[MS-SQP]:

MSSearch Query Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are 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 as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does 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 documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability
6/27/2008	1.0	Major	Revised and edited the technical content
12/12/2008	1.01	Editorial	Revised and edited the technical content
7/13/2009	1.02	Major	Revised and edited the technical content
8/28/2009	1.03	Editorial	Revised and edited the technical content
11/6/2009	1.04	Editorial	Revised and edited the technical content
2/19/2010	2.0	Editorial	Revised and edited the technical content
3/31/2010	2.01	Editorial	Revised and edited the technical content
4/30/2010	2.02	Editorial	Revised and edited the technical content
6/7/2010	2.03	Editorial	Revised and edited the technical content
6/29/2010	2.04	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.05	Minor	Clarified the meaning of the technical content.
9/27/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	2.06	Editorial	Changed language and formatting in the technical content.
3/18/2011	2.06	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	2.06	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	2.7	Minor	Clarified the meaning of the technical content.
4/11/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
9/12/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	2.7	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/30/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	2.7	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
2/10/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	2.7	None	No changes to the meaning, language, or formatting of the technical content.
6/23/2016	2.7	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	2.7	None	No changes to the meaning, language, or formatting of the technical content.
9/19/2017	3.0	Major	Significantly changed the technical content.
12/12/2017	4.0	Major	Significantly changed the technical content.
6/19/2018	4.1	Minor	Clarified the meaning of the technical content.

Table of Contents

1 Introduction

This document specifies the MSSearch Query Protocol, which enables a protocol client to communicate with a protocol server to issue search queries.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

binary large object (BLOB): A discrete packet of data that is stored in a database and is treated as a sequence of uninterpreted bytes.

column: See field.

command tree: A combination of **restrictions** and sort orders that are specified for a search query.

Coordinated Universal Time (UTC): A high-precision atomic time standard that approximately tracks Universal Time (UT). It is the basis for legal, civil time all over the Earth. Time zones around the world are expressed as positive and negative offsets from UTC. In this role, it is also referred to as Zulu time (Z) and Greenwich Mean Time (GMT). In these specifications, all references to UTC refer to the time at UTC-0 (or GMT).

full-text index catalog: A collection of full-text index components and other files that are organized in a specific directory structure and contain the data that is needed to perform queries.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

handle: Any token that can be used to identify and access an object such as a device, file, or a window.

HRESULT: An integer value that indicates the result or status of an operation. A particular HRESULT can have different meanings depending on the protocol using it. See [MS-ERREF] section 2.1 and specific protocol documents for further details.

index server: A server that is assigned the task of crawling.

inflectional form: A variant of a root token that has been modified according to the linguistic rules of a given language. For example, inflections of the verb "swim" in English include "swim," "swims," "swimming," and "swam."

item: A unit of content that can be indexed and searched by a search application.

language code identifier (LCID): A 32-bit number that identifies the user interface human language dialect or variation that is supported by an application or a client computer.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

named pipe: A named, one-way, or duplex pipe for communication between a pipe server and one or more pipe clients.

natural language query: Query text that contains words and does not contain any property **restrictions**.

noise word: See stop word.

property identifier: A unique integer or a 16-bit, numeric identifier that is used to identify a specific attribute or property.

query expansion: A process in which one or more tokens are added to a search query. Typically, the additional tokens are selected based on similarity to existing tokens in a search query, such as inflectional forms, synonyms, and phonetic similarity.

query result: A result that is returned for a query. It contains the title and URL of the item, and can also contain other managed properties and a hit-highlighted summary.

query server: A server that has been assigned the task of fulfilling search queries.

restriction: A set of conditions that an item meets to be included in the search results that are returned by a query server in response to a search query.

row: A collection of **columns** that contains property values that describe a single item in a set of items that match the **restriction** specified in a query.

search catalog: All of the crawl data that is associated with a specific search application. A search catalog provides information that is used to generate query results.

search query: A complete set of conditions that are used to generate search results, including query text, sort order, and ranking parameters.

SharePoint Search SQL syntax: A set of SQL-based rules that govern the construction of a search query.

sort order: A set of rules in a search query that defines the ordering of rows in the search result. Each rule consists of a managed property, such as modified date or size, and a direction for order, such as ascending or descending. Multiple rules are applied sequentially.

stemming: A type of query expansion that factors relationships between words by reducing inflected words to their stem form or expanding stems to their inflected forms. For example, the words "swimming" and "swam" can be associated with the stem "swim."

token: A word in an item or a search query that translates into a meaningful word or number in written text. A token is the smallest textual unit that can be matched in a search query. Examples include "cat", "AB14", or "42".

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

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

[MS-ERREF] Microsoft Corporation, "Windows Error Codes".

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".

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

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol".

[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

[SALTON] Salton, G., "Automatic Text Processing: The Transformation Analysis and Retrieval of Information by Computer", 1988, ISBN: 0201122278.

[UNICODE] The Unicode Consortium, "The Unicode Consortium Home Page", http://www.unicode.org/

1.2.2 Informative References

[MSDN-FULLPROPSPEC] Microsoft Corporation, "FULLPROPSPEC structure", http://msdn.microsoft.com/en-us/library/ms690996.aspx

[MSDN-OLEDBP-OI] Microsoft Corporation, "OLE DB Programming", http://msdn.microsoft.com/en-us/library/502e07a7(VS.80).aspx

[MSDN-PROPSET] Microsoft Corporation, "Property Sets", http://msdn.microsoft.com/en-us/library/ms691041.aspx

[MSDN-QUERYERR] Microsoft Corporation, "Query-Execution Values", http://msdn.microsoft.com/en-us/library/ms690617.aspx

1.3 Overview

The search service running on a **query server** helps efficiently organize the extracted features of a collection of **items**. The MSSearch Query Protocol allows a protocol client to communicate with a protocol server hosting a search service to issue search queries. When processing files, an **index server** analyzes a set of items, extracts useful information, and then organizes the extracted information in such a way that properties of those items can be efficiently returned in response to search queries. A collection of items that can be queried comprises a search catalog. A **search catalog** contains a mechanism for quick word matching and a mechanism for quick retrieval of property values. The index server makes search catalogs available to query servers.

Conceptually, a search catalog consists of a logical table of properties with the text or value and corresponding **language code identifier (LCID)** stored in **columns** of the table. Each row of the table corresponds to a separate item in the scope of the search catalog, and each column of the table corresponds to a property.

1.3.1 Remote Querying

The MSSearch Query Protocol enables protocol clients to perform search queries against a remote protocol server hosting a search service. See [MS-SEARCH] for more information about the **SharePoint Search SQL syntax**.

The protocol client initiates a search query with the following steps:

1. The protocol client requests a connection to a protocol server hosting a search service.
2. The protocol client sends the following parameters for the **search query**:
   - Rowset properties, for example the search catalog name and configuration information.
   - The **restriction** to specify what items are to be included or excluded from the **query results**.
   - The order in which the query results are to be returned.
   - The columns to be returned in the result set.
   - The maximum number of **rows** that are to be returned for the search query.
   - The maximum time for query execution.
3. The protocol client requests a result set from the protocol server, and the protocol server responds by sending the protocol client the property values for the items that were included in the query results for the protocol client's query. After the protocol client is finished with the search query, or no longer requires additional query results, the protocol client contacts the protocol server to release the search query.

After the protocol server has released the search query, the protocol client sends a request to disconnect from the protocol server. The protocol client may also disconnect from the protocol server without issuing a disconnect request. The connection is then closed. Alternatively, the protocol client issues another search query and repeats the sequence from step 2.

1.4 Relationship to Other Protocols

The MSSearch Query Protocol relies on the SMB protocol, as described in [MS-SMB], for message transport. No other protocol depends directly on the MSSearch Protocol<1>.

1.5 Prerequisites/Preconditions

It is assumed that the protocol client has obtained the name of the protocol server and a search catalog name before this protocol is invoked. How a protocol client does this is not addressed in this specification.

It is also assumed that the protocol client and protocol server have a security association that is usable with **named pipes**, as described in [MS-SMB].

1.6 Applicability Statement

The MSSearch Query protocol is designed for querying search catalogs on a remote server from a client. The MSSearch Query protocol is designed to handle a query load of up to 100 search queries per second. Typical size of the rowset is expected in the range of 0-5000, with up to 4 columns.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

This protocol uses **HRESULT** values as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided that they set the C bit (0x20000000) for each vendor-defined value, indicating the value is a customer code.

This protocol uses NTSTATUS values as defined in [MS-ERREF] section 2.3. Vendors can choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

Property Identifiers

Properties are represented by **property identifiers**. Each property MUST have a **GUID**. This identifier consists of a GUID, representing a collection of properties called a property set plus either a string or a 32-bit integer to identify the property within the set. If the integer form of the identifier is used, the value MUST NOT be one of the following: 0x00000000, 0xFFFFFFFF, and 0xFFFFFFFE. Vendors can guarantee that their properties are uniquely defined by placing them in a property set defined by their own GUIDs<2>.

1.9 Standards Assignments

This protocol has no standards assignments, only private assignments that are made by using the allocation procedures described in other protocols.

A named pipe has been allocated to this protocol as described in [MS-SMB]; the assignments are shown in the following table.

Parameter	Value	Reference
Pipe name	\pipe\OSearch	[MS-SMB]
Pipe name	\pipe\SPSearch	[MS-SMB]

2 Messages

The following sections specify how MSSearch Query Protocol messages are transported and common MSSearch Query Protocol data types.

Note All 2-byte, 4-byte, and 8-byte signed and unsigned integers in the following structures and messages MUST be transferred in **little-endian** byte order.

2.1 Transport

All messages MUST be transported using a named pipe, as specified in [MS-SMB]. The following pipe names are used<3>:

- \pipe\OSearch
- \pipe\SPSearch

This protocol uses the underlying SMB named pipe protocol to retrieve the identity of the caller that made the connection, as specified in [MS-SMB]. The protocol client MUST set SECURITY_IDENTIFICATION as the **ImpersonationLevel** in the request [MS-SMB] to open the named pipe.

2.2 Message Syntax

2.2.1 Structures

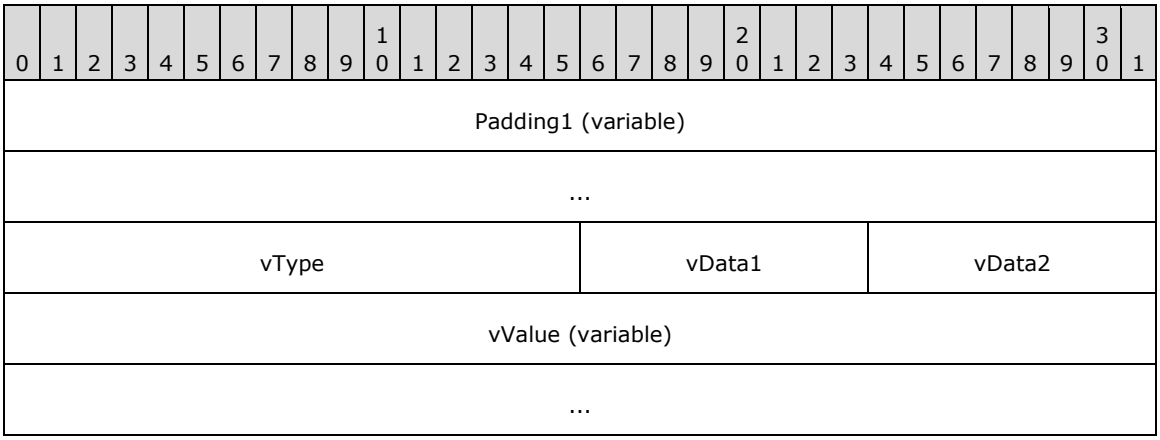
This section details data structures that are defined and used by the MSSearch Query Protocol. The following table summarizes the data structures defined in this section.

Structure	Description
CBaseStorageVariant	Contains the value on which to perform a match operation for a property that is specified in a CPropertyRestriction structure.
CFullPropSpec	Contains a property specification.
CContentRestriction	Contains a string to match for a property.
CNatLanguageRestriction	Contains a **natural language query** match for a property.
CNodeRestriction	Contains an array of **command tree** nodes specifying the **restrictions** for a search query.
CPropertyRestriction	Contains a property value to match with an operation.
CSort	Identifies a column to sort.
CVectorRestriction	Contains an array of command tree nodes specifying the restrictions for a vector space array query, as specified in [SALTON].
CRestriction	A restriction node in a query command tree.

Structure	Description
CColumnSet	Describes the columns to return.
CDbColId	Contains a column identifier.
CDbProp	Contains a rowset property.
CDbPropSet	Contains a set of rowset properties.
CPidMapper	Maps from message internal property identifiers to CFullPropSpecs.
CRowsetProperties	Contains the configuration information for a search query.
CRowVariant	Contains the fixed-size portion of a variable-length data type stored in the CPMGetRowsOut message.
CSortSet	Contains the **sort orders** for a search query.
CTableColumn	Contains a column for the CPMSetBindingsIn message.
QUERYMETADATA	Contains information about a search query.

2.2.1.1 CBaseStorageVariant

The **CBaseStorageVariant** structure contains the value on which to perform a match operation for a property specified in the CPropertyRestriction structure.



Padding1 (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

vType (2 bytes): A type indicator that indicates the type of vValue. It MUST be one of the values specified in the following table.

Value	Meaning
VT_EMPTY 0x0000	**vValue** is not present.
VT_NULL 0x0001	**vValue** is not present.
VT_I1 0x0010	A 1-byte signed integer.
VT_UI1 0x0011	A 1-byte unsigned integer.
VT_I2 0x0002	A 2-byte signed integer.
VT_UI2 0x0012	A 2-byte unsigned integer.
VT_BOOL 0x000B	A Boolean value; a 2-byte integer. **Note** Contains 0x0000 (FALSE) or 0xFFFF (TRUE).
VT_I4 0x0003	A 4-byte signed integer.
VT_UI4 0x0013	A 4-byte unsigned integer.
VT_R4 0x0004	An IEEE 32-bit floating point number, as specified in [IEEE754].
VT_INT 0x0016	A 4-byte signed integer.
VT_UINT 0x0017	A 4-byte unsigned integer. **Note** that this is identical to VT_UI4 except that VT_UINT cannot be used with VT_VECTOR (defined in the following table); the value chosen is a choice made by the higher layer that provides it to MSSearch Query Protocol, but the MSSearch Query Protocol treats VT_UINT and VT_UI4 as identical, with the exception noted previously.
VT_ERROR 0x000A	A 4-byte unsigned integer containing an HRESULT, as specified in [MS-ERREF], section 2.
VT_I8 0x0014	An 8-byte signed integer.
VT_UI8 0x0015	An 8-byte unsigned integer.
VT_R8 0x0005	An IEEE 64-bit floating point number, as specified in [IEEE754].
VT_CY 0x0006	An 8-byte two's complement integer (scaled by 10,000).
VT_DATE	A 64-bit floating point number, as specified in

Value	Meaning
0x0007	[IEEE754], representing the number of days since 00:00:00 on December 31, 1899, **Coordinated Universal Time (UTC)**.
VT_FILETIME 0x0040	A 64-bit integer representing the number of 100-nanosecond intervals since 00:00:00 on January 1, 1601, UTC.
VT_CLSID 0x0048	A 16-byte binary value containing a GUID.
VT_BLOB 0x0041	A 4-byte unsigned integer count of bytes in the **binary large object (BLOB)** followed by that many bytes of data.
VT_BLOB_OBJECT 0x0046	A 4-byte unsigned integer count of bytes in the BLOB followed by that many bytes of data.
VT_BSTR 0x0008	A 4-byte unsigned integer count of bytes in the string followed by a string, as specified in the following section under **vValue**.
VT_LPSTR 0x001E	A null-terminated ANSI string.
VT_LPWSTR 0x001F	A null-terminated Unicode (as specified in [UNICODE]) string.
VT_VARIANT 0x000C	When used in a CTableColumn description, **vValue** is a CRowVariant structure. Otherwise, it is a **CBaseStorageVariant** structure. MUST be combined with a type modifier of **VT_VECTOR**.

The following table specifies the type modifiers for **vType**. Type modifiers can be combined with **vType** using the bitwise OR operation to change the meaning of **vValue** to indicate it is one of the possible array types.

Value	Meaning
VT_VECTOR 0x1000	If the type indicator is combined with **VT_VECTOR** by using an OR operator, **vValue** is a counted array of values of the indicated type. See section 2.2.1.1.1.1. This type modifier MUST NOT be combined with the following types: VT_INT, VT_UINT, VT_BLOB, and VT_BLOB_OBJECT.

