and there were no query hits, then the protocol client does not send any more messages to this protocol server for this query sequence.

If the protocol server returned a hit list in the response, then the protocol client sends requests for result details for each item and its associated managed properties.

If no items are returned because the **max hits** field in the query request message was set to 0 (even though matching items are present on the protocol server), the protocol client SHOULD return the aggregation results from the query response message to the higher level application (or end user). This is an optimization for when the higher level application or end-user just wants to retrieve aggregation information.

The protocol server uses rank or sort information to merge results if the protocol client queries several protocol servers, as specified in section 3.2.1.1. By performing the merge before the result details request stage, the protocol client avoids sending any unnecessary result details requests for items later dropped during sorting.

If the protocol server sets the **maxerror** field in the returned aggregation data, as specified in the **aggregationdata** field in section 2.2.7, this specifies that a cut-off occurred, and the occurrence count for aggregation buckets in the results could be inaccurate. The protocol client SHOULD send a refine query request (see next section) to recalculate the occurrence counts for the aggregation buckets which were impacted by the cut-off. The protocol client MUST then replace the original returned aggregation occurrence counts in the temporary query state with the results of the refine query request for the specific aggregation buckets which were affected by the cut-off.

If field collapsing was enabled and did take effect, the protocol client SHOULD save the content of the Field Collapsing with Collapsed Results Removed field into the **CollapseList** local query state, so that it can be returned to a higher level application or end-user. This higher level application or end-user can then use this information to tag or reformat the query results to specify that field collapsing has occurred (a normal frontend behavior is to add a button to show more results for the value collapsed on, which could lead to the query being rerun without collapsing).

3.2.5.3.3 Sending a Refine Query Request

This is a new query request with the same parameters as the original request, except that the new aggregation contains only the aggregations where the results had **maxerror** field set to 1 and

therefore a cut-off had been triggered. In addition, a refine specification MUST be added to the request to specify the aggregation buckets on which to recalculate the number of occurrences, as specified in section 2.2.6.

This query request MUST use a new channel identifier. The protocol client maintains a **SentARefineRequest** state for each outstanding channel identifier so that it can recognize a refine response, because the refine response contains the same message code and structure as a normal response.

3.2.5.3.4 Receiving a Refine Query Response

This is a normal query response message, except that it does not contain information regarding which items matched the query. It also contains no information about field collapsing or sorting. Where a cutoff occurred, the protocol client MUST replace the results from the initial query response with the aggregation data from this response. Consequently, the **AggregationResults** query state contains the exact same number of aggregation results as the original query response, but the occurrence counts for all aggregation buckets are now also correct.

Because this response is structurally the same as a query response (including the message code), the protocol client SHOULD check the **SentARefineRequest** query state to decide whether a query response is a refine response

3.2.5.4 Result Details Request and Response

3.2.5.4.1 Sending a Result Details Request

If the query response contains query hits, then the protocol client assembles a result details request, as specified in section 2.2.8. This is used to retrieve the managed properties that are associated with each item in the result set. The protocol client includes the query stack from the original request so that the protocol server can highlight the original query terms when returning managed properties, because the protocol server has not stored any state information about the original query. If the protocol client does not include the query stack, the protocol server SHOULD NOT perform hit highlighting on the result items, but instead fall back to returning the complete field. The protocol client MUST copy the startup time of the protocol server, as specified in the **LastStartOfServers** state, to the **datestamp** field of the result details request.

If the **query flags** field of the result details request message is present, it MUST be a copy of the **query flags** field from the original query request. If the **include ranking information** feature is enabled in the **query flags** field, then the protocol server MUST re-evaluate the query to generate extra rank calculation debug information. To do this, the protocol server requires enough information to evaluate the query in the same manner as the query request, and the protocol client MUST provide the **ranking** and **current date and time** fields (if freshness boost was calculated) to make this possible. If the protocol server does not support generating rank log information, it MUST generate a dummy value for this in the response. The format and content of the extra rank debug information is implementation specific.

If the protocol supports revisions of search indexes, then the protocol client copies the revision information from the **GenerationTable** field in the query response to the **GenerationTable** field in the result details request (section 2.2.8). This ensures that the item the protocol server returns is the same version the query response found to be a match.

If the protocol client requests a set of managed properties other than the protocol server default, then it MUST specify a summary class in the **wanted summary class** field of the result details request. The available summary classes are specified in the file named "summary.cf" (as specified in

[MS-FSSCFG] section 2.20), which the protocol client retrieves using the protocol specified in [MS-FSCX].

3.2.5.4.2 Receiving Result Details Responses

The protocol server sends one or more result details responses, each of which represents one query hit. The **item content** field MUST be decoded as specified in the summary class in the result details request. The summary.cf and summary.map files store the configuration on the content of each summary class. These are available through the [MS-FSCX] protocol. For more information about these files, see [MS-FSSCFG] sections 2.18 and 2.19.

To further decode datetime and numeric managed properties, the protocol client uses the information stored in the file named "maptransform.xml", specified in [MS-FSSCFG] section 2.3. This file is downloaded from the configuration protocol server using the protocol specified in [MS-FSCX].

Details about how the hit highlighting is encoded can be found in the "fsearch.addon" file (see [MS-FSSCFG] section 2.9), which is also available using the configuration protocol specified in [MS-FSCX].

3.2.5.4.3 Receiving a Multi-part Message End Message

This message specifies that the protocol server sent all responses for the items specified in the associated results details request. If a protocol client did not receive all of the responses, it can send a new request for that specific item. If so, it MUST use a new channel identifier in the new request, because this is a separate request. This separate request is terminated by a new multi-part end message. How the protocol client behaves when it is repeatedly unable to retrieve managed properties for all items is implementation specific.

3.2.5.5 Receiving a Queue Length Message

This message does not use a channel identifier and therefore does not share the same channel identifier as the query request message it responds to. The content of the message MUST be ignored by the client.

3.2.5.6 Statistics Query Request and Response

3.2.5.6.1 Sending a Statistics Query Request

This requests statistics about the state of the protocol server. These statistics include information about cache usage and query latency. If the protocol client wants to merge results from several protocol servers, the protocol server sets the **output command** field to 1 (section 2.2.11), to make the protocol server send the response as an XML fragment. The protocol client SHOULD then merge multiple fragments into one message as specified in [MS-FSSADM] section 7.

3.2.5.6.2 Receiving a Statistics Query Response

If the protocol client has requested information about multiple protocol servers, then the answers from all servers SHOULD be merged into one result. See section 3.2.5.3.1.

3.2.6 Timer Events

The time limit expires if the protocol client waits longer than the timeout for a response from the protocol server specified by the channel identifier. When a timeout occurs, the protocol client

removes the channel identifier from the list of outstanding channel identifiers that are contained in the **SentChannelIds** state.

A protocol client MUST implement a PING request timer, and send a PING request each second when it triggers. A timer MUST be used for each PING request so the protocol client can determine whether the protocol server responds, and update the **ServerAvailable** or **ServersDown** states accordingly.

3.2.7 Other Local Events

None.

3.3 Server Details

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The protocol server does not store any state when processing queries, but it MUST respond to a message with the correct message type and the channel identifier that was contained in the query (section 3.2.1).

Protocol servers impose limitations on results that the protocol client cannot override. For example, the protocol client can request 10,000 query hits, but the protocol server returns only 4000. The protocol server has a default maximum limit of 100000. This limit is configurable.

Index column identifier: A number greater than or equal to 0 that specifies which column the protocol server represents. Configured at install time.