When the VT_VARIANT vType is used in a **CBaseStorageVariant** structure, it MUST be combined with a type modifier of VT_VECTOR. There is no such limitation when the VT_VARIANT **vType** is used in a CTableColumn structure, which specifies individual binding.

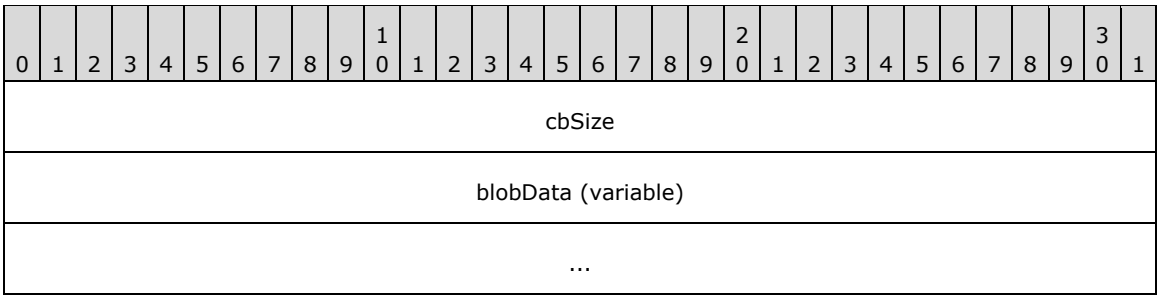
vData1 (1 byte): The value of this field MUST be set to 0x00.

vData2 (1 byte): The value of this field MUST be set to 0x00.

vValue (variable): The value for the match operation. The syntax MUST be as indicated in the **vType** field. The following table summarizes sizes for the **vValue** field, dependent on the **vType** field for fixed-length data types. The size is in bytes.

vType	Size
VT_I1, VT_UI1	1
VT_I2, VT_UI2, VT_BOOL	2
VT_I4, VT_UI4, VT_R4, VT_INT, VT_UINT, VT_ERROR	4
VT_I8, VT_UI8, VT_R8, VT_CY, VT_DATE, VT_FILETIME	8
VT_CLSID	16

If **vType** is set to VT_BLOB or VT_BSTR, the structure of **vValue** is specified in the following diagram.



For **vType** set to VT_BLOB, this field is opaque binary BLOB data.

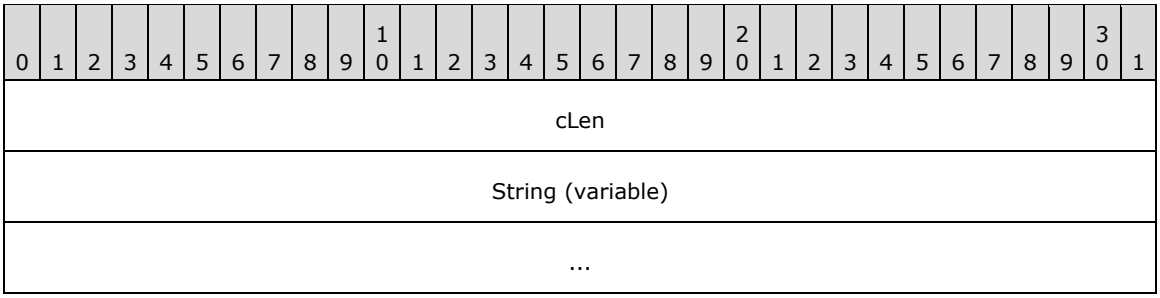
For **vType** set to VT_BSTR, this field is a set of characters. The protocol client and protocol server MUST be configured to have interoperable character sets (which is not addressed in this protocol). There is no requirement that it be null-terminated.

cbSize (4 bytes): A 32-bit unsigned integer. Indicates the size of the **blobData** field in bytes. If **vType** is set to VT_BSTR, **cbSize** MUST be set to 0x00000000 when the string represented is an empty string.

blobData (variable): MUST be of length **cbSize** in bytes.

For a **vType** set to either VT_LPSTR or VT_LPWSTR, the structure of **vValue** is shown in the following diagram, with these caveats:

- If **vType** is set to VT_LPSTR, cLen indicates the size of the string in ANSI characters, and string is a null-terminated ANSI string.
- If **vType** is set to VT_LPWSTR, cLen indicates the size of the string in Unicode characters, and string is a null-terminated Unicode string.



cLen (4 bytes): A 32-bit unsigned integer, indicating the size of the **string** field including the terminating null. A value of 0x00000000 indicates that no such string is present.

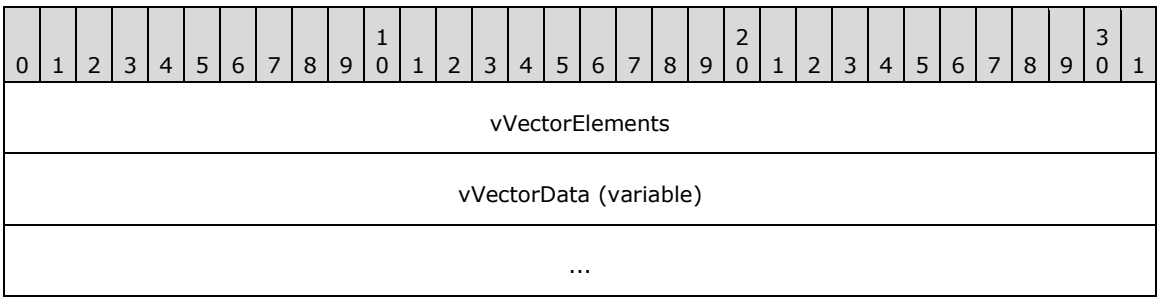
String (variable): Null-terminated string. This field MUST be absent if **cLen** equals 0x00000000.

2.2.1.1.1 CBaseStorageVariant Structures

The **VT_VECTOR** structure is used in the **CBaseStorageVariant** structure.

2.2.1.1.1.1 VT_VECTOR

The **VT_VECTOR** structure is used to pass one-dimensional arrays.



vVectorElements (4 bytes): Unsigned 32-bit integer, indicating the number of elements in the **vVectorData** field.

vVectorData (variable): An array of items that have a type indicated by **vType** with the 0x1000 bit cleared. The size of an individual fixed-length item can be obtained from the fixed-length data type table, as specified in section 2.2.1.1. The length of this field in bytes can be calculated by multiplying **vVectorElements** by the size of an individual item.

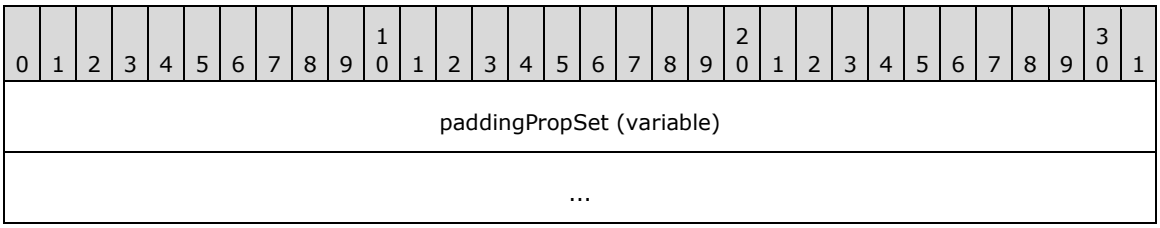
For variable-length data types, **vVectorData** contains a sequence of consecutively marshaled simple types in which the type is indicated by **vType** with the 0x1000 bit cleared.

The elements in the **vVectorData** field MUST be separated by 0 to 3 padding bytes such that each element begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this array. If padding bytes are present, the value they contain is arbitrary. The contents of the padding bytes MUST be ignored by the receiver.

2.2.1.2 CFullPropSpec

The **CFullPropSpec** structure contains a property set GUID and a property identifier to uniquely identify a property. A **CFullPropSpec** instance has a property set GUID and either an integer property identifier or a string property name. For properties to match, the **CFullPropSpec** structure MUST match the column identifier in the **full-text index catalog**. There is no conversion between property identifiers and property names. Property names are case insensitive.

For more information, see the Indexing Service definition of **FULLPROPSPEC** in [MSDN-FULLPROPSPEC].



_guidPropSet (16 bytes)
...
...
ulKind
PrSpec

paddingPropSet (variable): This field MUST be 0 to 7 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 8 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

_guidPropSet (16 bytes): The GUID of the property set to which the property belongs.

ulKind (4 bytes): A 32-bit unsigned integer. MUST be set to 0x00000001.

PrSpec (4 bytes): A 32-bit unsigned integer which contains the property identifier.

2.2.1.3 CContentRestriction

The CContentRestriction structure contains a word or phrase to match in the search catalog for a specific property.

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
_Property (variable)																															
...																															
Padding1 (variable)																															
...																															
Cc																															
_pwcsPhrase (variable)																															
...																															
Padding2 (variable)																															
...																															
Lcid																															
_ulGenerateMethod																															

_Property (variable): A CFullPropSpec structure. This field indicates the property on which to perform a match operation.

Padding1 (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

Cc (4 bytes): A 32-bit unsigned integer, specifying the number of characters in the **_pwcsPhrase** field.

_pwcsPhrase (variable): A Unicode string that is not null-terminated representing the word or phrase to match for the property. This field MUST NOT be empty. The **Cc** field contains the length of the string.

Padding2 (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

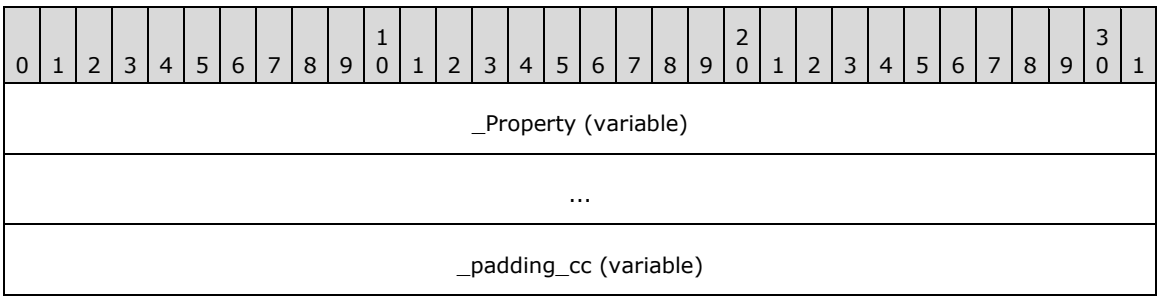
Lcid (4 bytes): A 32-bit unsigned integer, indicating the LCID of **_pwcsPhrase**, as specified in [MS-LCID].

_ulGenerateMethod (4 bytes): A 32-bit unsigned integer, specifying the method to use when generating alternate word forms. The following table specifies the possible values for this field along with their meanings.

Value	Meaning
GENERATE_METHOD_EXACT 0x00000000	Exact match. Each word in the phrase MUST match exactly in the search catalog.
GENERATE_METHOD_PREFIX 0x00000001	Prefix match. Each word in the phrase is considered a match if the word is a prefix of a crawled string. For example, if the word "barking" is crawled, then "bar" would match when performing a prefix match.
GENERATE_METHOD_INFLECT 0x00000002	Matches **inflectional forms** of a word. An inflectional form of a word is a variant of the root word in the same part of speech that has been modified, according to linguistic rules of a given language. For example, inflectional forms of the verb swim in English include swim, swims, swimming, and swam.

2.2.1.4 CNatLanguageRestriction

The **CNatLanguageRestriction** structure contains a natural language query match for a property.



...
Cc
_pwcsPhrase (variable)
...
_padding_lcid (variable)
...
Lcid

_Property (variable): A CFullPropSpec structure. This field indicates the property on which to perform the match operation.

_padding_cc (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

Cc (4 bytes): A 32-bit unsigned integer, specifying the number of characters in the **_pwcsPhrase** field.

_pwcsPhrase (variable): A Unicode string that is not null-terminated with the text to be searched for within the specific property. This string MUST NOT be empty. The **Cc** field contains the length of the string.

_padding_lcid (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

Lcid (4 bytes): A 32-bit unsigned integer indicating the LCID of **_pwcsPhrase**, as specified in [MS-LCID].

2.2.1.5 CNodeRestriction

The **CNodeRestriction** structure contains an array of command tree **restriction** nodes for constraining the results of a search query.

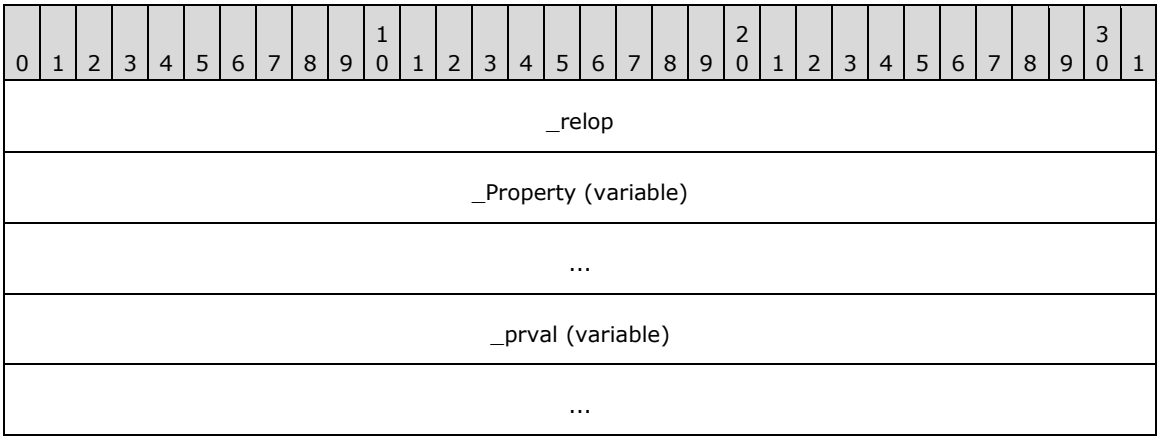
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
_cNode																															
_paNode (variable)																															
...																															

_cNode (4 bytes): A 32-bit unsigned integer specifying the number of CRestriction structures contained in the **_paNode** field.

_paNode (variable): An array of **CRestriction** structures. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this array. If padding bytes are present, the value they contain is arbitrary. The content of the padding bytes MUST be ignored by the receiver.

2.2.1.6 CPropertyRestriction

The **CPropertyRestriction** structure contains a property to get from each row, a comparison operator, and a constant. For each row, the value returned by the specific property in the row is compared against the constant to see if it has the relationship specified by the **_relop** field. For the comparison to be true, the data types of the values MUST match.



_relop (4 bytes): A 32-bit unsigned integer specifying the relation to perform on the property. **_relop** MUST be one of the following values with an optional bitwise-OR mask applied to the value.

Value	Meaning
PREQ 0x00000004	An equality comparison.
PRNE 0x00000005	A not-equal comparison.

The possible values for the optional mask are listed in the following table.

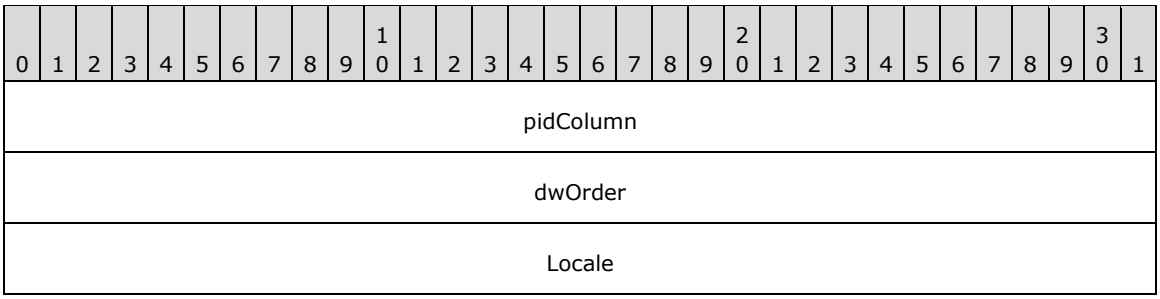
Value	Meaning
PRAny 0x00000200	The **restriction** is true if any element in the property value has the relationship with some element in the **_prval** field.

_Property (variable): A CFullPropSpec structure indicating the property on which to perform a match operation.

_prval (variable): A CBaseStorageVariant structure containing the value to relate to the property

2.2.1.7 CSort

The **CSort** structure identifies a column, direction and LCID to sort by.



pidColumn (4 bytes): A 32-bit unsigned integer. This is the index in CPidMapper for the property to sort by.

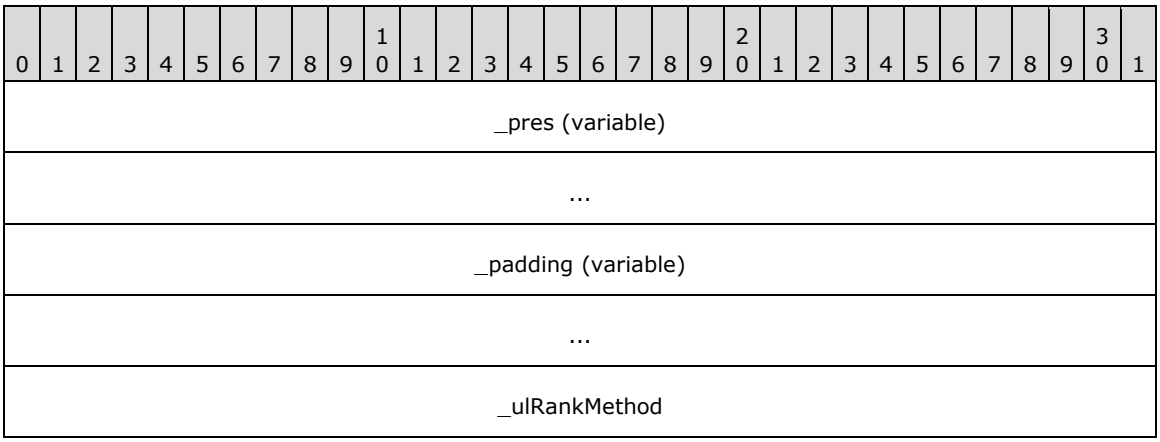
dwOrder (4 bytes): A 32-bit unsigned integer. MUST be one of the following values, specifying how to sort based on the column.

Value	Meaning
QUERY_SORTASCEND 0x00000000	The rows are to be sorted in ascending order based on the values in the column specified.
QUERY_SORTDESCEND 0x00000001	The rows are to be sorted in descending order based on the values in the column specified.

Locale (4 bytes): A 32-bit unsigned integer indicating the LCID (as specified in [MS-LCID]) of the column. The LCID determines the sorting rules to use when sorting textual values.

2.2.1.8 CVectorRestriction

The **CVectorRestriction** structure contains a weighted OR operation over **restriction** nodes. Vector restrictions represent queries using the full text vector space model of ranking (as specified in [SALTON]). In addition to the OR operation they also compute a rank depending on the ranking algorithm.



_pres (variable): A CNodeRestriction command tree on which a ranked OR operation is to be performed.

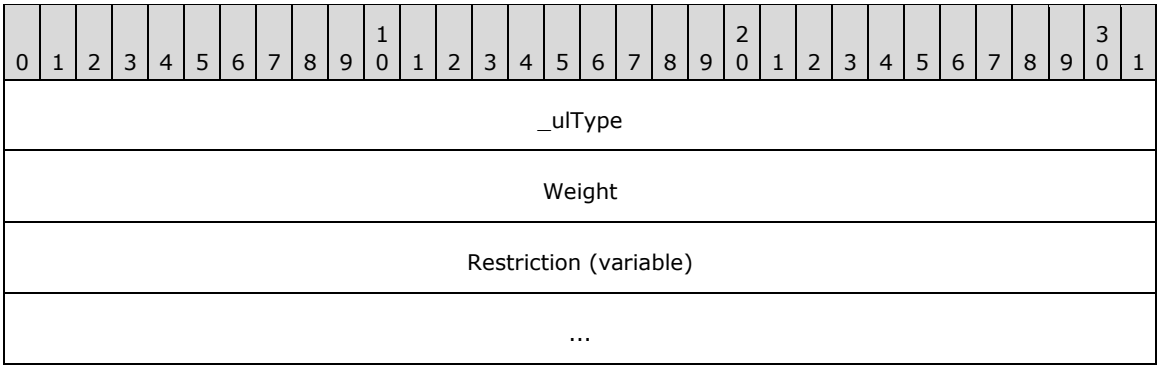
_padding (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

_ulRankMethod (4 bytes): A 32-bit unsigned integer specifying a ranking algorithm that MUST be set to one of the following values.

Value	Meaning
VECTOR_RANK_MIN 0x00000000	Use the minimum algorithm as specified in [SALTON].
VECTOR_RANK_MAX 0x00000001	Use the maximum algorithm as specified in [SALTON].
VECTOR_RANK_INNER 0x00000002	Use the inner product algorithm as specified in [SALTON].
VECTOR_RANK_DICE 0x00000003	Use the Dice coefficient algorithm as specified in [SALTON].
VECTOR_RANK_JACCARD 0x00000004	Use the Jaccard coefficient algorithm as specified in [SALTON].

2.2.1.9 CRestriction

The **CRestriction** structure contains a **restriction** node in a query command tree.



_ulType (4 bytes): A 32-bit unsigned integer indicating the restriction type used for the command tree node. The type determines what is found in the **Restriction** field of the structure, as described in the following table. This MUST be set to one of the following values.

Value	Meaning
RTNone 0x00000000	The node represents a **noise word** in a vector query.
RTAnd 0x00000001	The node contains a CNodeRestriction on which a logical AND operation is to be performed.
RTOr 0x00000002	The node contains a **CNodeRestriction** on which a logical OR operation is to be performed.
RTNot 0x00000003	The node contains a CRestriction on which a NOT operation is to be performed.

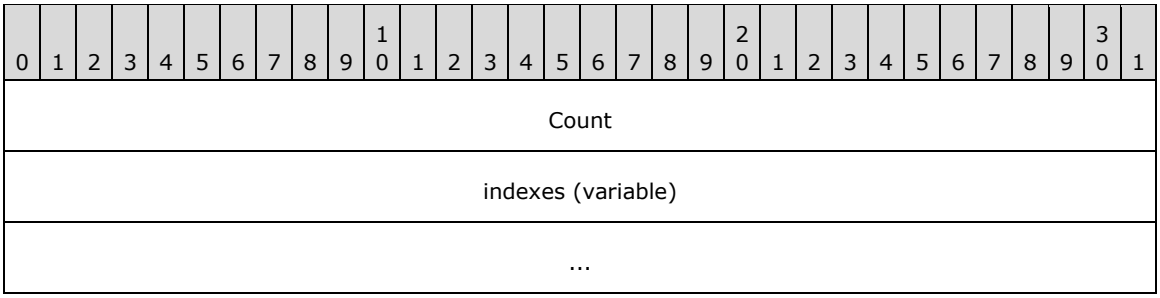
Value	Meaning
RTContent 0x00000004	The node contains a CContentRestriction.
RTProperty 0x00000005	The node contains a CPropertyRestriction.
RTProximity 0x00000006	The node contains a **CNodeRestriction** with an array of **CContentRestriction** structures. Any other kind of restriction is undefined. The restriction requires the words or phrases found in the **CContentRestriction** structures to be within a query server defined range to be a match. The query server can also compute a rank based on how far apart the words or phrases are.
RTVector 0x00000007	The node contains a CVectorRestriction.
RTNatLanguage 0x00000008	The node contains a CNatLanguageRestriction.
RTPhrase 0xFFFFFFFD	The node contains a **CNodeRestriction** on which a phrase match is to be performed.

Weight (4 bytes): A 32-bit unsigned integer representing the weight of the node. **Weight** indicates the node's importance relative to other nodes in the query command tree. Higher weight values are more important.

Restriction (variable): The restriction type for the command tree node. The syntax MUST be as indicated by the **_ulType** field.

2.2.1.10 CColumnSet

The **CColumnSet** structure specifies the column numbers to be returned. This structure is used in reference to a specific CPidMapper structure.



Count (4 bytes): A 32-bit unsigned integer specifying the number of elements in the indexes array.

indexes (variable): An array of 4-byte unsigned integers representing zero-based indexes into the **aPropSpec** array in the corresponding **CPidMapper** structure. The corresponding property values are returned as columns in the result set.

2.2.1.11 CDbColId

The **CDbColId** structure contains a column identifier.

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
eKind																															
padding (variable)																															
...																															
GUID (16 bytes)																															
...																															
...																															
ulId																															

eKind (4 bytes): MUST be set to 0x00000001.

padding (variable): This field MUST be 0 to 7 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 8 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

GUID (16 bytes): The property GUID.

ulId (4 bytes): This field contains an unsigned 32-bit integer specifying the property identifier.

2.2.1.12 CDbProp

The **CDbProp** structure contains an OLE-DB DBPROP database property. These properties control how search queries are interpreted by the query server. For more information about OLE-DB, see [MSDN-OLEDBP-OI].

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
DBPROPID																															
DBPROPOPTIONS																															
DBPROPSTATUS																															
colid (variable)																															
...																															
_padding (variable)																															
...																															

vValue (variable)
...

DBPROPID (4 bytes): A 32-bit unsigned integer indicating the property identifier. This field uniquely identifies each property in a particular search query but has no other interpretation. In particular, it is not a **ulId** as found in the CDbColId structure.

DBPROPOPTIONS (4 bytes): Property options. This field MUST be set to 0x00000000.

DBPROPSTATUS (4 bytes): Property status. This field MUST be set to 0x00000000.

colid (variable): A **CDbColId** structure that defines the database property being passed.

_padding (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

vValue (variable): A CBaseStorageVariant containing the property value.

2.2.1.12.1 Database Properties

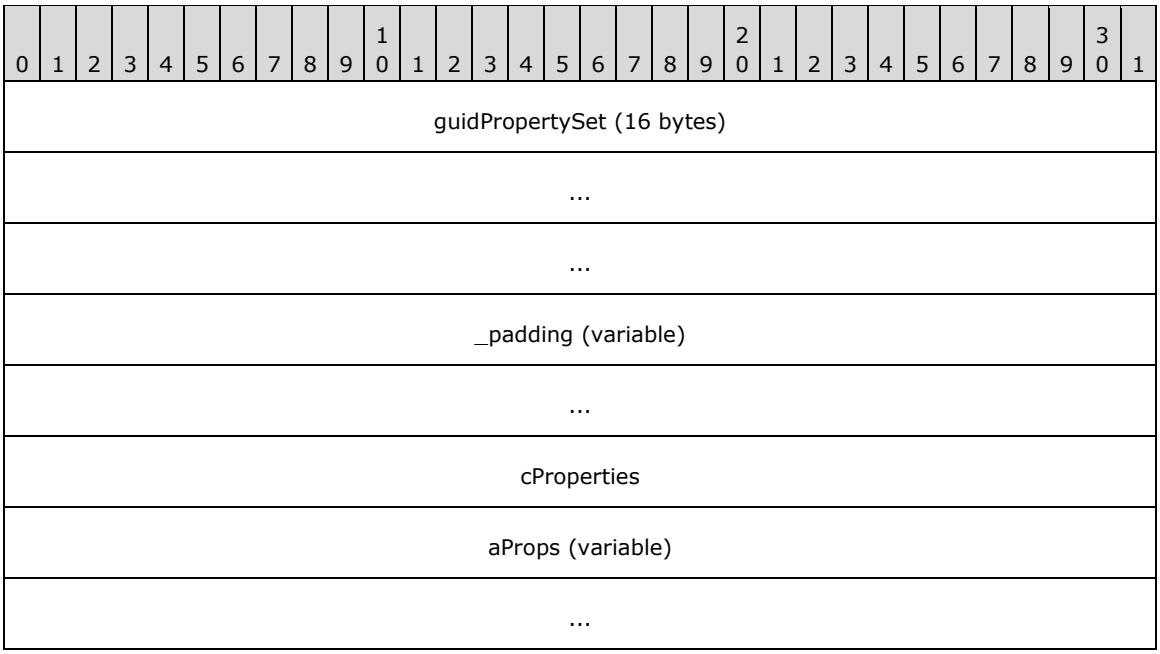
The MSSearch Query Protocol supports the following database properties to control the behavior of the query server. These properties are grouped into three property sets identified in the **guidPropertySet** field of the CDbPropSet structure.

The following table lists the properties that are part of the **DBPROPSET_FSCIFRMWRK_EXT** property set.

Value	Meaning
DBPROP_CI_CATALOG_NAME 0x00000002	Specifies the name of the search catalog or search catalogs to query. Value MUST be a VT_LPWSTR or a VT_BSTR. The structure MUST be set such that the **eKind** field contains 0x00000001 and the **GUID** and **ulID** fields are filled with zeros.
DBPROP_CI_QUERY_TYPE 0x00000007	Specifies the type of query using a CDbColId structure. The structure MUST be set such that the **eKind** field contains 0x00000001 and the **GUID** and **ulID** fields are filled with zeros. When this property is specified the **vValue** field MUST contain 0x00000000, indicating a regular search query.

2.2.1.13 CDbPropSet

The **CDbPropSet** structure contains a set of properties. The first field, **guidPropertySet**, is not padded and will begin where the previous structure in the message ended (as indicated by the "previous structure" entry in the following diagram). The 1-byte length of "previous structure" is arbitrary, and is not meant to suggest **guidPropertySet** will begin on any particular boundary. However, the **cProperties** field MUST be aligned to begin at a multiple of 4 bytes from the beginning of the message, and, hence, the format is depicted as follows.



guidPropertySet (16 bytes): A GUID identifying the property set. MUST be set to the binary form of the value DBPROPSET_FSCIFRMWRK_EXT (A9BD1526-6A80-11D0-8C9D-0020AF1D740E), identifying the property set of the properties contained in the **aProps** field.

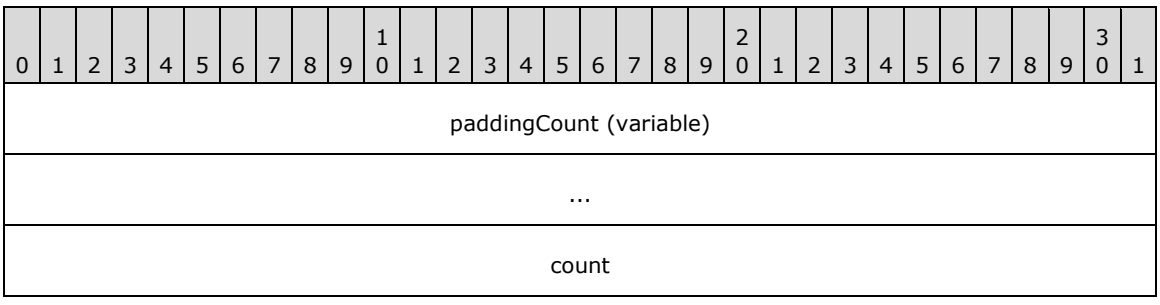
_padding (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

cProperties (4 bytes): A 32-bit unsigned integer containing the number of elements in the **aProps** array.

aProps (variable): An array of CDbProp structures containing properties. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this array. If padding bytes are present, the value they contain is arbitrary. The content of the padding bytes MUST be ignored by the receiver.

2.2.1.14 CPidMapper

The **CPidMapper** structure contains an array of property specifications and serves to map from a property offset to a CFullPropSpec. The more compact property offsets are used to name properties in other parts of the protocol. Because offsets are more compact they allow shorter property references in other parts of the protocol.



aPropSpec (variable)
...

paddingCount (variable): This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the byte offset from the beginning of the message to the **count** field is a multiple of 4 bytes. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

count (4 bytes): A 32-bit unsigned integer containing the number of elements in the **aPropSpec** array.

aPropSpec (variable): An array of CFullPropSpec structures.

2.2.1.15 CRowsetProperties

The **CRowsetProperties** structure contains configuration information for a search query.

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
_uBooleanOptions																															
_ulMaxOpenRows																															
_ulMemoryUsage																															
_cMaxResults																															
_cCmdTimeout																															
_dwQueryID (optional)																															

_uBooleanOptions (4 bytes): This field specifies various query Boolean options.

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
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF

A - U0: MUST be set to 1 and MUST be ignored.

B through O - U1, U2, A1, A2, **U3, U4, U5, U6, U7, U8, U9, U10, U11, U12:** MUST be set to 0 and MUST be ignored.

P - IN: Specifies the desired **noise word** behavior. Its value MUST be 1 if the noise words are ignored, and 0 otherwise.

Q through U - U13, U14, U15, U16, U17: MUST be set to 0 and MUST be ignored.