If the protocol server implements the PING response fully, then the protocol server SHOULD also maintain the following states:

Process Count: The total number of search processes on this protocol server.

Active Processes: The active number of search processes on this protocol server

Total Partition Count: The total number of partitions.

Active Partitions: The number of available active partitions.

The number of search processes and partitions is used only by the PING response to specify the health of the implementation. For an implementation that does not expose health details, these states are not used.

3.3.1.1 Handling Multiple Protocol Clients and Multiple Queries per Client

The protocol server processes information from multiple protocol clients. Each protocol client maintains a TCP/IP connection to the protocol server, and the protocol server processes multiple messages over the same connection.

The protocol server responds to a request message by using the channel identifier of the request in the corresponding response message (section 3.2.1). If the protocol server sends an error message

instead, the error message MUST still contain the channel identifier of the query request. The channel identifier does not specify the sequence of operations, nor is it unique to a specific protocol client.

3.3.1.2 Handling PING Requests

The protocol server MUST always respond immediately to a PING request. A PING message does not contain a channel identifier, but a protocol server MUST respond to the PING request on the same TCP/IP connection on which it was received.

3.3.1.3 Search Index

This protocol does not impose limitations on the data structures in which the protocol server stores the content for which the protocol client searches against. The minimal requirements are that it MUST have an index of items that consists of a list of managed properties. The data structure MUST enable searching in specific managed properties and in full-text indexes. In addition, the search index MUST support aggregation on a result set based on the managed properties that are associated with an item, as specified in the aggregation request section 2.2.6.

The full-text indexes and managed properties that are available are specified in the file named "index.cf", see [MS-FSSCFG] section 2.1.2. The managed properties that support aggregations are specified in the file named "indexConfig.xml", see [MS-FSSCFG] section 2.10.

The protocol server MUST handle the state where some of the managed properties do not have a value for every item, and that some of the managed properties have multiple values.

3.3.1.4 Evaluating Queries

The protocol server goes through several states when evaluating a search query, as shown in the following figure.

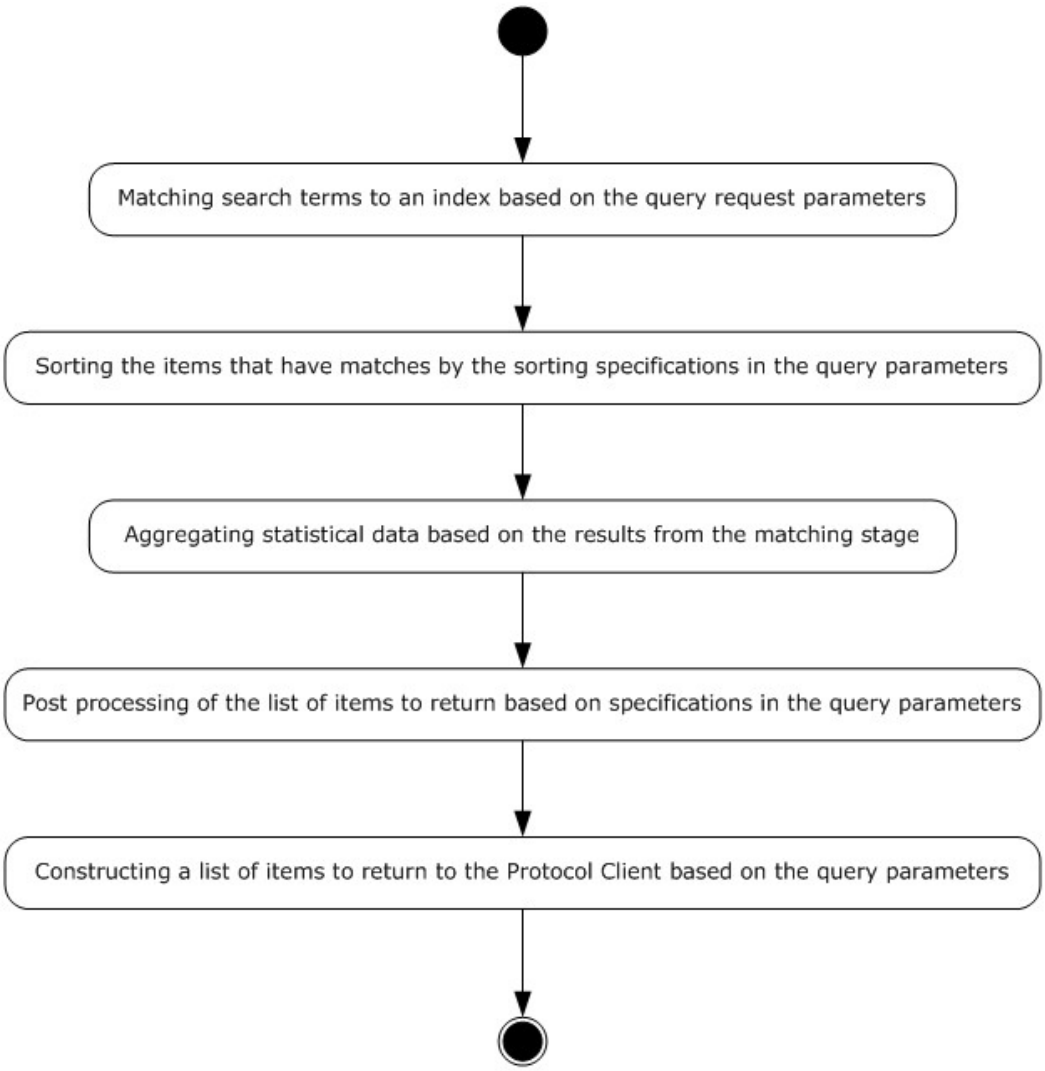


Figure 7: Search query states

The protocol server then returns this list of items and the aggregation results to the protocol client. This is just the list of items, rather than the managed properties associated with an item. The managed properties are returned as a separate stage specified in the following section.

3.3.1.5 Returning Query Hit Details

The protocol server goes through several states when sending a requested item, as shown in the following figure.