V - AT: Specifies the desired **token** inclusion behavior that MUST be used by the server unless the inclusion behavior is specified explicitly by keyword syntax as specified in [MS-SEARCH] section 2.2.11.8. If the value is 0, the server MUST return only the search results containing all of the

tokens in the query. Otherwise, the server MUST return search results that contain any of the tokens.

W - ES: Specifies the desired **stemming query expansion** behavior. If the value is 1, the server MUST use stemming query expansion. If the value is 0, the server MUST NOT use stemming query expansion.

X through AF - EP, EN, **IT, U18, U19, U20, U21, U22, U23:** MUST be set to 0 and MUST be ignored.

_ulMaxOpenRows (4 bytes): A 32-bit unsigned integer. MUST be set to 0x00000000. Not used, and MUST be ignored.

_ulMemoryUsage (4 bytes): A 32-bit unsigned integer. MUST be set to 0x00000000. Not used, and MUST be ignored.

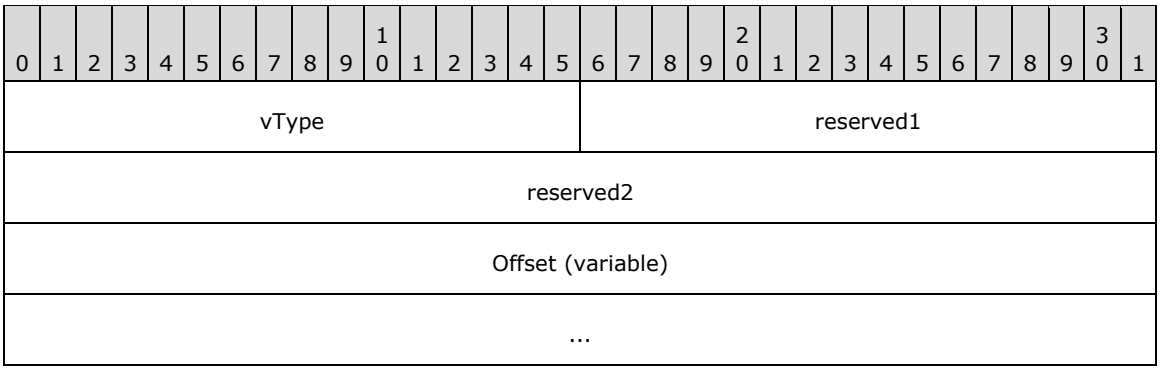
_cMaxResults (4 bytes): A 32-bit unsigned integer, specifying the maximum number of rows that are to be returned for the query. If **_cMaxResults** is set to 0x00000000 then server assumes all results are requested and behaves as if 0xFFFFFFFF was specified in **_cMaxResults**.

_cCmdTimeout (4 bytes): A 32-bit unsigned integer, specifying the number of seconds at which a query is to time out, counting from the time the query starts executing on the server. On a timeout, the query is interrupted and terminated, and the server continues to communicate with the client using the regular sequence of messages. A value of 0x00000000 means that the query is not to time out.

_dwQueryID (4 bytes): A 32-bit unsigned integer that identifies the query for debugging purposes. This field MUST only be present if both protocol client and protocol server are capable of handling **_dwQueryID** value as indicated by the **_iClientVersion** field in the CPMConnectIn message and the **_serverVersion** field in the CPMConnectOut message. The value of this field can be any arbitrary value. The protocol server SHOULD use this value in any logging related to the query being executed (if any).

2.2.1.16 CRowVariant

The **CRowVariant** structure contains the fixed-size portion of a variable-length data type stored in the CPMGetRowsOut message.



vType (2 bytes): A type indicator, indicating the type of **vValue**. It MUST be set to VT_I4 or VT_LPWSTR, as specified in section 2.2.1.1.

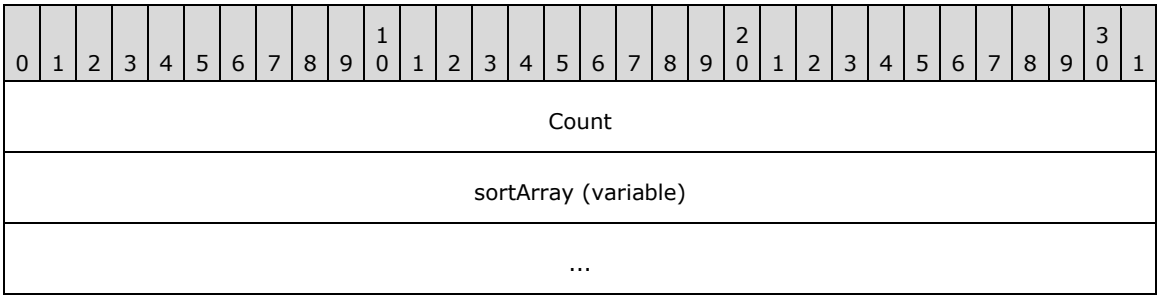
reserved1 (2 bytes): Not used. MUST be ignored on receipt.

reserved2 (4 bytes): Not used. MUST be ignored on receipt.

Offset (variable): An offset to variable-length data (for example, a string). This MUST be a 32-bit value (4-bytes long) if 32-bit offsets are being used (per the rules in section 2.2.3.7), or a 64-bit value (8-bytes long) if 64-bit offsets are being used.

2.2.1.17 CSortSet

The **CSortSet** structure contains the sort order of the search query.

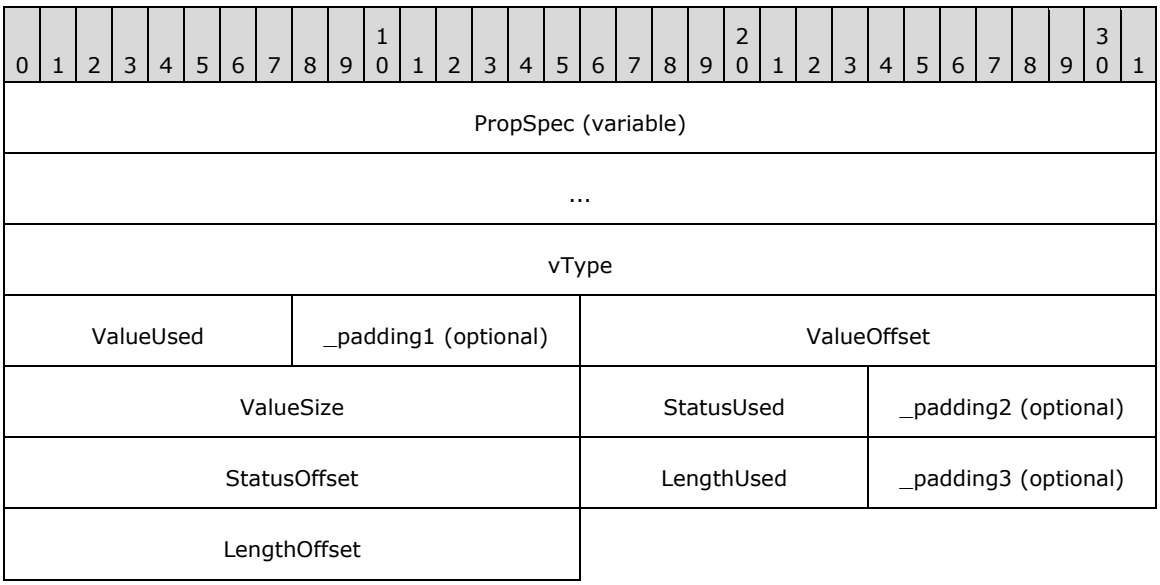


count (4 bytes): A 32-bit unsigned integer specifying the number of elements in **sortArray**.

sortArray (variable): An array of CSort structures describing the order in which to sort the results of the search query. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure has a 4-byte alignment from the beginning of a message. Such padding bytes can be any arbitrary value, and MUST be ignored on receipt.

2.2.1.18 CTableColumn

The **CTableColumn** structure contains a column of a CPMSetBindingsIn message



PropSpec (variable): A **CFullPropSpec** structure.

vType (4 bytes): A 32-bit reserved field. MUST be set to 0x0000000C.

ValueUsed (1 byte): A 1-byte reserved field. MUST be set to 0x01.

_padding1 (1 byte): A 1-byte field.

Note This field MUST be inserted before **ValueOffset** if, without it, **ValueOffset** would not begin at an even offset from the beginning of the message. The value of this byte is arbitrary, and MUST be ignored.

ValueOffset (2 bytes): An unsigned 2-byte integer specifying the offset of the column value in the row.

ValueSize (2 bytes): An unsigned 2-byte integer specifying the size of the column value in bytes.

StatusUsed (1 byte): A 1-byte reserved field. MUST be set to 0x01.

_padding2 (1 byte): A 1-byte field. Note This field MUST be inserted before **StatusOffset** if, without it, the **StatusOffset** field would not begin at an even offset from the beginning of the message. The value of this byte is arbitrary, and MUST be ignored.

StatusOffset (2 bytes): An unsigned 2-byte integer. Specifies the offset of the column status in the row.

Status is represented as one byte in the response by the offset specified in the **StatusOffset** request field. The status byte MUST be equal to 0x00.

LengthUsed (1 byte): A reserved 1-byte field. MUST be set to 0x01.

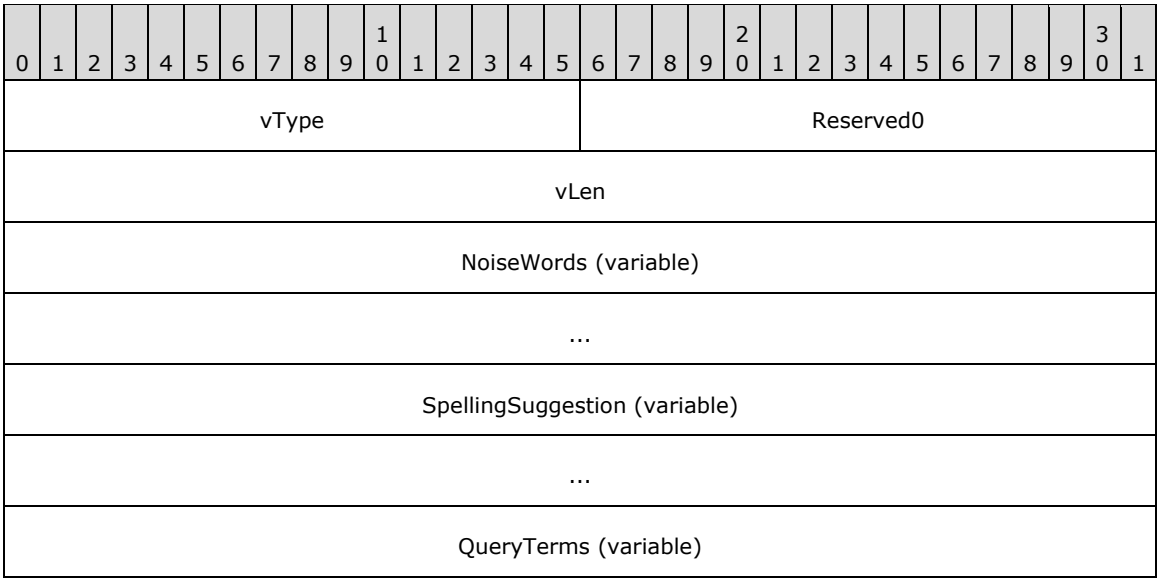
_padding3 (1 byte): A 1-byte field.

Note This field MUST be inserted before **LengthOffset** if, without it, **LengthOffset** would not begin at an even offset from the beginning of a message. The value of this byte is arbitrary, and MUST be ignored.

LengthOffset (2 bytes): An unsigned 2-byte integer specifying the offset of the column length in the row. In CPMGetRowsOut, length is represented by a 32-bit unsigned integer by the offset specified in **LengthOffset**.

2.2.1.19 QUERYMETADATA

The **QUERYMETADATA** structure contains a serialized representation of the metadata about a search query. This structure is returned in the **vValue** field of the CPMFetchValueOut message.



...
TermIds (variable)
...
EstimatedCount

vType (2 bytes): A 16-bit reserved field describing the type of the property. MUST be set to VT_BLOB as specified in section 2.2.1.1.

Reserved0 (2 bytes): A reserved 16-bit field. MUST be set to 0x0000.

vLen (4 bytes): A 32-bit field. MUST be set to the length in bytes of the **NoiseWords**, **SpellingSuggestion**, **QueryTerms**, **TermIds** and **EstimatedCount** fields.

NoiseWords (variable): A **CBaseStorageVariant** containing terms which were treated as **noise words** during query execution. The **vType** field of this structure MUST be set to VT_VECTOR | VT_LPWSTR (0x101F). The **vValue** field MUST contain an array of 0 or more query terms which were treated as noise words by the query. See serialization for **vValue** in section 2.2.1.1.

SpellingSuggestion (variable): A **CBaseStorageVariant** containing terms which have been determined by the server as being alternate spelling of terms specified in the query<4>. The **vType** field of this structure MUST be set to VT_LPWSTR (0x001F). The **vValue** field MUST contain space-delimited keywords and any keywords which are spelling suggestions MUST be prefixed with "<suggestion>" and post fixed with "</suggestion>". If there are no spelling suggestions then the **vValue** MUST contain a null-terminated empty VT_LPWSTR. See serialization for **vValue** in section 2.2.1.1.

QueryTerms (variable): A **CBaseStorageVariant** containing terms from the query. The **vType** field of this structure MUST be set to VT_VECTOR | VT_LPWSTR (0x101F). The **vValue** field MUST contain an array of 0 or more query terms. See serialization for **vValue** in section 2.2.1.1.

TermIds (variable): A **CBaseStorageVariant** containing term identifiers from the search query. The **vType** field of the **TermIds** field MUST be set to VECTOR | VT_UI4 (0x1013), and the **vVectorElements** field of the **TermIds** structure MUST be set to the same value as the **vVectorElements** field of the **QueryTerms** structure. The **vVectorData** field SHOULD contain term identifier values that are specific to the protocol server implementation. The protocol client MUST ignore the values in **vVectorData<5>**.

See serialization for **vValue** in section 2.2.1.1.

EstimatedCount (4 bytes): A 32-bit field containing the estimated number of total results, regardless of the number of rows requested by the protocol client.

2.2.2 Message Headers

All MSSearch Query Protocol messages have a 16-byte header.

The following diagram shows the MSSearch Query Protocol message header format.

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
_msg																															

_status
_ulChecksum
_ulReserved2

_msg (4 bytes): A 32-bit integer that identifies the type of message following the header.

The following table lists the MSSearch Query Protocol messages and the integer values specified for each message. As shown in the table, some values identify two messages in the table. In those instances, the message following the header can be identified by the direction of the message flow. If the direction is protocol client to protocol server, the message with "In" appended to the message name is indicated. If the direction is protocol server to protocol client, the message with "Out" appended to the message name is indicated.

Value	Meaning
0x000000C8	CPMConnectIn or CPMConnectOut
0x000000C9	CPMDisconnect
0x000000CA	CPMCreateQueryIn or CPMCreateQueryOut
0x000000CB	CPMFreeCursorIn or CPMFreeCursorOut
0x000000CC	CPMGetRowsIn or CPMGetRowsOut
0x000000D0	CPMSetBindingsIn
0x000000E4	CPMFetchValueIn or CPMFetchValueOut

_status (4 bytes): An HRESULT, indicating the status of the requested operation. When sent by the protocol client, this field MAY contain any value and the protocol server MUST ignore the value<6>.

_ulChecksum (4 bytes): A 32-bit integer field. The **_ulChecksum** MUST be calculated as specified in section 3.2.4 for the following messages:

- CPMConnectIn
- CPMCreateQueryIn
- CPMSetBindingsIn
- CPMGetRowsIn
- CPMFetchValueIn

Note For all other messages from the protocol client, **_ulChecksum** MUST be ignored by the receiver. A protocol client MUST ignore the **_ulChecksum** field.

_ulReserved2 (4 bytes): A 32-bit integer field. If the message type is CPMGetRowsIn and 64-bit offsets are used, as specified in section 2.2.3.7, this field MUST contain the upper 32 bits of the base value to use for pointer calculations in the row buffer (the **_ulClientBase** field of the CPMGetRowsIn message contains the lower 32 bits). Otherwise, the value MUST be set to 0x00000000.

2.2.3 Messages

The following sections specify the MSSearch Query Protocol messages.

2.2.3.1 CPMConnectIn

The **CPMConnectIn** message begins a session between the protocol client and protocol server.

The format of the **CPMConnectIn** message that follows the header is shown in the following diagram.