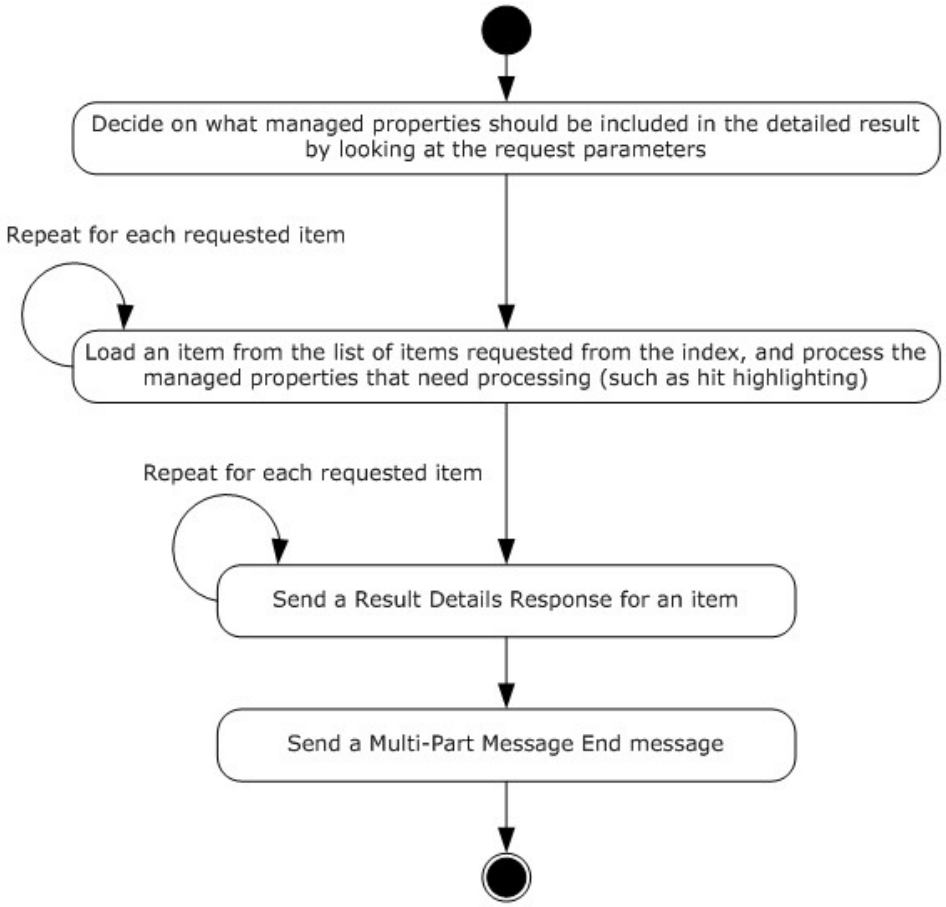


Figure 8: Returning a requested item

When the protocol client requests details for a list of items by sending a result details request message, the protocol server assembles a result details response message containing the managed properties requested. Some of these managed properties are processed before returned to the protocol client, such as performing hit highlighting on the words in the query if the protocol server supports this.

3.3.2 Timers

The protocol server SHOULD implement a configurable timer to stop long running queries that exceed the acceptable search time (default is 12 seconds).

3.3.3 Initialization

If the search node does not contain any items and does not have a search index available, then the protocol server MUST initialize itself so that it responds as if it had a search index containing 0 items.

If a protocol server is part of a redundant system and detects it has a too old search index, it SHOULD update the search index before it opens the TCP/IP port for serving requests if not explicitly configured to ignore this. This avoids dis-synchronization of protocol servers which MUST have the same content. With an out of sync protocol server, result details requests could

permanently fail because of missing documents (or older documents that have been deleted appearing in the result set).

In addition to the files specified in section 3.1.3, the protocol server requires the file named "indexConfig.xml" to determine which managed properties permit aggregation. The protocol server SHOULD also have access to the file named "rank.cf" to specify available rank profiles, and how to combine the rank components when calculating rank. Both of these files are available from the Configuration Server, as specified in [MS-FSCX].

For more information about the file named "indexConfig.xml", see [MS-FSSCFG] section 2.10. For more information about the file named "rank.cf", see [MS-FSSCFG] section 2.14.

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Message Processing Events and Sequencing Rules

The protocol server supports a number of different messages, which are specified in section 2.2. To determine which type of message is received, the protocol server uses the **message code** field, which uniquely identifies the type of message. The codes are specified for each message in section 2.2. If a message arrives with an unknown or non-valid message code, the protocol server behavior is undefined. The messages the protocol server can receive from a protocol client is specified in section 3.1.1.

3.3.5.1 Monitoring the Protocol Connection

3.3.5.1.1 Messages

3.3.5.1.1.1 Sending a PING Request answer

A protocol server responds with a PING request answer when it receives a PING request to confirm that it still is available. Completion of the fields in the PING response is implementation-dependent, as specified in section 2.2.4. The health of the protocol server that uses multiple search processes is specified by the ratio between the **total number of search processes** field and the **active number of search processes** fields.

If an implementation does not use multiple search processes on one protocol server, then both the **total number of search processes** field and the **active number of search processes** field are set to 1 to stop the monitoring process. In this case it will always appear to be running properly as long as it responds to the PING requests, because the **total number of search processes** field contains the same value as the **active number of search processes** field. This is currently used only in statistics/logs.

Similarly, the protocol server components expose the availability of the complete search index by setting the **total partitions** and the **active partitions** fields. The protocol server sets both of these values to 1 to specify that it does not use partitioning and therefore manages the complete search index. The response associated with these settings specifies full availability, even though the protocol servers use index partitioning.

A protocol server MAY stop responding to a PING message to inform a protocol client that it is not available for queries. The current protocol server implementation continues answering, even though the complete index set is not up and running.

3.3.5.1.1.2 Receiving a PING Request

If the protocol server received a PING request, as specified in the **message code** field (section 2.2.3), the protocol server MUST respond with a PING response message (section 2.2.4). It does not do anything with the content of the PING request message.

3.3.5.2 Processing Queries

The protocol server receives a query request that contains parameters that specify query processing. It processes the request, then sends a query response that contains a list of items that match the query.

3.3.5.2.1 Messages

3.3.5.2.1.1 Sending a Query Response

If the protocol server processes the query successfully against the complete search index, then it sends the results to the protocol client in a query response, as specified in section 2.2.7. If the protocol server did not manage to search the complete search index, then the result is only sent if the **allow partial result**s flag is set in the **query flags** field of the request. Otherwise, it sends an error message to specify the reason for the failure if the **enable error message** bit flag is set in the **query flags** field of the request. If the **enable error message** bit flag was not set, then it MUST fail silently.

As long as the query request from the protocol client was not a query to do aggregation refinement (see section 3.2.5.3.3) the protocol server MUST return up to the number of items specified in the **max hits** field of the query request, beginning at the offset specified in the query response's **offset** field. If field collapsing was enabled during query evaluation, a protocol server can end up returning less than the maximum number of items, even though the original query resulted in more items

Aggregation results MUST be encoded as specified for the **aggregationdata** field in section 2.2.7, and they MUST be serialized in the same order as the aggregation requests occurred in the request **aggregation specification** field.

After calculating the list of items to return, a protocol server SHOULD follow the caching instructions in the **user cache lines** field and **max offset** field of the query request if it has a cache, and these features are enabled. These fields specify to the protocol server that a similar search will probably be done shortly.

If the **report search coverage** flag is set in the **query flags** field of the request, the protocol server MUST include search coverage information in the query response, as specified in section 2.2.7.

3.3.5.2.1.2 Receiving a Query Request

The **enabled features** and **query flags** fields in the message specify which features to enable in the query and how to process it.

If an error occurs during message validation or query processing, and the **enable error message** flag of the **query flags** field is set for this request, the protocol server SHOULD send an error message that specifies the failure (section 2.2.5). The protocol server MAY choose not to respond to a query request message instead of sending an error message in some cases, such as if the error happens because of fundamental lack of resources in the search processes.

The query is included as a serialized query stack as the final field of the message payload. The protocol server evaluates the query following the specification for each operator in section 2.2.6. There is no limitation on the number of operators in a query, but the maximum length of the request is 60,000,008 bytes.

For some of the query operators it is possible to specify a specific part of the index against which the operator MUST be evaluated. If no index is specified for the query operators, they MUST all fall back to querying the default index specified in index.cf (see [MS-FSSCFG] section 2.12.3).

The result of the query request is a list of items that match the criteria in the query stack.

The protocol supports revisions of search indexes. If a protocol server supports revisions of search indexes, it SHOULD return a data structure in the **generationtable** field of the query response message (section 2.2.7) which specifies the revision used for the current query.