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
_iClientVersion																															
_fClientIsRemote																															
_cbBlob1																															
_paddingcbdBlob2 (variable)																															
...																															
_cbBlob2																															
_padding																															
...																															
...																															
MachineName (variable)																															
...																															
UserName (variable)																															
...																															
_paddingcPropSets (variable)																															
...																															
cPropSets																															
PropertySets (variable)																															
...																															

_paddingExtPropset (variable)
...
cExtPropSet
ExtPropertySets (variable)
...

_iClientVersion (4 bytes): A 32-bit integer indicating whether the protocol client is assumed capable of handling 64-bit offsets in CPMGetRowsOut messages<7>.and whether the protocol client is assumed capable of handling the **_dwQueryID** field in the CRowsetProperties structure. The value for this field MUST be set to one of the following values.

Value	Meaning
0x00000102	The protocol client is not capable of handling 64-bit offsets in CPMGetRowsOut messages and is not capable of handling the **_dwQueryID** field in the CRowsetProperties structure.
0x00010102	The protocol client is capable of handling 64-bit offsets in **CPMGetRowsOut** messages and is not capable of handling the **_dwQueryID** field in the **CRowsetProperties** structure.
0x00000103	The protocol client is not capable of handling 64-bit offsets in **CPMGetRowsOut** messages and is capable of handling the **_dwQueryID** field in the CRowsetProperties structure.
0x00010103	The protocol client is capable of handling 64-bit offsets in **CPMGetRowsOut** messages and is capable of handling the **_dwQueryID** in the **CRowsetProperties** structure.

_fClientIsRemote (4 bytes): A Boolean value indicating whether the protocol client is running on a different computer than the protocol server.

_cbBlob1 (4 bytes): A 32-bit unsigned integer indicating the size in bytes of the **cPropSets** and **PropertySets** fields, combined.

_paddingcbdBlob2 (variable): This field MUST be 0 to 7 bytes in length. The length of this field MUST be such that the byte offset from the beginning of the message to the first structure contained in the **_cbBlob2** field is a multiple of 8 bytes. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

_cbBlob2 (4 bytes): A 32-bit unsigned integer indicating the size in bytes of the **cExPropSet** and **ExtPropertySets** fields, combined.

_padding (12 bytes): 12 bytes of padding. MUST be ignored.

MachineName (variable): The computer name of the protocol client. The name string MUST be a null-terminated array of less than 512 Unicode characters, including the NULL terminator.

UserName (variable): A string that represents the user name of the person who is running the application that invoked this protocol. The name string MUST be a null-terminated array of fewer than 512 Unicode characters when concatenated with **MachineName**.

_paddingcPropSets (variable): This field MUST be 0 to 7 bytes in length. The number of bytes MUST be the number required to make the byte offset of the **cPropSets** field a multiple of 8 from the beginning of the message that contains this structure. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

cPropSets (4 bytes): A 32-bit unsigned integer indicating the number of **CDbPropSet** structures following this field. This field MUST be set to 0x00000001.

PropertySets (variable): An array of **CDbPropSet** structures. The number of elements in this array MUST be equal to **cPropSets**.

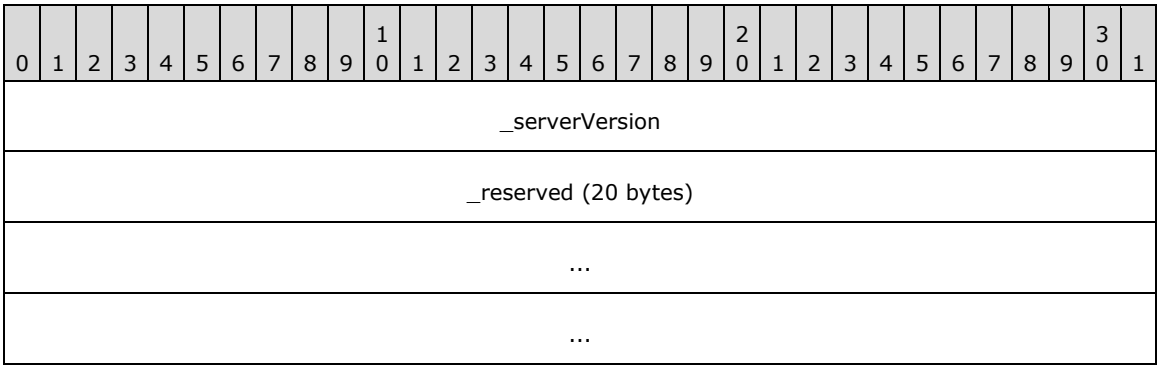
_paddingExtPropset (variable): This field MUST be 0 to 7 bytes in length. The number of bytes MUST be the number required to make the byte offset of the **cExtPropSets** field from the beginning of the message that contains this structure equal a multiple of 8. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

cExtPropSet (4 bytes): A 32-bit reserved field. MUST be set to 0x00000001.

ExtPropertySets (variable): An array of **CDbPropSet** structures. The number of elements in this array MUST be equal to **cExtPropSet**.

2.2.3.2 CPMConnectOut

The **CPMConnectOut** message contains a response to a CPMConnectIn message. The format of the **CPMConnectOut** message that follows the header is shown in the following diagram.



_serverVersion (4 bytes): A 32-bit integer, indicating whether the server can support 64-bit offsets, as specified in section 2.2.3.7 and whether the server can support the **_dwQueryID** field in the CRowsetProperties structure.

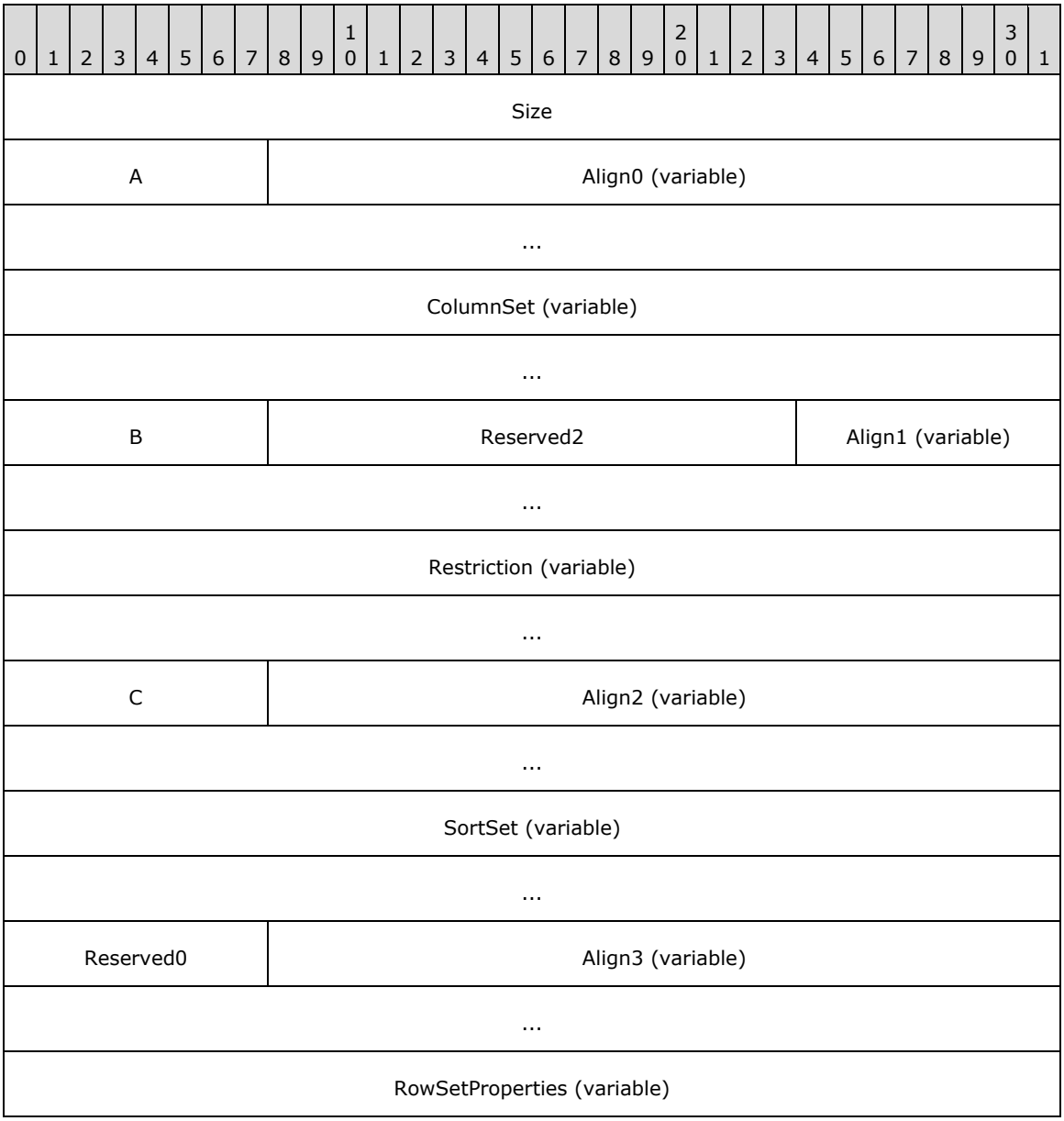
Value	Meaning
0x00000102	The protocol server can only send 32-bit offsets and can only work with **CRowsetProperties** structure without the **_dwQueryID** field.
0x00010102	The protocol server can send 32-bit or 64-bit offsets and can only work with a **CRowsetProperties** structure without the **_dwQueryID** field.
0x00000103	The protocol server can only send 32-bit offsets and can work with a **CRowsetProperties** structure with

Value	Meaning
	and without the **_dwQueryID** field.
0x00010103	The protocol server can send 32-bit or 64-bit offsets and can work with **CRowsetProperties** structure with and without the **_dwQueryID** field.

_reserved (20 bytes): A reserved field. The protocol server SHOULD<8> set all bits of this field to 0. The protocol client MUST ignore this value.

2.2.3.3 CPMCreateQueryIn

The **CPMCreateQueryIn** message creates a new search query. The format of the **CPMCreateQueryIn** message that follows the header is shown in the following diagram.



...
PidMapper (variable)
...
Reserved1
LCID

Size (4 bytes): A 32-bit unsigned integer indicating the number of bytes from the beginning of this field to the end of the message.

A - CColumnSetPresent (1 byte): A byte field indicating if the **ColumnSet** field is present. This field MUST be set to one of the following values.

Value	Meaning
0x00	The **ColumnSet** field MUST be absent.
0x01	The **ColumnSet** field MUST be present.

Align0 (variable): A field structure of 0, 1, 2 or 3 bytes that is used to align the next field to a 4-byte boundary. MUST be ignored by the protocol server.

ColumnSet (variable): A CColumnSet structure containing the property offsets for properties in CPidMapper that are returned as a column.

B - CRestrictionPresent (1 byte): A byte field indicating if the **Restriction** field is present.

Note If set to any nonzero value, the **Restriction** field MUST be present. If set to 0x00, **Restriction** MUST be absent.

Reserved2 (2 bytes): A 16-bit reserved field. MUST contain the value 0x0101.

Align1 (variable): A field structure of 0, 1, 2 or 3 bytes that is used to align the next field to a 4-byte boundary. MUST be ignored by the protocol server.

Restriction (variable): A CRestriction structure containing the command tree of the search query.

C - CSortSetPresent (1 byte): A byte field indicating if the **SortSet** field is present.

Note If set to any nonzero value, the **SortSet** field MUST be present. If set to 0x00, **SortSet** MUST be absent.

Align2 (variable): A field structure of 0, 1, 2 or 3 bytes that is used to align the next field to a 4-byte boundary. MUST be ignored by the protocol server.

SortSet (variable): A CSortSet structure indicating the sort order of the search query.

Reserved0 (1 byte): An 8-bit reserved field reserved for future use. MUST be ignored by the protocol server.

Align3 (variable): A field structure of 0, 1, 2 or 3 bytes that is used to align the next field to a 4-byte boundary. MUST be ignored by the protocol server.

RowSetProperties (variable): A CRowsetProperties structure providing configuration information for the search query.

PidMapper (variable): A **CPidMapper** structure that maps from property offsets to full property descriptions.

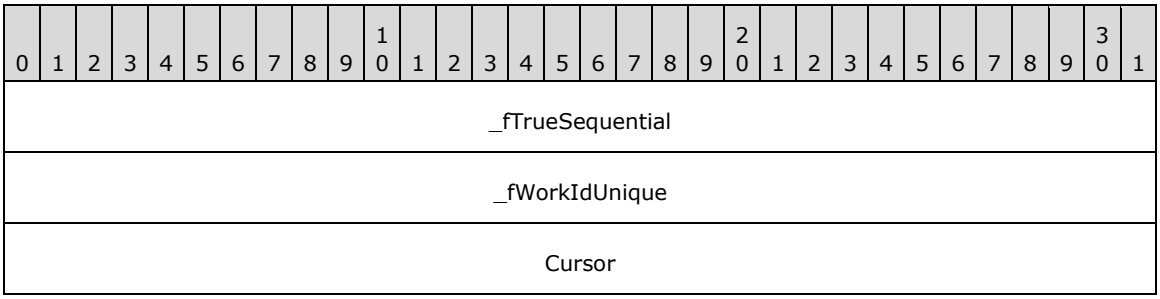
Reserved1 (4 bytes): A 32-bit reserved field. MUST be set to 0x00000000.

LCID (4 bytes): A 32-bit unsigned integer, indicating the LCID of the search query, as specified in [MS-LCID].

2.2.3.4 CPMCreateQueryOut

The **CPMCreateQueryOut** message contains a response to a CPMCreateQueryIn message.

The format of the **CPMCreateQueryOut** message that follows the header is shown in the following diagram.



_fTrueSequential (4 bytes): A 32-bit unsigned integer that indicates whether the protocol server can use the search catalog in such a way that query results will likely be delivered faster. This field MUST be set to one of the values in the following table.

Value	Meaning
0x00000000	For the search query provided in **CPMCreateQueryIn**, there would be a bigger latency in delivering query results.
0x00000001	For the search query provided in **CPMCreateQueryIn**, the protocol server can use the search catalog in such a way that query results will likely be delivered faster.

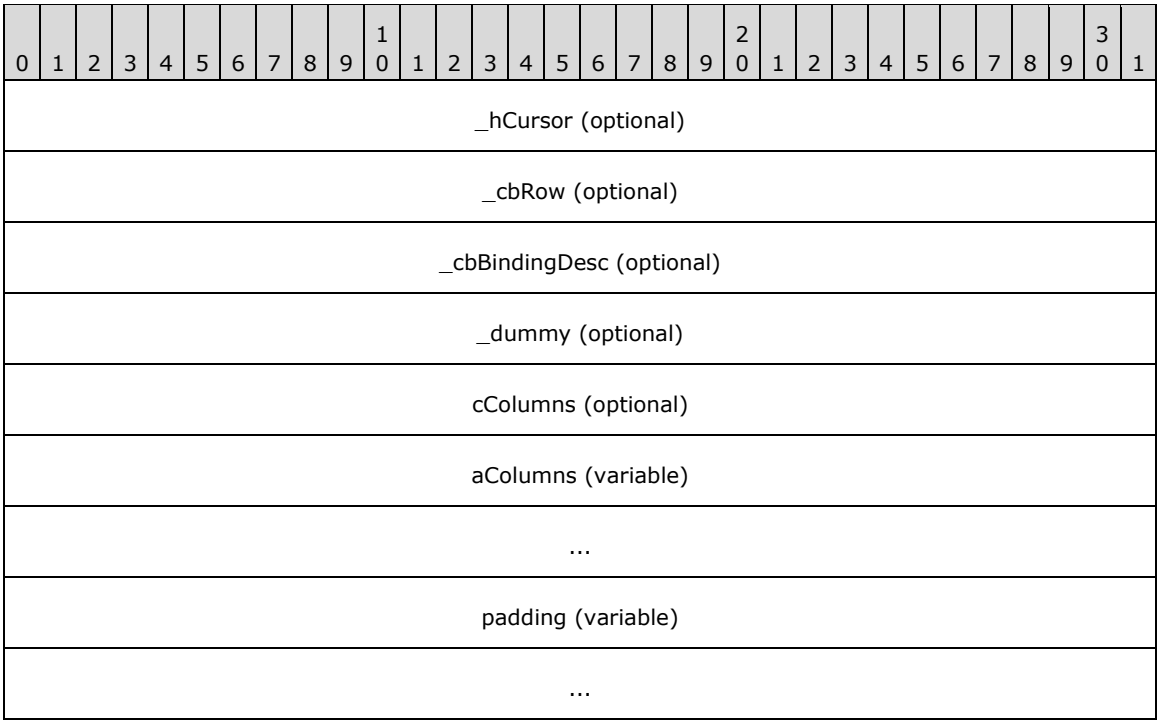
_fWorkIdUnique (4 bytes): A Boolean value indicating if the document identifiers pointed to by the cursors are unique throughout query results. MUST be set to one of the following values.