This value SHOULD then be used by the protocol client to request details from the same index generation (see section 3.3.5.4). This permits returning older versions of items, even if they have changed in the search index. If a protocol server does not support this feature, then it can ignore the index generation value. When the protocol client requests details from a specific revision of the search index (section 3.2.5.4), the protocol server sends the older versions of items if available, even if a newer search index version exists. If the protocol implementation does not support this feature, then it SHOULD ignore the **generation specification** field.

After evaluating the query, the results MUST be sorted according to the sort specifications in the query request. If no sort specification is present, then the default sorting MUST be by rank in descending order. If a sort specification is in place, but no explicit order is given, the protocol server MUST always default to sorting in a descending order. If a sorting specification is given, it MUST be evaluated according to the sort specification section in section 2.2.6, and the protocol server MUST implement support for all sort primitives specified in that section.

The protocol does not impose any restrictions on how to calculate rank, but the protocol server SHOULD implement support for different rank profiles so the protocol client can easily change between different ways of calculating rank.

If a rank profile is specified, and there is no sorting specification present, then the results MUST be sorted according to this rank profile if the protocol server supports rank profiles. The first integer from the **rank profile specification** field SHOULD be used to lookup the rank profile specification in the rank.cf file (see [MS-FSSCFG] section 2.11) if the protocol server wants to calculate a compatible rank.

If both a rank profile and a sort specification are supplied, then the rank profile MUST only be used for sorting if "[rank]" is specified in the sort specification.

A special case of ranking is random ranking. The protocol supports two types of random ranking, simple and full. Simple random search is performed by using only the **random seed** field from the request, while full random sorting is specified using the **sort specification** field. Simple random search re-sorts only the highest ranked items up to the maximum number of returned items for a query to limit resource usage. If the search index is reorganized, a simple random search might not return the same items even if no items have changed. A full random ranking, on the other hand, processes the complete result set, and computes the same result for a specified seed value, even if the search index was reorganized. However, the full random ranking can change if the number of available items changes.

If a value is specified for the **current date and time** field, then the protocol server uses it as the current time when calculating the freshness boost, as specified in section 2.2.6.

If the protocol client requests an aggregation, then the aggregation specifications MUST be evaluated over the complete result set of the query according to the specifications in section 2.2.6. If aggregation bucket cut-off parameters are specified in the aggregation specifications (for performance reasons), then the protocol server MUST limit the number of aggregated buckets returned accordingly. If the protocol server actually performs a cut-off then the protocol server MUST set the **Maxerror** flag in the **AggrElement** information element for this aggregation in the query response message to specify this. How the cutoff is done depends on the implementation, and for some implementations this can lead to an incorrect occurrence count in the aggregation buckets returned. For an implementation where the bucket count will never be wrong, even when a cut-off has occurred, then the **maxerror** SHOULD NOT be set.

After the protocol server sends the result, and the protocol client determines that a cut off has occurred, then the protocol client SHOULD send an additional query request message with an aggregation refine specification for these aggregation buckets to verify that the occurrence count is correct as specified in section 2.2.6. The protocol server MUST then recalculate the occurrence values ,for these aggregation buckets and return the new values in a query response message. The performance impact is limited because only the specified aggregation buckets are counted. Because the refine does not have any cut-off specifications, cut-offs MUST NOT take effect (and **maxerror** MUST never be set). The refine query response contains only aggregation data.

A protocol server implements request evaluation so that the sorting and collapsing processes do not occur for refine requests, because the protocol server does not send query hits to the protocol client. It only needs to generate the result set of the query.

After the result set is sorted, an additional field collapse and re-sort can be performed. Field collapsing permits folding of results where the items contain the same value for a specified managed property. This can be used to avoid too many similar query hits, or to remove duplicates. The managed property to collapse on is specified in the **Field Collapsing with Collapsed Results Removed** field, and the number of items to keep per unique managed property is specified in the **Field collapsing** field of the query request. Field collapsing is only performed when both fields are specified. When the protocol server performs field collapsing, it reorders the result set so that multiple items that contain the managed property value it is collapsing on are grouped together sequentially in the result set. The order of the entries in a collapsed group is the sequence specified in the rules used to sort the complete result set. If more than the number of items to keep per unique managed property exists in the result set, the protocol server removes these from the result set.

An example would be to field collapse on a managed property containing the length of a domain name. If the number of items to keep per unique domain length is specified as 3, the protocol server would keep up to 3 items per unique domain length. The up to 3 items would appear sequentially in the result list beginning at the place the highest ranked (or sorted) item appears.

If the **report queue length** bit flag is set in the **query flags** field of the request, the protocol server sends a queue length message (section 2.2.10) before sending the query response.

3.3.5.3 Returning Results

3.3.5.3.1 Messages

3.3.5.3.1.1 Sending a Result Details Response

All result details responses use the same channel identifier that was used for the initial request, as specified in section 3.2.1. The protocol server SHOULD send an error message if an error occurs during request processing and the **enable error message** flag in the **query flags** field in the message is set. The protocol server MUST NOT send any result details responses or multi-part

message end messages with the same channel identifier after an error message has been sent (until the protocol client reuses the value in a new message sequence).

If the **WantedResClass** bit flag is set in the **enabled features** field of the request, then the protocol server MUST use the **Wanted Summary class** field to determine which managed properties to return. This is done by looking up which managed properties a summary class consists of in the "summary.cf" file, as specified in [MS-FSSCFG] section 2.18. If the **WantedResClass** bit flag is not set then the protocol server MUST use the default summary class specified in the file. The protocol server does not return managed properties other than the ones configured in the summary class.

The protocol server SHOULD use the query stack to generate an appropriate hit highlighting of search terms if this is supported by the protocol server and the managed properties are configured to use the hit highlighting feature. This configuration is stored in the "summary.map" file. Configuration on how to do hit highlighting is in the "fsearch.addon" file. The protocol server implementation MAY use origin values to comment operators in such a way that hit highlighting can treat them differently if configured to do so. If the query stack is not supplied in the request message, then the protocol server SHOULD fall back to returning the managed property without hit highlighting, because it is not possible to highlight without knowing the original query terms. If the **include ranking information** flag is set in the **query flags** field, then the protocol server SHOULD re-evaluate the query to generate debug ranking information. The debug information SHOULD be returned in the ranklog predefined summary field if the summary class includes it. Interpretation of this rank log is implementation specific. The query is re-evaluated because the protocol server does not save information from the original request. Re-evaluating the query does not change the items that the protocol server sends to the protocol client in the **docids** field of the response. The protocol server also uses other fields in the result details request, such as the **rank profile** and **current date and time** fields, to re-evaluate the query (if needed).

For more information about the file named "summary.map", see [MS-FSSCFG] section 2.19. For more information about the file named "fsearch.addon", see [MS-FSSCFG] section 2.9. For more information about origin values, see the operator origin field in section 2.2.6.

3.3.5.3.1.2 Receiving a Result Details Request

When the protocol server receives a result details request, it returns one result details response message per item requested (section 2.2.9). When all the requested items are returned, the protocol server sends a multi-part message end response (section 2.2.2) to confirm completion. If the protocol server supports revisions of search indexes, then it uses the **GenerationTable** field in the request when it loads the content of the managed properties from the search index. If the requested revision is no longer available, the protocol server sends an error message with the error code 16 if the protocol client requested error messages.

If a protocol server does not support revisions of search indexes, the **GenerationPresent** bit flag and data field in the request can be ignored.

The protocol server verifies that the **datestamp** field of the request is equal to the startup time of the **timestamp** field of the PING response. If it is not equal, it returns an error message that contains error code 20 (section 2.2.5).