Value	Meaning
0x00000000	The document identifiers are unique only throughout the rowset.
0x00000001	The document identifiers are unique across multiple query results.

Cursor (4 bytes): A 32-bit unsigned integer representing the **handle** to the cursor that identifies the query being executed.

2.2.3.5 CPMSetBindingsIn

The **CPMSetBindingsIn** message requests the binding of columns to a rowset. The protocol server will reply to the **CPMSetBindingsIn** request message using the header section of the **CPMSetBindingsIn** message with the results of the request contained in the **_status** field. The format of the **CPMSetBindingsIn** message that follows the header is shown in the following diagram.



_hCursor (4 bytes, optional): A 32-bit value representing the handle from the CPMCreateQueryOut message that identifies the search query for which to set bindings. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

_cbRow (4 bytes, optional): A 32-bit unsigned integer indicating the size in bytes of a row. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

_cbBindingDesc (4 bytes, optional): A 32-bit unsigned integer indicating the length in bytes of the fields following the _dummy field. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

_dummy (4 bytes, optional): This field is unused, and MUST be ignored. It can be set to any arbitrary value. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

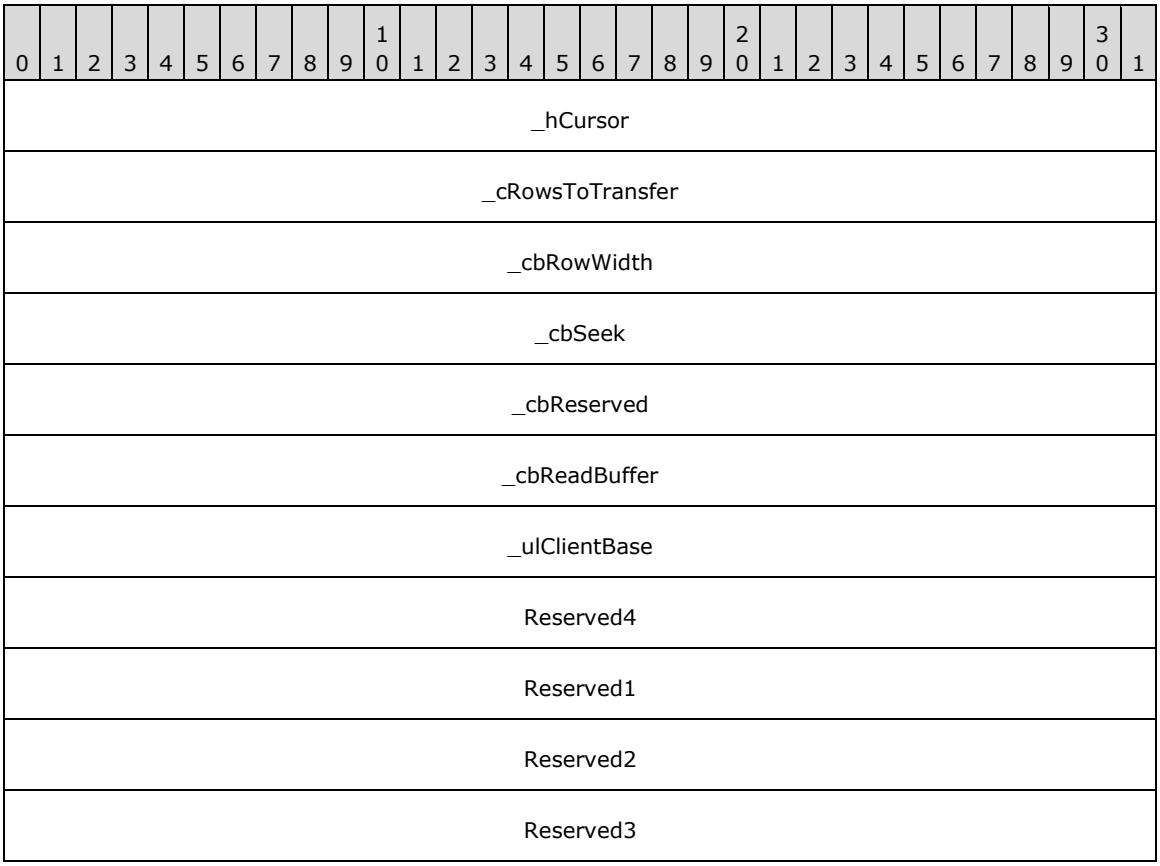
cColumns (4 bytes, optional): A 32-bit unsigned integer indicating the number of elements in the **aColumns** array. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

aColumns (variable): An array of the CTableColumn structures describing the columns of a row in the rowset. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure has a 4-byte alignment from the beginning of a message. Such padding bytes can be any arbitrary value, and MUST be ignored on receipt.

padding (variable): This field MUST be of the length necessary (0 to 3 bytes) to pad the message out to a multiple of 4 bytes in length. The value of the padding bytes can be any arbitrary value. This field MUST be ignored by the receiver. This field MUST be present when the message is sent by the protocol client, and MUST be absent when the message is sent by the protocol server.

2.2.3.6 CPMGetRowsIn

The **CPMGetRowsIn** message requests rows from a search query. The format of the **CPMGetRowsIn** message that follows the header is shown in the following diagram.



_hCursor (4 bytes): A 32-bit value representing the handle from the CPMCreateQueryOut message identifying the search query for which to retrieve rows.

_cRowsToTransfer (4 bytes): A 32-bit unsigned integer indicating the maximum number of rows the protocol client expects to receive in response to this message.

_cbRowWidth (4 bytes): A 32-bit unsigned integer indicating the length of a row in bytes.

_cbSeek (4 bytes): A 32-bit reserved field. MUST be set to 0x0000000C.

_cbReserved (4 bytes): A 32-bit unsigned integer indicating the offset, in bytes, of the Rows field in the CPMGetRowsOut response message. This offset begins from the first byte of the message header and MUST be set such that the Rows field follows the Reserved1 field.

_cbReadBuffer (4 bytes): A 32-bit unsigned integer. Note This field MUST be set to the maximum of the following two values rounded up to the nearest 512-byte multiple: the value of **_cbRowWidth**, or 1,000 times the value of **_cRowsToTransfer**. The value MUST NOT exceed 0x00004000.

_ulClientBase (4 bytes): A 32-bit unsigned integer indicating the base value to use for pointer calculations in the row buffer. If 64-bit offsets are being used, the **_ulReserved2** field of the message header is used as the upper 32 bits, and **_ulClientBase** is used as the lower 32 bits of a 64-bit value. See section 2.2.3.7.

Reserved (4 bytes)4: A 32-bit reserved field. MUST be set to 0x00000000.

Reserved1 (4 bytes): A 32-bit reserved field. MUST be set to 0x00000001.

Reserved2 (4 bytes): A 32-bit reserved field. MUST be set to 0x00000000.

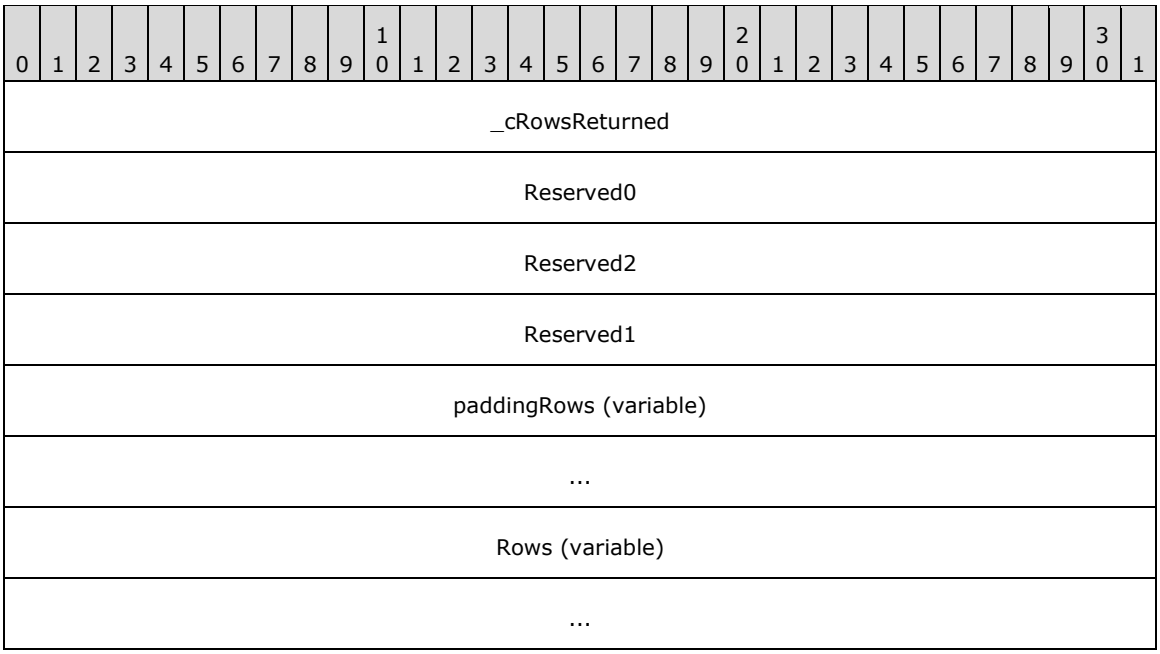
Reserved3 (4 bytes): A 32-bit reserved filed. MUST be set to 0x00000000.

2.2.3.7 CPMGetRowsOut

The **CPMGetRowsOut** message replies to a CPMGetRowsIn message with the rows of a search query. Protocol servers MUST format offsets to variable-length data types in the row field as follows.

- The protocol client indicated it was a 32-bit system (0x00000102 or 0x00000103 in the **_iClientVersion** field of CPMConnectIn): Offsets are 32-bit integers.
- The protocol client indicated it was a 64-bit system (**_iClientVersion** set to 0x00010102 or 0x00010103 in **CPMConnectIn**), and the protocol server indicated that it was a 32-bit system (**_serverVersion** set to 0x00000102 or 0x00000103 in CPMConnectOut): Offsets are 32-bit integers.
- The protocol client indicated it was a 64-bit system (**_iClientVersion** set to 0x00010102 or 0x00010103 in **CPMConnectIn**), and the protocol server indicated that it was a 64-bit system (**_serverVersion** set to 0x00010102 or 0x00010103 in **CPMConnectOut**): Offsets are 64-bit integers.

The format of the **CPMGetRowsOut** message that follows the header is depicted in the following diagram.



_cRowsReturned (4 bytes): A 32-bit unsigned integer indicating the number of rows returned in the Rows field.

Reserved0 (4 bytes): A 32-bit reserved field. MUST be set to 0x000000.

Reserved2 (4 bytes): A 32-bit reserved field. MUST be ignored by the receiver.

Reserved1 (4 bytes): A 32-bit reserved field. MUST be ignored by the receiver.

paddingRows (variable): This field MUST be of sufficient length (0 to **_cbReserved**-1 bytes) to pad the **Rows** field to **_cbReserved** offset from the beginning of a message where **_cbReserved** is the value in the **CPMGetRowsIn** message. Padding bytes used in this field can be any arbitrary value. This field MUST be ignored by the receiver.

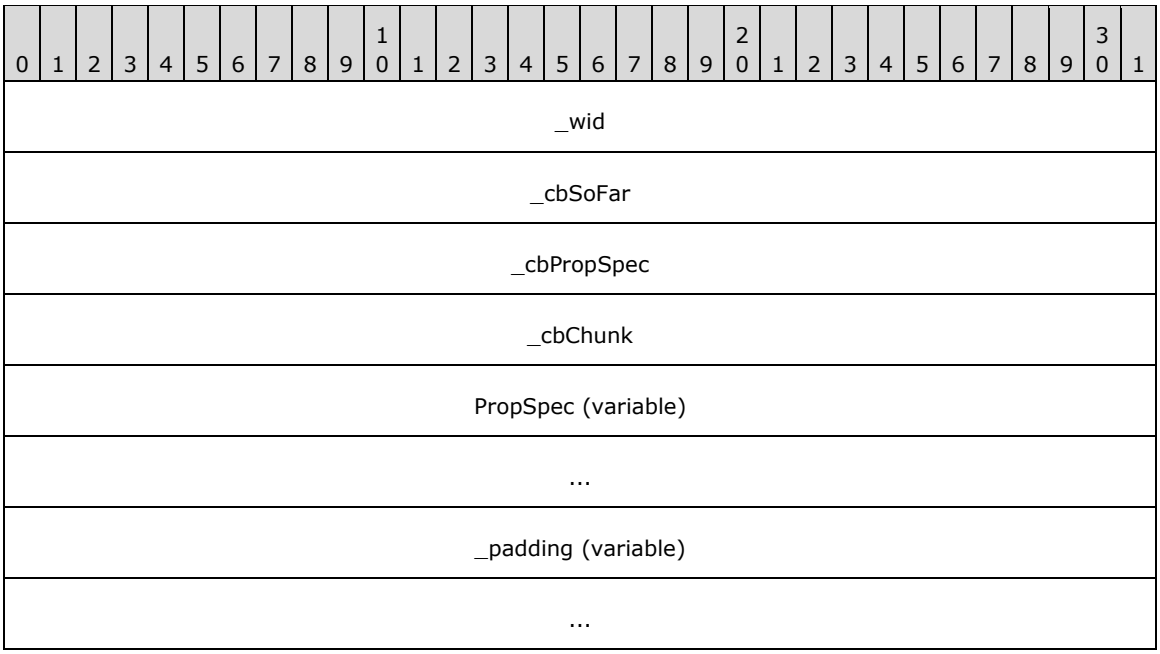
Rows (variable): Row data is formatted as prescribed by column information in the most recent CPMSetBindingsIn message. Rows MUST be stored in forward order (for example, row 1 before row 2). Fixed-sized columns MUST be stored at the offsets specified by the most recent **CPMSetBindingsIn** message.

Columns MUST be stored as CRowVariants with **vType** set to VT_I4 or VT_LPWSTR. Because the total size of the **Rows** field is specified by the **_cbReadBuffer** field of the CPMGetRowsIn message, if row data does not fit exactly into the **Rows** field of the **CPMGetRowsOut** message then there will be unused padding within the **Rows** field.

2.2.3.8 CPMFetchValueIn

The **CPMFetchValueIn** message requests metadata about the most recent search query initiated with a CPMCreateQueryIn message.

The format of the **CPMFetchValueIn** message that follows the header is shown in the following diagram.



_wid (4 bytes): A 32-bit reserved field. MUST be set to 0xFFFFFFFF.

_cbSoFar (4 bytes): A 32-bit unsigned integer containing the number of bytes previously transferred for this property. Note This field MUST be set to 0x00000000 in the first message.

_cbPropSpec (4 bytes): A 32-bit unsigned integer containing the size of the PropSpec field in bytes. The value MUST NOT be 0x00000000 for the first message. It MUST be 0x00000000 for subsequent messages.

_cbChunk (4 bytes): A 32-bit unsigned integer containing the maximum number of bytes that the sender can accept in a CPMFetchValueOut message<9>. The value of this field MUST be greater than 0x0000001C.