3.3.5.4 Returning statistics

3.3.5.4.1 Messages

3.3.5.4.1.1 Sending a Statistics Query Response

When the protocol server replies to a statistics query request, it MUST answer with the same channel identifier as in the request. It MUST also follow the output command specifications in the statistics query request (see section 2.2.11) to decide how to format the payload.

3.3.5.4.1.2 Receiving a Statistics Query Request

When the protocol server receives a statistics query request, it MUST return a statistics query response (section 2.2.12).

3.3.6 Timer Events

If a timer is implemented to stop long running queries, then an event SHOULD be triggered once it goes beyond this limit. The protocol server SHOULD then abort the long-running query to limit the amount of resources used. This SHOULD be a configurable limit, defaulting to 12 seconds.

3.3.7 Other Local Events

None.

4 Protocol Examples

For all examples, the first column describes the data on the wire in hexadecimal notation, while the second column describes the hexadecimal values.

4.1 Full Query/Result

The example in this section searches for items where the string "w03" is present in a managed property named "string1", and the index named "meta.collection" contains the value "navtest". Single-level sorting and aggregation are requested, based on managed property string1. Aggregation is enabled, as described in section 4.2.1.5 except that cutoff is not requested in this example.

4.1.1 Query Request

The request follows the specification in section 2.2.6.

Required fields:	
00 00 00 F8	Message data length: 248 byte
00 00 00 DA	Message code 218 – query request.
00 00 00 1E	Channel identifier
00 00 29 86	Features enabled (**Sort specification**, **Rank profile specification**, **Aggregation specification**, **Generation specification**, **Field Collapsing** and **Parsed Query** bit flags are set)
00 00 00 00	Query type is unknown or it contains multiple operators joined by an **AND** operator
00 00 00 00	Requested offset. A value of 0 specifies the beginning of the result set.
00 00 00 0A	Maximum number of query hits to return
00 08 80 0C	Query flags (**Allow error message**, **top level search**, **report queue length**, and **report search coverage** fields are set)
Optional fields as described in the enabled features field	
00 00 00 08	Generation table length = 8
00 00 01 00 00 00 00 00	Generation table.
00 00 00 00	Rank profile to use.
00 00	Dataset. Always set to 0.

Required fields:	
00 00	
00 00 00 01	Field collapsing. Specifies the number of items to keep per collapsed item. The collapse field specification is not present, so collapsing does not occur.
00 00 00 0E	Specifies the length of the Sort Specification field. In this example, the length is 14 bytes
2D 62 61 74 76 6E 75 6D 65 72 69 63 33 20	Sort specification field that contains "-batvnumeric3 ".
00 00 00 6A	Specifies the length of the Aggregation specification field. In this example, the specification is 106 bytes
28 68 69 73 74 20 3A 62 75 63 6B 65 74 73 20 3A 75 6E 69 71 75 65 20 3A 63 75 74 6D 61 78 62 75 63 6B 65 74 73 20 31 30 30 30 20 62 61 76 6E 73 74 72 69 6E 67 31 29 28 63 6F 75 6E 74 20 62 61 76 6E 73 74	Contains the string "(hist :buckets :unique :cutmaxbuckets 1000 bavnstring1)(count bavnstring1)(hitcount)".

Required fields:	
72 69 6E 67 31 29 28 63 6F 75 6E 74 6E 7A 20 62 61 76 6E 73 74 72 69 6E 67 31 29 28 68 69 74 63 6F 75 6E 74 20 29	
00 00 00 03	Specifies that there is a parsed query and it has approximately 3 operators.
00 00 00 01 00 00 00 02	**AND** operator with an arity of 2.
00 00 00 04 00 00 00 0F 6D 65 74 61 2E 63 6F 6C 6C 65 63 74 69 6F 6E 00 00 00 07 6E 61 76 74 65 73 74	String term. Because the integer that occurs immediately after the operator is greater than zero, an index specification occurs after the length of that integer. In this case the length is 15. The name of the index is "meta.collection" (see [MS-FSFIXML] section 2.5.2.1.3 for more about this specific index). Then follows the string term, whose length is 7 and that contains "navtest".
00 00 00 04 00 00 00 07 73 74 72 69 6E 67	Second string term. The name of the index is "string1". String to search for in this managed property: "w03"

Required fields:	
31 00 00 00 03 77 30 33	

4.1.2 Query Response

The response specifies that only 3 items match the query.

Required Fields	
00 00 01 00	Size 256
00 00 00 D9	Message code 217.
00 00 00 1E	Channel identifier
00 00 00 F1	**Enabled Feature** bit flags (**Dummy**, **SortDataPresent**, **AggregationDataPresent**, **CoveragePresent** and **GenerationPresent** bit flags)
00 00 00 00	Offset is 0
00 00 00 03	**NumHits** field contains the value 3
00 00 00 03	**TotalHits** field contains the value 3
00 00 00 00	**Maxrank** field is set to 0, so ranking was not performed
00 00 00 00	Docstamp
Optional fields	
00 00 00 08	Generation table length = 8
00 00 00 01 00 00 00 02	Generation Data
Sort specification	
00 00 00 08	SortIndex[1]=8, offset to the beginning of item 2
00 00 00 10	SortIndex[2]=16, offset to the beginning of item 3
00 00 00 18	SortIndex[3]=24, points to end of sort data blob.
3F EB FF FF FF FF FF FF	**SortData** byte array for item 1
3F F7 FF FF FF FF FF FF	**SortData** byte array for item 2
C0 24 00 00 00 00 00 00	**SortData** byte array for item 3

Required Fields	
Aggregation data	
00 00 00 6C	Aggregation data length = 108
01 00 00 01 20 03 10 08 00 00 00 00 03 00 00 00 28 03 14 0A 00 00 00 00 09 00 00 00 00 00 00 00 30 03 10 08 00 00 00 00 03 00 00 00 47 03 04 02 00 00 00 00 00 00 00 00 04 00 00 00 2C 00 00 00 03 00 00 00 77 30 31 02 00 00 00 03 00 00 00 77 30 32 03 00 00 00 03 00 00 00 77 30 33 03 00 00 00 03 00 00 00 77 30 34 01 00 00 00	Aggregation data blob
Partial result information	
00 00 00 00 00 00 15 6B	Internal
00 00 00 02	Nodes
00 00 00 01	**Fullresult** field set to 1 specifies that the response is a complete response.
Query hit list	
00 00 00 04 00 00 00 00 00 00 00 00 49 14 28 2C	Hit 1
00 00 00 03 00 00 00 00 00 00 00 00 49 14 28 2C	Hit 2
00 00 00 21 00 00 00 00 00 00 00 00 49 14 28 2C	Hit 3

4.1.3 Result Details Request Message

The protocol client uses the items from the response to retrieve the item summaries.

Data	Description
00 00 00 88	Message data length: 136 byte
00 00 00 DB	Message code 219 (Result Details)
00 00 00 24	Channel identifier
00 00 00 85	Features bit flags (**Datestamp**, **query stack** and **Generation Specification**).
49 2A B9 0E	Docstamp (number of seconds that elapsed after 1970-01-01 UTC).
00 00 00 08	Length of Generation table.
00 00 00 00 00 00 00 00	GenerationTable field.
00 00 00 03	For debug, the query stack is included because the **query stack** field is set.
00 00 00 40	Length of query stack (64 bytes)
00 00 00 01 00 00 00 02 00 00 00 04	Debug Query stack for internal use. This is a copy of the query stack in the original request.

Data	Description
00 00 00 0F 6D 65 74 61 2E 63 6F 6C 6C 65 63 74 69 6F 6E 00 00 00 07 6E 61 76 74 65 73 74 00 00 00 04 00 00 00 07 73 74 72 69 6E 67 31 00 00 00 03 77 30 33	
00 00 00 04 00 00 00 00 49 14 28 2C	Request for item 1
00 00 00 03 00 00 00 00 49 14 28 2C	Request for item 2
00 00 00 21 00 00 00 00 49 14 28 2C	Request for item 3

4.1.4 Result Details Response

The response returns one message for each result item requested. The following tables show the detailed message content for this example. When decoding this message, the summary.cf file is used to decide which type and which fields are present.

Message 1 (query hit 1 in the sorted result)	
00 00 01 2C	Message length is 300
00 00 00 CD	Message code is 205
00 00 00 24	Channel identifier
00 00 00 04	DocId
FF FF FF 3F	Summaryclass = 1073741823
Field Content	
28 00	Length of field
62 39 37 34 39 31 61 33 35 63 30 30	Field content: b97491a35c0094022e1624b041c473a1_navtest

Message 1 (query hit 1 in the sorted result)	
39 34 30 32 32 65 31 36 32 34 62 30 34 31 63 34 37 33 61 31 5F 6E 61 76 74 65 73 74	
04 00	Length of field
69 64 39 34	Field content: "id04"
07 00	Length of field
6E 61 76 74 65 73 74	Field content: "navtest"
00 00 00 00	<empty property of type **longstring**>
00 00 00 00	<empty property of type **longstring**>
00 00 00 00	<empty property of type **longstring**>
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	<8 empty properties of type **string**>
04 00	Length of field
69 64 30 34	Field content: "id04"
00 00	<empty property>
03 00	Length of next field
34 37 31	"471"
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	<8 empty properties of type **string**>
07 00	Length of next field
75 6E 6B 6E 6F 77 6E	"unknown"
07 00	Length of next field
75 6E 6B 6E 6F 77 6E	"unknown"
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	<22 empty properties of type **string**>
0F 00	Length of next field
77 30 31 3B 77 30 32 3B 77 30 33 3B 77 30 34	"w01;w02;w03;w04"
13 00	Length of next field

Message 1 (query hit 1 in the sorted result)	
77 30 31 3B 77 30 32 3B 77 30 33 3B 77 30 34 3B 77 30 35	"w01;w02;w03;w04;w05"
17 00	Length of next field
77 30 31 3B 77 30 32 3B 77 30 33 3B 77 30 34 3B 77 30 35 3B 77 30 36	"w01;w02;w03;w04;w05;w06"
01 00	Length of next field
35	"5"
01 00	Length of next field
35	"5"
01 00	Length of next field
35	"5"
00 00 00 00	<empty property of type **longstring**>
00 00 00 00	<empty property of type **longstring**>
00	<empty property>

The next two query hits contains content whose format is the same as query hit 1, and therefore are not specified in this example.

Message 2 (query hit 2 in the sorted result)	
00 00 01 20	Message length is 288
00 00 00 CD	Message code is 205
00 00 01 14	Channel identifier
00 00 00 03	DocId
FF FF FF 3F	Summaryclass = 1073741823

Message 3 (query hit 3 in the sorted result)	
00 00 01 00	Message length is 256
00 00 00 CD	Message code is 205
00 00 01 14	Channel identifier
00 00 00 21	DocId
FF FF FF 3F	Summaryclass = 1073741823

The following table is the multi-part end message that specifies that the protocol server has sent all result details responses.

Message 4	
00 00 00 08	Message length = 8
00 00 00 C8	Message type = 200
00 00 01 14	Channel identifier

4.2 Detailed Query

4.2.1 Aggregation Examples

The following examples shows query requests for aggregation and the associated responses.

4.2.1.1 Basic Numeric Data Aggregation

The protocol client requests aggregation on a signed 32-bit integer managed property named "numeric1", and an aggregation bucket width of 1. This implies that the protocol server returns occurrence data for all values (there is a bucket per result value). The aggregator is named "bavnnumeric1", and was associated with the "numeric1" managed property in the file named index.cf. The request from the client is as follows.

(max bavnnumeric1)(min bavnnumeric1)(sum bavnnumeric1)(count bavnnumeric1)(countnz bavnnumeric1)(hitcount)(hist :width 1 bavnnumeric1)

The following table describes the query response, beginning with the **AggregationData** field.

Signature	Value	Data type
00 00 28 14 00 00 00 00 00 10 00 00	max=4096	int32_l
08 00 28 14 00 00 00 00 01 00 00 00	min=1	int32_l
10 00 2C 14 00 00 00 00 1C 20 00 00 00 00 00 00	sum=8220	int64_l
20 03 10 08 00 00 00 00 1F 00 00 00	hitcount=31	uint32_l
28 03 14 0A 00 00 00 00 10 00 00 00 00 00 00 00	count=16	uint64_l
30 03 10 08 00 00 00 00 10 00 00 00	countnz=16	uint32_l

The **hist** aggregation type occurs next in the response, and it is followed by the histogram buckets.

Signature	Value	Data type
4B 03 28 14 00 00 00 00 10 00 00 00	AggType = 105. 16 rows	int32_l

The aggregation bucket results occur next in the response.

Data on the wire	Value	Number of occurrences in the aggregation bucket
01 00 00 00 01 00 00 00	Numeric=1	Bucket count=1
02 00 00 00 01 00 00 00	Numeric=2	Bucket count=1
03 00 00 00 01 00 00 00	Numeric=3	Bucket count=1

Data on the wire	Value	Number of occurrences in the aggregation bucket
05 00 00 00 01 00 00 00	Numeric=5	Bucket count=1
0A 00 00 00 01 00 00 00	Numeric=10	Bucket count=1
0B 00 00 00 01 00 00 00	Numeric=11	Bucket count=1
0C 00 00 00 01 00 00 00	Numeric=12	Bucket count=1
10 00 00 00 01 00 00 00	Numeric=16	Bucket count=1
20 00 00 00 01 00 00 00	Numeric=32	Bucket count=1
40 00 00 00 01 00 00 00	Numeric=64	Bucket count=1
80 00 00 00 01 00 00 00	Numeric=128	Bucket count=1
00 01 00 00 01 00 00 00	Numeric=256	Bucket count=1
00 02 00 00 01 00 00 00	Numeric=512	Bucket count=1
00 04 00 00 01 00 00 00	Numeric=1024	Bucket count=1
00 08 00 00 01 00 00 00	Numeric=2048	Bucket count=1
00 10 00 00 01 00 00 00	Numeric=4096	Bucket count=1

4.2.1.2 Numeric Data Aggregation with Predefined-Width Aggregation Buckets

The protocol client requests aggregation on a signed 32-bit integer managed property named "numeric1". It also requests fixed width aggregation buckets, with value ranges that are described as follows.

- Value < 5
- 5 <= Value < 10
- 10 <= Value < 15
- 15 <= Value

The aggregator is named "bavnnumeric13". The protocol client request for aggregation is shown in the following example.

(max bavnnumeric13)(min bavnnumeric13)(sum bavnnumeric13)(count bavnnumeric13)(countnz bavnnumeric13)(hitcount)(hist :buckets '(5 10 15) bavnnumeric13)

The following table specifies the aggregation data part of the query response.