PropSpec (variable): A CFullPropSpec structure specifying the property to retrieve. If **_cbPropSpec** is not 0 then the following field values MUST be set on this structure, otherwise this structure MUST be omitted:

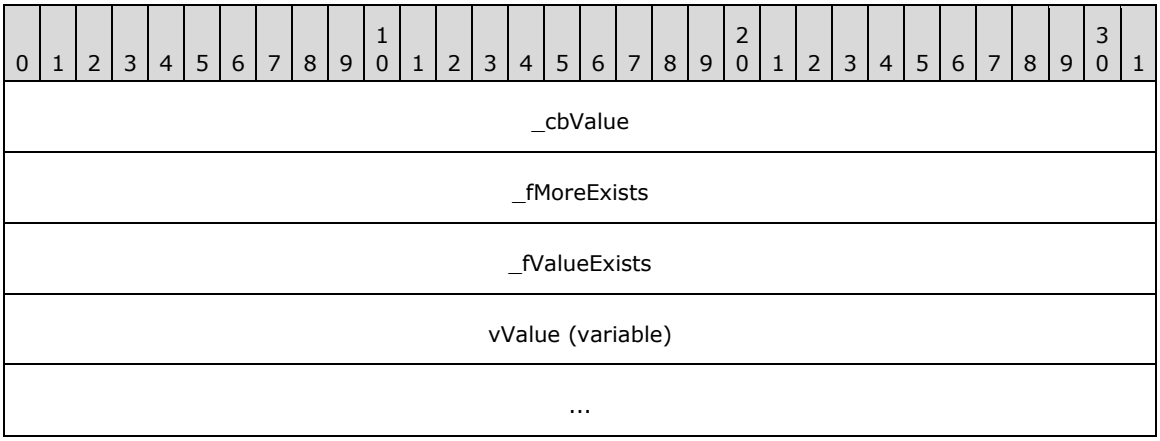
Field	Value
_guidPropSet	E83758B4-0C6E-435B-BCC6-268021EFAD6C
ulKind	PRSPEC_PROPID (0x00000001)
PrSpec	0x00000000

_padding (variable): This field MUST be of the length necessary (0 to 3 bytes) to pad the message out to a multiple of 4 bytes in length. The value of the padding bytes can be any arbitrary value. This field MUST be ignored by the receiver.

2.2.3.9 CPMFetchValueOut

The **CPMFetchValueOut** message replies to a CPMFetchValueIn message with a metadata about the most recent previously issued query. As specified in section 3.1.5.4, this message is sent after each **CPMFetchValueIn** message until all bytes of the metadata are transferred. The message length including header MUST be less than or equal to the value of **_cbChunk** specified in the **CPMFetchValueIn** message.

The format of the **CPMFetchValueOut** message that follows the header is shown in the following diagram.



_cbValue (4 bytes): A 32-bit unsigned integer containing the size in bytes of the **vValue** field.

_fMoreExists (4 bytes): A Boolean value indicating whether there are additional **CPMFetchValueOut** messages available. If this value is set to 0x00000001, the message size MUST be equal to the value of the **_cbChunk** field in **CPMFetchValueIn**.

Value	Meaning

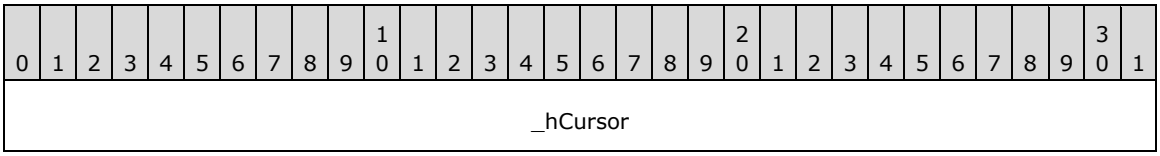
Value	Meaning
0x00000000	There is no additional data available.
0x00000001	There is additional data available.

_fValueExists (4 bytes): A reserved 32-bit unsigned integer. MUST be set to 0x00000001.

vValue (variable): A portion of a byte array containing a QUERYMETADATA where the offset of the beginning of the portion is the value of **_cbSoFar** in **CPMFetchValueIn**.

2.2.3.10 CPMFreeCursorIn

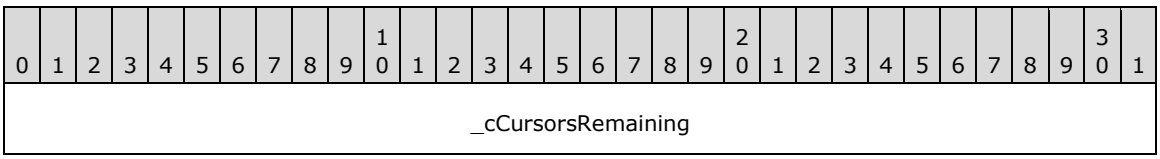
The **CPMFreeCursorIn** message requests the release of a cursor. The format of the **CPMFreeCursorIn** message that follows the header is shown in the following diagram.



_hCursor (4 bytes): A 32-bit value representing the handle of the cursor from the CPMCreateQueryOut message to release.

2.2.3.11 CPMFreeCursorOut

The **CPMFreeCursorOut** message replies to a CPMFreeCursorIn message with the results of freeing a cursor. The format of the **CPMFreeCursorOut** message that follows the header is shown in the following diagram.



_cCursorsRemaining (4 bytes): A 32-bit unsigned integer indicating the number of cursors still in use for the search query.

2.2.3.12 CPMDisconnect

The **CPMDisconnect** message ends the connection with the server.

The message MUST NOT include a body; only the message header (as specified in section 2.2.2) is to be sent.

2.2.4 Errors

All MSSearch Query Protocol messages MUST return a successful HRESULT code on success; otherwise, they return a 32-bit nonzero error code that can be either an HRESULT or an NTSTATUS value (see section 1.8).

All error values MUST be treated the same; the error MUST be considered fatal and reported to the higher-level caller. Future messages MAY be sent over the same pipe as if no error had occurred<10>.

3 Protocol Details

MSSearch Query Protocol message requests require only minimal sequencing. All messages MUST be preceded by an initial CPMConnectIn message (for example, at least one **CPMConnectIn** for each named pipe connection). Beyond the initial connection, there is no other sequencing required by the protocol. However, it is advised that the higher layer adhere to a meaningful message sequence; and for messages that are received out of this sequence or with invalid data, the protocol server will respond with an error. Note that some messages are also dependent on the higher layer, providing valid data that was received in messages earlier in the sequence.

A typical message sequence for a simple search query from a protocol client to a remote computer is illustrated in the following diagram.

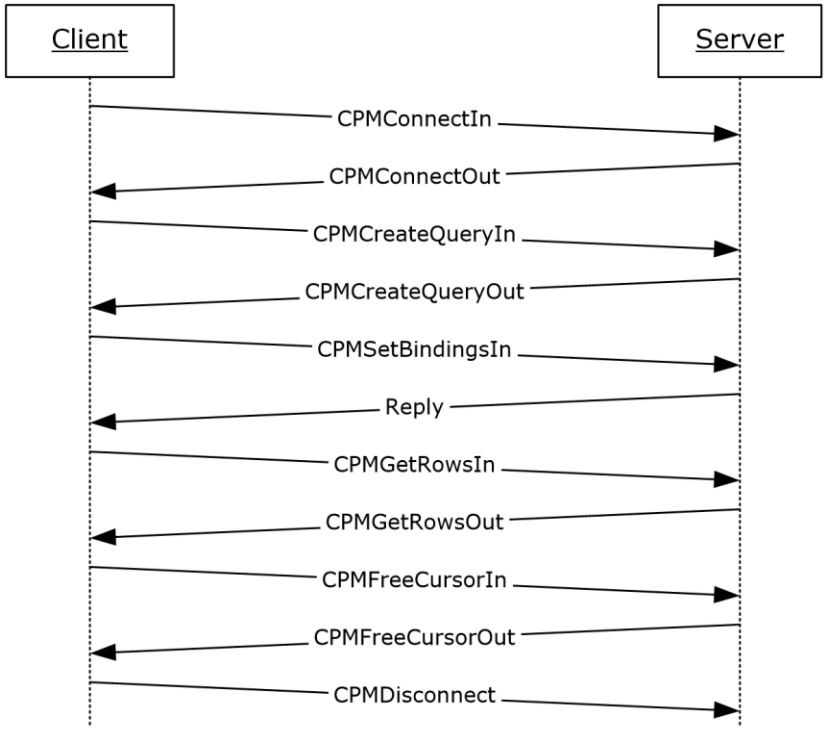


Figure 1: Typical message sequence for a simple query from protocol client to remote computer

The messages represented in the preceding diagram represent a subset of all of the MSSearch Query Protocol messages used for querying a remote query server search catalog.

3.1 Server Details

3.1.1 Abstract Data Model

The following section specifies data and state maintained by the MSSearch Query Protocol server. The data provided in this document explains how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

A query server implementing the MSSearch Query Protocol maintains the following abstract data elements:

- The list of protocol clients connected to the protocol server.
- Information about each protocol client, which includes:
  - Protocol client's version (as specified in the CPMConnectIn message.
  - Search catalog associated with the protocol client (by a **CPMConnectIn** message).
  - Additional client properties as specified in the database properties.
  - Protocol client's search query.
- List of cursor handles for the search query and position in result set for each handle.
- Current set of bindings.
- Current status of the search query which includes (for each cursor):
  - Number of rows in query result.
  - Numerator and denominator of query completion.
- The current state of the query server, which is one of "not initialized", "running", or "shutting down".

Note that most of the time the protocol server is in "running" state. The following is the state machine diagram for the protocol server.

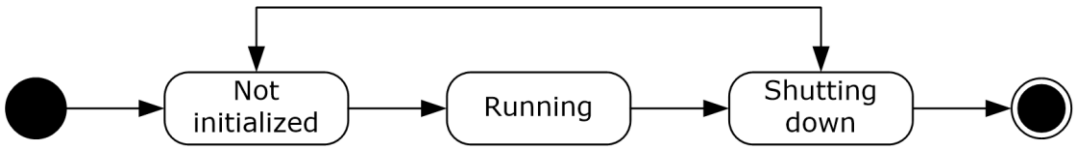


Figure 2: State machine diagram for the protocol server

3.1.2 Timers

None.

3.1.3 Initialization

Upon initialization, the protocol server MUST set its state to "not initialized" and begin listening for messages on the named pipe specified in section 1.9. After doing any other internal initialization, it MUST transition to the "running" state.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

Whenever an error occurs during processing of a message sent by a protocol client, the protocol server MUST report an error back to the protocol client as follows:

1. Stop processing the message sent by the protocol client.

2. Respond with the message header (only) of the message sent by the protocol client, keeping **_msg** field intact.

3. Set the **_status** field to the error code value.

When a message arrives, the protocol server MUST check the field value to see if it is a known type (see section 2.2.2). If the type is not known, it MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error. The protocol server MUST then validate the **_ulChecksum** field value if the message type is one of the following:

- CPMConnectIn (0x000000C8)
- CPMCreateQueryIn (0x000000CA)
- CPMSetBindingsIn (0x000000D0)
- CPMGetRowsIn (0x000000CC)
- CPMFetchValueIn (0x000000E4)

The protocol server MUST validate that the **_ulChecksum** field was calculated as specified in section 3.2.4. If the **_ulChecksum** value is invalid, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

Next, the protocol server checks which state is it in. If its state is "not initialized", the protocol server MUST report a CI_E_NOT_INITIALIZED (0x8004180B) error. If the state is "shutting down", the protocol server MUST report a CI_E_SHUTDOWN (0x80041812) error.

After a header has been determined to be valid and the protocol server to be in "running" state, further message-specific processing MUST be done as specified in the following subsections.

Some messages are only valid after a previous message has been sent. An identifier or handle from the earlier message can be required as input to the later message. These requirements are detailed in the following sections. In the following table that summarizes the relationship between messages, an X mark means that the protocol client MUST NOT send the message specified in the row before it received the response specified in the column.

	CPMConnectOut	**CPMCreateQueryOut**	**CPMSetBindingsIn**	**CPMGetRowsOut**	**CPMFetchValueOut**	**CPMFreeCursorOut**
CPMConnectIn						
CPMCreateQueryIn	X					
CPMSetBindingsIn	X	X				
CPMGetRowsIn	X	X	X			
CPMFetchValueIn	X	X	X			
CPMFreeCursorIn	X	X				

3.1.5.1 Receiving a CPMConnectIn Request

When the protocol server receives a CPMConnectIn request from a protocol client, the protocol server MUST do the following:

1. Check if the protocol client is in the list of connected clients. If this is the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Checks if the specified search catalog exists and not in the stopped state. If this is not the case, the protocol server MUST report a MSS_E_CATALOGNOTFOUND (0x80042103) error.

3. Add the protocol client to the list of connected clients.

4. Associate the search catalog with the protocol client.

5. Store the information passed in the **CPMConnectIn** message (such as search catalog name or protocol client version) in the protocol client state.

6. Respond to the protocol client with a CPMConnectOut message.

3.1.5.2 Receiving a CPMCreateQueryIn Request

When the protocol server receives a CPMCreateQueryIn message request from a protocol client, the protocol server MUST do the following:

1. Check if the protocol client is in the list of connected clients. If this is not the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check if the protocol client already has a search query associated with it. If this is the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

3. Parse the **restriction** set, sort orders, and groupings that are specified in the search query. If the protocol server encounters an error, it MUST report an appropriate error. If this step fails for any other reason, the protocol server MUST report the error encountered. For information about query server query errors, see [MSDN-QUERYERR].

4. Save the search query in the state for the protocol client.

5. Make any preparations required to serve rows to a protocol client and associate the search query with new cursor handles. The cursor handles MUST be returned to the protocol client in the CPMCreateQueryOut response.

6. Initialize the number of rows to the currently calculated number of rows (which can be 0 if the search query did not begin to execute or some number if the search query is in a process of execution), and initialize the numerator and denominator of search query completion.

7. Respond to the protocol client with a CPMCreateQueryOut message.

3.1.5.3 Receiving a CPMSetBindingsIn Request

When the protocol server receives a **CPMSetBindingsIn** message request from a protocol client, the protocol server MUST do the following:

1. Check if the protocol client has a query associated with it. If this is not the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check if the cursor handle passed is in the list of the protocol client's cursor handles. If this is not the case, the protocol server MUST report an E_INVALIDARG (0x80070057) error.

3. Verify that binding information is valid (that is, the column at least specifies value, length or status to be returned; no overlap in bindings for value, length or status; and value, length and status fit in the row size specified) and if not, report a DB_E_BADBINDINFO (0x80040E08) error.

4. Save the binding information associated with the columns specified in the aColumns field. If this step fails for any reason, the protocol server MUST report that an error was encountered.

5. Respond to the protocol client with a message header (only) with _msg set to CPMSetBindingsIn, and _status set to the results of the specified binding.

3.1.5.4 Receiving a CPMFetchValueIn Request

When the protocol server receives a CPMFetchValueIn message request from a protocol client, the protocol server MUST do the following:

1. Check if the protocol client has a query associated with it. If this is not the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Prepare a CPMFetchValueOut message. If this step fails for any reason, the protocol server MUST report the error encountered, which is an **HRESULT** or an NTSTATUS value (see section 1.8).

3. Set **_fValueExists** to 0x00000001.

4. Set **vType** to 0x41 (VT_BLOB).

The protocol server MUST store the ignored terms of the search query into a CBaseStorageVariant with **vType** VT_VECTOR | VT_LPWSTR. The server MUST use VT_VECTOR | VT_LPWSTR with zero elements if there were no ignored terms.

The protocol server MUST store any spelling suggestions of the query terms into a **CBaseStorageVariant** with **vType** VT_LPWSTR. MUST contain space-delimited keywords and any keywords which are spelling suggestions MUST be prefixed with "<suggestion>" and post fixed with "</suggestion>". If there are no spelling suggestions then vValue MUST contain a null-terminated empty VT_LPWSTR.

The protocol server MUST store the query terms into a **CBaseStorageVariant** with **vType** VT_VECTOR | VT_LPWSTR. The protocol server MUST use VT_VECTOR | VT_LPWSTR with zero elements if there were no query terms.

For each query term the protocol server MUST determine a term identifier or 0x0000000. The protocol server MUST store the term identifiers into a **CBaseStorageVariant** with **vType** VT_VECTOR | VT_UI4.

The protocol server MUST serialize the estimated total number of results for the search query into a 32-bit value. The protocol server MUST then:

1. Serialize the values of the **CBaseStorageVariants** and 32-bit value from steps 0-0 to a QUERYMETADATA structure and copy, starting from the **_cbSoFar** offset, at most **_cbChunk** bytes (but not past the end of the serialized property) to **vValue** field. If this step fails for any reason, the protocol server MUST report an error.

2. Set **_cbValue** to the size of the sent.

3. If the length of serialized property is greater than **_cbSoFar** added to **_cbValue**, set **_fMoreExists** to 0x00000001; otherwise, set it to 0x00000000.

4. Respond to the protocol client with the **CPMFetchValueOut** message.

3.1.5.5 Receiving a CPMGetRowsIn Request

When the protocol server receives a CPMGetRowsIn message request from a protocol client, the protocol server MUST do the following:

1. Check if the protocol client has a query associated with it. If this is not the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check if the cursor handle passed is in the list of the protocol client's cursor handles. If this is not the case, the protocol server MUST report an E_INVALIDARG (0x80070057) error.

3. Check if the protocol client has a current set of bindings. If this is not the case, the protocol server MUST report an E_UNEXPECTED (0x8000FFFF) error.