Signature	Value	Data type
00 00 28 14 00 00 00 00 00 10 00 00	max=4096	int32_l
08 00 28 14 00 00 00 00 01 00 00 00	min=1	int32_l
10 00 2C 14 00 00 00 00 1C 20 00 00 00 00 00 00	sum=8220	int64_l

Signature	Value	Data type
20 03 10 08 00 00 00 00 1F 00 00 00	hitcount=31	uint32_l
28 03 14 0A 00 00 00 00 10 00 00 00 00 00 00 00	count=16	uint64_l
30 03 10 08 00 00 00 00 10 00 00 00	countnz=16	uint32_l
Signature	Histogram aggregation type	Number of buckets
3B 03 10 14 00 00 00 00 04 00 00 00	AggType=103	4
Data on the wire	Bucket index	Bucket occurrences
00 00 00 00 03 00 00 00	0 (maps to "Value < 5")	3
01 00 00 00 01 00 00 00	1 (maps to "5 <= Value < 10")	1
02 00 00 00 03 00 00 00	2 (maps to "10 <= Value < 15")	3
03 00 00 00 09 00 00 00	3 (maps to "15 <= Value")	9

4.2.1.3 Numeric Data Aggregation with Aggregation Bucket

4.2.1.4 Aggregation over One Numeric and One String Managed Property

The protocol client requests aggregation on a string managed property named "string1" and a signed 32-bit integer managed property named "numeric1". The string aggregator is named "bavnstring1" and the numeric aggregator is named "bavnnumeric1". The protocol server will return information for no more than 1000 unique strings. An aggregation bucket width of 1 is requested for the numeric aggregation. The protocol client request for aggregation is described as follows.

(hist :buckets :unique :cutmaxbuckets 1000 bavnstring1)

(count bavnstring1)(countnz bavnstring1)(hitcount)

(max bavnnumeric1)(min bavnnumeric1)(sum bavnnumeric1)(count bavnnumeric1)

(countnz bavnnumeric1)(hitcount)(hist :width 1 bavnnumeric1)

This is the response to the client request. The following table describes the **AggregationDat**a field. Data on the wire is shown in hexadecimal values in the first column, and decoded attributes are shown in the following columns.

Data on the wire	Value	Navigator
00 00 28 14 00 00 00 00 05 00 00 00	max=5	numeric1
08 00 28 14 00 00 00 00 F6 FF FF FF	min=-10	numeric1
10 00 2C 14 00 00 00 00 FE FF FF FF FF FF FF FF	sum=-2	numeric1
20 03 10 08 00 00 00 00 03 00 00 00	hitcount=3	string1
20 03 10 08 00 00 00 00 03 00 00 00	hitcount=3	numeric1

Data on the wire	Value	Navigator
28 03 14 0A 00 00 00 00 09 00 00 00 00 00 00 00	count=9	numeric1
28 03 14 0A 00 00 00 00 03 00 00 00 00 00 00 00	count=3	string1
30 03 10 08 00 00 00 00 03 00 00 00	countnz=3	numeric1
30 03 10 08 00 00 00 00 03 00 00 00	countnz=3	string1

The following data describes the histogram returned for the string aggregator, with aggregator type 104, 4 aggregation buckets and a payload of 44 bytes. In this example all strings have a length of 3 bytes.

Data on the wire	Max Error	Aggregation buckets
47 03 04 02 00 00 00 00 00 00 00 00 04 00 00 00 2C 00 00 00	0 (so there is no refine process)	4 buckets, 44 bytes in size.

Then, the data for each of the string buckets occurs next in the response.

Data on the wire	String	Occurrence count
03 00 00 00 77 30 31 02 00 00 00	"w01"	2
03 00 00 00 77 30 32 03 00 00 00	"w02"	3
03 00 00 00 77 30 33 03 00 00 00	"w03"	3
03 00 00 00 77 30 34 01 00 00 00	"w04"	1

The following data specifies the histogram returned for the numeric aggregator, with 3 aggregation buckets.

Data on the wire	Histogram aggregation type	Number of buckets
4B 03 28 14 5C C2 CA 39 03 00 00 00	AggType=105	3

The data for each of the numeric buckets occurs next.

Data on the wire	Numeric1 value	Occurrence count
F6 FF FF FF 01 00 00 00	-10	1
03 00 00 00 01 00 00 00	3	1
05 00 00 00 01 00 00 00	5	1

4.2.1.5 Aggregation with Aggregation Bucket Refine

In this example, the protocol client requests refinement of an aggregation.

The protocol client first requests an aggregation of the managed property "string1". The string aggregator is named "bavnstring1", and the protocol client requests a cut-off by aggregation bucket frequency, where it only returns buckets with more than 2 values. When a cut-off occurs,

occurrence count values for a bucket can be wrong. The protocol client then sends a new for refinement on the aggregation result to recount the occurrence values so that they are correct.

(hist :buckets :unique :cutfreq 2 bavnstring1)(count bavnstring1)(countnz bavnstring1)(hitcount)

This is the response to the client request. The following table describes the **AggregationData** field. Data on the wire is shown in hexadecimal values in the first column, and decoded attributes are shown in the following columns.

Data on the wire	Value	Navigator
20 03 10 08 00 00 00 00 1F 00 00 00	hitcount=31	string1
28 03 14 0A 00 00 00 00 3C 00 00 00 00 00 00 00	count=60	string1
30 03 10 08 00 00 00 00 1F 00 00 00	countnz=31	string1

The following data specifies the histogram returned, with 2 aggregation buckets. The maxerror field is set, so the protocol client is informed that a cut-off occurred.

Data on the wire	Maxerror field	Aggregation bucket
47 03 04 02 00 00 00 00 02 00 00 00 02 00 00 00 16 00 00 00	The maxerror field is set to 2	2 buckets, 22 bytes in all.

The data for each of the string buckets occurs next.

Data on the wire	String length	String1 value	Occurrence count
03 00 00 00 77 30 31 1F 00 00 00	3	"w01"	31
03 00 00 00 77 30 32 1A 00 00 00	3	"w02"	26

The protocol client reuses the previous request to refine the aggregation on the values that can have wrong occurrence counts.

(hist :buckets :unique :cutfreq 2 bavnstring1)

(count bavnstring1)(countnz bavnstring1)(hitcount)

(refine bavnstring1 2 3'w01 3'w02)

The refine operator requests refinement on 2 string aggregation buckets. The expression "3'w01" specifies a string length of 3 whose managed property value is associated with the aggregation bucket.

This is the response to the client refine request. The following table describes the **AggregationData** field. Data on the wire is shown in hexadecimal values in the first column, and decoded attributes are shown in the following column. The response contains only the refined aggregation data and not any query hits.

Data on the wire	Aggregation bucket count
52 03 10 08 00 00 00 00 02 00 00 00	2

The data for each of the refinement buckets occurs next in the response.

Data on the wire	Bucket number	Recalculated occurrences
1F 00 00 00	1	31
1A 00 00 00	1	26

The aggregation bucket count is returned in the same sequence as in the refinement aggregation request. In this case the occurrence was correct even before the refinement, but this was not guaranteed because **maxerror** was set.

4.2.2 Count Operator

This example describes the **COUNT** operator, which differs from other query requests in that it specifies the part of the managed property against which the query processes. The parsed query consists of a **COUNT** operator and an **IN** operator, and operates against the "title" managed property of the search index. The protocol server returns any items where the term "cnn" occurs 3 times or more, but less than 5 times. The protocol client request is as follows.

Required fields	
00 00 00 68	Size 104 bytes
00 00 00 DA	Message code: 218 (Query request)
00 00 00 58	Channel identifier
00 00 28 06	Features enabled (**rank profile specification**, **generation specification**, **field collapsing** and **parsed query** bit flags are set)
00 00 00 00	Query type 0 - parsed query
00 00 00 00	Requested Offset. A value of 0 specifies the beginning of the result set.
00 00 00 0A	Maximum number of query hits to return (10).
00 08 80 0C	Query flags field (**Allow Error Message**, **top level search**, **report queue length**, and **report search coverage** bit flags)
Variable payload fields (as specified in the **enabled features** flags)	
00 00 00 08	Generation table length = 8
00 00 00 01 00 00 00 00	Generation table.
00 00 00 00 00 00 00 00	Rank profile to use.

Required fields	
00 00 00 01	Describes the number of items to keep per collapsed field. The collapse field specification is not present, so collapsing does not occur.
Parsed Query	
00 00 00 05	Parsed query, 5 operators.
00 00 00 0E	**IN** operator
00 00 00 02	The **IN** operator has an arity of 2.
00 00 00 10	Complete internal property region operator. The **IN** operator thus covers the whole internal property in the **COUNT** case.
00 00 00 12	**COUNT** operator (18)
00 00 00 02	Minimum number of occurrences
00 00 00 05	Maximum number of occurrences
00 00 00 10	Complete internal property region operator. The **COUNT** operator covers the whole internal property.
00 00 00 04	A string term
00 00 00 05	The length of the field name is 5
74 69 74 6C 65	The managed property name is "title"
00 00 00 03	The string term has a length of 3 bytes.
63 6E 6E	The string term is "cnn"

4.2.3 Internal Property Region Search

This example shows a query that uses an internal property region search on parts of a managed property. In the managed property "xml" it searches for items which contain the string "Title-text" in the internal property region named "title". The search uses the **RANK** operator to increase the rank for query hits that were also found elsewhere in the managed property.

Required fields	
00 00 01 30	Size 304
00 00 00 DA	Message code: 218 - Query request.
00 00 00 7A	Channel identifier

Required fields	
00 00 28 06	Features enabled (**Rank Profile Specification**, **Generation Specification**, **Field Collapsing**, and **Parsed Query** bit flags)
00 00 00 00	Query type. A value of 0 specifies that this is a parsed query
00 00 00 00	Requested Offset. A value of 0 specifies the beginning of the result set.
00 00 00 0A	Maximum number of query hits to return
00 08 80 0C	Query flags (**Allow error message**, **top level search**, **report queue length**, and **report search coverage** bit flags)
Optional fields	
00 00 00 08	Generation table length = 8
00 00 00 01 00 00 00 00	Generation table.
00 00 00 00 00 00 00 00	Rank profile to use.
00 00 00 01	Describes the number of items to keep per collapsed field. The collapse field specification is not present, so collapsing does not occur
00 00 00 0D	Indicates 13 operators, although there are only 11.
00 00 00 03	**RANK** operator
00 00 00 02	Arity 2
00 00 00 00	Always set to 0.
00 00 00 0E	Contained in the associated **Internal property region** field
00 00 00 02	Arity of 2.
00 00 00 0F	Internal property region
00 00 00 0F	Length of index name
5B 73 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search.
00 00 00 07	Length of internal property region specification
6D 65 73 73 61 67 65	Internal property region: "message"
00 00 00 0E	**IN** operator
00 00 00 02	Arity of 2
01 00 00 0F	Internal property region operator. The bitmask for this internal property region field specifies that this is the internal property region to return.
00 00 00 0F	Length of the name of the index
5B 73 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search

Required fields	
00 00 00 05	Length of internal property region specification
74 69 74 6C 65	Internal property region: "title"
00 00 00 0D	Ordered **NEAR**
00 00 00 02	Arity of 2
00 00 00 00	Distance=0
00 00 00 04	String Term
00 00 00 0F	Index name length
5B 63 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search
00 00 00 06	Length of string term
74 69 74 6C 65 54	"titleT" (query string)
00 00 00 04	String term
00 00 00 0F	Index name length
5B 63 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search
00 00 00 05	Length of string term
74 65 78 74 54	"textT" (query string)
00 00 00 0D	Ordered **NEAR**
00 00 00 02	Arity of 2
00 00 00 00	Distance=0
00 00 00 04	String term
00 00 00 0F	Index name length
5B 63 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search
00 00 00 06	Length of string term
74 69 74 6C 65 54	"titleT" (query string)
00 00 00 04	String term
00 00 00 0F	Index name length
5B 63 5D 5F 62 73 63 70 78 6D 6C 2E 61 6C 6C	Name of the index to search
00 00 00 05	String term length
74 65 78 74 54	"textT" (query string)

4.3 PING

4.3.1 Ping Request

On the wire	Description
00 00 00 04	Size of remainder of message is 4 bytes
00 00 00 CE	Message code: 206 Ping Request Message

4.3.2 Ping Request Answer

On the wire	Description
00 00 00 1C	Size of message is 28 bytes excluding this field
00 00 00 D2	Message code: 210, Ping response
00 00 00 01	Partition identifier: 1
49 76 88 B8	Time stamp: The time the protocol server started, represented by the number of seconds that elapsed after 1970-01-01 (UTC)
00 00 00 03	Total number of search processes: 3
00 00 00 02	Active number of search processes: 2
00 00 00 03	Total partitions: 3
00 00 00 02	Active partitions: 2

4.4 Error

4.4.1 Single Error

This example describes a single error from the protocol server. The channel identifier in the error message is the same as the one in the query that failed.

On the wire	Description
00 00 00 43	Size of message data is 67 bytes
00 00 00 CB	Message code 203 - Error message.
00 00 00 02	Channel identifier
00 00 00 08	Error code 8 - Lost connection to sub node

On the wire	Description
00 00 00 1B	Error message length (27 bytes)
4C 6F 73 74 20 63 6F 6E 6E 65 63 74 69 6F 6E 20 74 6F 20 73 75 62 2D 6E 6F 64 65	"Lost connection to sub-node"

4.4.2 Multiple Errors

This example describes the special error message that the protocol server sends when multiple errors occur. This special message encapsulates all of the error messages that occurred. The error code for the second error is specified as a UTF-8 encoded string, rather than a numeric field as for the first error code.

On the wire	Description
00 00 00 5E	Size of message data is 94 bytes
00 00 00 CB	Message code 203 - Error message
00 00 00 0A	Channel identifier
00 00 00 09	Error code 9 - Multiple errors occurred
00 00 00 4E	Error message length (78 bytes)
4E 6F 20 65 6E 67 69 6E 65 20 61 76 61 69 6C 61 62 6C 65 20 66 6F 72 20 70 61 72 74 69 74 69 6F 6E 20 30 0A	The first error message, followed by a newline(\n). The string is "No engine available for partition 0" (36 bytes).
32	The second error code (50 is the UTF-8 character value of "2", which is "Error Parsing Query", error code 2).
27 75 6E 6B 6E 6F 77 6E 20 66 75 6E 63 74 69 6F 6E 20 6E 61 6D 65 20 61 74 20 69 6C 6C 65 67 61 6C 20 76 61 6C 75 65 29 27	The second error string: "'unknown function name at illegal value'", which has a length of 41 bytes.

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 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.

7 Change Tracking

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

8 Index

S

T

V