4. Prepare a CPMGetRowsOut message. The protocol server MUST position the cursor at the beginning of the query results. If this step fails for any reason, the protocol server MUST report the error encountered, which is an **HRESULT** or an NTSTATUS value (see section 1.8).

5. Fetch as many rows as will fit in a buffer, the size of which is indicated by **_cbReadBuffer**, but not more than indicated by **_cRowsToTransfer**. When fetching rows, the protocol server MUST compare each selected column's property value type to the type that is specified in the protocol client's current set of bindings (see section 3.1.1). Store the actual number of rows fetched in **_cRowsReturned**.

6. Store fetched rows in the **Rows** field (see section 2.2.3.7 on the structure of the **Rows** field).

7. Respond to the protocol client with the CPMGetRowsOut message.

3.1.5.6 Receiving a CPMFreeCursorIn Request

When the protocol server receives a CPMFreeCursorIn message request from the protocol client, the protocol server MUST do the following:

1. Check if the protocol client has a query associated with it. If this is not the case, the protocol server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check if the cursor handle passed is in the list of the protocol client's cursor handles. If this is not the case, the protocol server MUST report an E_INVALIDARG (0x80070057) error.

3. Release the cursor and associated resources (see section 3.1.1 for a complete list) for this cursor handle.

4. Remove the cursor from the list of cursors for that protocol client.

5. Respond with a CPMFreeCursorOut message, setting the **_cCursorsRemaining** field with the number of cursors remaining in this protocol client's list.

If there are no more cursors for this protocol client, the protocol server MUST release the query and associated resources (see section 3.1.1).

3.1.5.7 Receiving a CPMDisconnect Request

When the protocol server receives a CPMDisconnect message request from the protocol client, the protocol server MUST remove the protocol client from the list of connected protocol clients and release all resources associated with the protocol client.

3.1.6 Timer Events

When the protocol server receives a CPMConnectIn request with a nonzero value in the **cCmdTimeout** field of CRowsetProperties then the protocol server MUST use a timer event to interrupt a search query that runs longer than the value specified by **cCmdTimeout**..

3.1.7 Other Local Events

When the protocol server is stopped, it MUST first transition to the "shutting down" state. It MUST then stop listening to the pipe, perform any other implementation-specific shutdown tasks, and then transition into the "stopped" state.

3.2 Client Details

3.2.1 Abstract Data Model

The following section describes data and state maintained by the MSSearch Query Protocol client. The data is provided to help explain how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this document.

A protocol client has the following abstract state:

- **Last Message Sent**: A copy of the last message sent to the protocol server.
- **Current Property Value**: A partial value of a deferred property, which is in the process of being retrieved.
- **Current Bytes Received**: The number of bytes received for Current Property Value so far.
- **Named Pipe Connection State**: A connection to the protocol server.

3.2.2 Timers

None.

3.2.3 Initialization

No actions are taken until a higher-layer request is received.

3.2.4 Higher-Layer Triggered Events

When a request is received from a higher layer, the protocol client MUST create a named pipe connection to the protocol server, using the details specified in section 2.1. If it is unable to do so, the higher-layer request MUST be failed. That is, in case of a failure to connect, it is the responsibility of the higher level to retry.

A header MUST be prepended with fields set as specified in section 2.2.2.

For messages that are specified as requiring a nonzero checksum, the **_ulChecksum** value MUST be calculated as follows:

1. The content of the message after the **_ulReserved2** field in the message header MUST be interpreted as a sequence of 32-bit integers. The protocol client MUST calculate the sum of the numeric values given by these integers.
2. Calculate the bitwise XOR of this value with 0x59533959.
3. Subtract the value given by **_msg** from the value that results from the bitwise XOR.

3.2.4.1 Query Server Query Messages

With the exception of CPMGetRowsIn/CPMGetRowsOut and CPMFetchValueIn/CPMFetchValueOut, there is a one-to-one relationship between MSSearch Query Protocol messages and higher-layer requests. For the two exceptions previously mentioned, there can be multiple messages generated by the protocol client to either satisfy size requirements or to retrieve a complete property. The higher layer SHOULD keep track of all query-specific information (such as cursor handles opened and **_wid** values for deferred property values) and also track if the protocol client is in a connected state, but this is not enforced in any way by the protocol client.

The client portion of the diagram in section 3 illustrates this sequence for a simple query.

3.2.4.1.1 Sending a CPMConnectIn Request

This message SHOULD be the very first request from the higher layer (if the protocol client is not connected, the protocol server will fail most of the messages). The higher level provides the protocol client with information necessary to connect. To serve the higher layer, the protocol client MUST do the following:

1. Fill in the message, using information that the higher layer client provided (see section 2.2.3.1) in **_iClientVersion**, **MachineName**, **UserName**, **PropertySets**, and **ExtPropertySets**.
2. Set **_fClientIsRemote**, **_cbBlob1**, **_cbBlob2**, **cPropSet**, and **cExtPropSet**, as specified in section 2.2.3.1.
3. Set the checksum in the **_ulChecksum** field.
4. Send the CPMConnectIn message to the protocol server.
5. Wait to receive a CPMConnectOut message back from the protocol server, silently discarding all other messages.
6. Report the value of the **_status** field of the response (and, if it was successful, the **_serverVersion**) back to the higher layer.

The higher layers SHOULD do the following actions on successful connection, but these are not enforced by the MSSearch Query Protocol client:

- Use a CPMCreateQueryIn request to create a search query with a purpose of retrieving results from the search catalog.

3.2.4.1.2 Sending a CPMCreateQueryIn Request

The higher layer SHOULD provide information for the query creation after the protocol client is connected. The higher layer provides the protocol client with a **restriction** set, columns set, sort order (can be omitted), rowset properties, and property identifier mapper structure. When this request is received from a higher layer, the protocol client MUST do the following:

1. Prepare a CPMCreateQueryIn as follows:
   - If a columns set is present, set **CColumnSetPresent** to 0x01 and fill the **ColumnsSet** field.
   - If restrictions are present, set **CRestrictionPresent** to 0x01 and fill the **Restriction** field.
   - If a sort set is present, set **CSortSetPresent** to 0x01 and fill the **SortSet** field.
   - Set the rest of fields as specified in section 2.2.3.3.
   - Calculate the **_ulCheckSum** field in the header.

2. Send the **CPMCreateQueryIn** message to the protocol server.

3. Wait to receive the CPMCreateQueryOut message (see section 3.2.5.1), silently discarding all other messages.

4. Report the value of the **_status** field of the response (and, if it was successful, the array of cursor handles and informative Boolean values, as specified in section 2.2.3.3) back to the higher layer.

3.2.4.1.3 Sending a CPMSetBindingsIn Request

The higher layer SHOULD set bindings for each column to be returned in the rows when it already has a valid cursor handle (after successfully receiving CPMCreateQueryOut, see section 3.2.5.1). The higher layer is expected to provide an array of CTableColumn structures for the **aColumns** field and a valid cursor handle. When this request is received from the higher layer, the protocol client MUST do the following:

1. Calculate the number of **CTableColumn** structures in the **aColumns** array and set the **cColumns** field to this value.

2. Calculate the total size in bytes of the **cColumns** and **aColumns** fields and set the **_cbBindingDesc** field to this value.

3. Set specified fields in the CPMSetBindingsIn message to the values provided by the higher application layer. Set the ulChecksum field to the value calculated as specified in section 3.2.5.

4. Send the finished **CPMSetBindingsIn** message to the protocol server.

5. Wait to receive a **CPMSetBindingsIn** message from the protocol server, discarding other messages.

6. Indicate the status from the **_status** field of the response to the higher layer.

The higher layers SHOULD then request a protocol client to send a CPMGetRowsIn message, but this is not enforced by the MSSearch Query Protocol.

3.2.4.1.4 Sending a CPMGetRowsIn Request

When the higher layer is about to receive rows data, it will provide the protocol client with a valid cursor. The higher layer SHOULD do so when it has a valid cursor, and the bindings had been set with a CPMSetBindingsIn message.

When this request is received from the higher layer, the protocol client MUST do the following:

1. Determine what unsigned integer value to specify for the **_cbReadBuffer** field. To determine this value, the client SHOULD take the maximum value from the following:

   - One thousand times the value of the **_cRowsToTransfer** field, rounded up to the nearest 512-byte multiple.
   - Value of **_cbRowWidth**, rounded up to the nearest 512-byte multiple.
   - Take the higher of these two values, up to the 16-kilobytes limit.
   - In cases where a single row is larger than 16 kilobytes, the protocol server cannot return results to this query.

2. Specify a client base for variable-sized row data in the client address space in **_ulClientBase** field<11>.

3. Set the value of **_cbSeek** (which would act as an offset for **Rows** start) to 0x0000000C.

4. Send a CPMGetRowsIn message to the protocol server.

3.2.4.1.5 Sending a CPMFetchValueIn Request

If this is the first CPMFetchValueIn message the protocol client has sent to request the specified property, the protocol client MUST do the following:

1. Set all the fields in a message, as specified in section 2.2.3.8.
2. Set **_cbSoFar** to 0x00000000.
3. Set Current Bytes Received to 0.
4. Send the **CPMFetchValueIn** message to the server.

3.2.4.1.6 Sending a CPMFreeCursorIn Request

After the higher level is no longer using the search query, it can release the resources on the protocol server by asking the protocol client to send a CPMFreeCursorIn message.

When this request is received, the protocol client MUST send a **CPMFreeCursorIn** message to the protocol server, containing the handle specified by the upper layer.

The protocol client MUST do the following:

1. Send the finished **CPMFreeCursorIn** message to the protocol server.
2. Wait to receive a CPMFreeCursorOut message from protocol server, discarding other messages.

3.2.4.1.7 Sending a CPMDisconnect Message

If the higher layer has no more queries for the query server, the application can request that the protocol client send a CPMDisconnect message to the protocol server to make more server resources available. When the application makes the request, the protocol client MUST simply send the message as requested. There is no response to this message from the protocol server.

3.2.5 Message Processing Events and Sequencing Rules

When the protocol client receives a message response from the protocol server, the protocol client MUST use the Last Sent Message to determine if the message received from the protocol server is the one expected by the protocol client. All messages with the **_msg** field different from the one in Last Sent Message MUST be ignored.

3.2.5.1 Receiving a CPMCreateQueryOut Response

When the protocol client receives a CPMCreateQueryOut message response from the protocol server, the protocol client MUST return **_status** (and, if the status is successful, cursor handle values) back to the higher layer. Any further actions are up to the higher layer.

For informative purposes, it is expected that higher layers can do the following actions, but these are not enforced by the MSSearch Query Protocol client.

Use CPMSetBindingsIn to set bindings for individual columns and do any subsequent actions on the query path.

3.2.5.2 Receiving a CPMFetchValueOut Response

When the protocol client receives a CPMFetchValueOut message response from the protocol server, the protocol client MUST do the following:

1. Check if the **_status** field in the header indicates success or failure. In case of failure, notify the higher layer. Otherwise, continue to the next step.
2. Check **_fValueExist**, and, if set to 0x00000000, notify the higher layer that the value was not found.
3. Otherwise, append **_cbValue** bytes from **vValue** to current metadata.
4. If **_fMoreExists** is set to 0x00000001, increment Current Bytes Received by **_cbValue** and send a CPMFetchValueIn message to the server, setting **_cbSoFar** to the value of Current Bytes Received, **_cbPropSpec** to zero, and **_cbChunk** to the buffer size requested by the higher layer.
5. If **_fMoreExists** is set to 0x00000000, interpret the value of the QUERYMETADATA structure as specified in section 2.2.1.19 and report it to the higher level.

3.2.5.3 Receiving a CPMGetRowsOut Response

When the protocol client receives a CPMGetRowsOut message response from the protocol server, the protocol client MUST do the following:

1. Check if the **_status** field in the header indicates success or failure.
2. If the **_status** value is STATUS_BUFFER_TOO_SMALL (0xC0000023), the protocol client MUST check the Last Message Sent state. If it does not contain a **CPMGetRowsIn** message, the received message MUST be silently ignored. Otherwise, the **protocol** client MUST send to the protocol server a new **CPMGetRowsIn** message with all fields identical to the stored one except that the **_cbReadBuffer** MUST be increased by 512 (but not greater than 0x4000). If **_status** is STATUS_BUFFER_TOO_SMALL (0xC0000023), and Last Message Sent already has **_cbReadBuffer** equal to 0x4000, the protocol client MUST report the error up to the higher level.
3. If the **_status** value is any other error value, the protocol client MUST indicate the failure up to the higher layer.
4. If the **_status** value indicates success, the results MUST be indicated up to the higher layer requesting the information, and further actions are up to the higher layer.

Higher layers SHOULD do the following actions, but these are not enforced by the MSSearch Query Protocol client:

- The higher layer SHOULD store, display, or otherwise use the data from row values.

3.2.5.4 Receiving a CPMFreeCursorOut Response

When the protocol client receives a successful CPMFreeCursorOut message response from the protocol server, the protocol client MUST return the **_cCursorsRemaining** value to the higher layer.

The following information is given for informative purposes only and is not enforced by the MSSearch Query Protocol client. The higher layer is expected to keep track of cursor handles and not to use ones that have already been freed. Once the number of **_cCursorsRemaining** is equal to 0x00000000, the higher layer can use the connection to specify another query (using a CPMCreateQueryIn message).

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 Obtaining Document Identifiers Based on Query Text

In the following example, consider a scenario in which the user JOHN on computer A wants to obtain the document identifiers of files that contain the word "Microsoft" from the set of items stored on server X in catalog SYSTEM. Assume that both computer A and B are running a 32-bit Windows Server 2003 operating system.

1. The user launches a Search Query Protocol client application and enters the search query. The application in turn passes the search query to the protocol client.
2. The protocol client establishes a connection with indexing server X. The protocol client uses the named pipe \pipe\OSearch to connect to the server X over SMB.
3. The protocol client then prepares a CPMConnectIn message with the following values.
    1. The header of the message is populated as follows:
        - **_msg** is set to 0x000000C8, indicating that this is a **CPMConnectIn** message.
        - **_status** is set to 0x00000000.
        - **_ulChecksum** contains the checksum, computed as specified in section 3.2.4.
        - **_ulReserved2** is set to 0x00000000.
    2. The body of the message is populated as follows:
        - **_iClientVersion** is set to 0x00000102, indicating that the client is not capable of handling 64-bit offsets in CPMGetRowsOut messages.
        - **_fClientIsRemote** is set to 0x00000001, indicating that the server is a remote server.
        - **_cbBlob1** is set to the size in bytes of the **cPropSet** and **PropertySets** fields combined.
        - **_cbBlob2** is set to the size in bytes of the **cExtPropSet** and **ExtPropertySets** fields combined.
        - **MachineName** is set to A.
        - **UserName** is set to JOHN.
        - **cPropSets** is set to 0x00000001.
        - The **PropertySets**[0] field is of type CDbPropSet. The **CDbPropSet** structure comprising **PropertySets**[0] field is populated as follows:
            - The **GuidPropSet** field is set to A9BD1526-6A80-11D0-8C9D-0020AF1D740E (DBPROPSET_FSCIFRMWRK_EXT).
            - The **cProperties** field is set to 0x00000002.
            - The **aProps** field is an array of CDbProp structures.
                - For the **aProps**[0] element:
                    - **DBPROPID** is set to 0x00000002 (DBPROP_CI_CATALOG_NAME).
                    - **DBPROPOPTIONS** is set to 0x0000000.

- **DBPROPSTATUS** is set to 0x00000000.

- For the **ColId** element:
  - **eKind** is set to 0x00000001.
  - **GUID** is null (all zeros), meaning the value applies to the query, not just a single column.
  - **ulID** is set to 0x00000000.
- For the **vValue** element:
  - **vType** is set to 0x001F (VT_LPWSTR).
  - **vData1** is set to 0x00.
  - **vData2** is set to 0x00.
  - **vValue** is set to "SYSTEM", the name of the search catalog being requested.
- For the **aProps**[1] element:
  - **DBPROPID** is set to 0x00000007 (DBPROP_CI_QUERY_TYPE).
  - **DBPROPOPTIONS** is set to 0x0000000.
  - **DBPROPSTATUS** is set to0x00000000.
- For the **ColId** element:
  - **eKind** is set to 0x00000001.
  - **GUID** is null (all zeros), meaning the value applies to the query, not just a single column.
  - **ulID** is set to 0x00000000.
- For the **vValue** element:
  - **vType** is set to 0x0003 (VT_I4).
  - **vData1** is set to 0x00.
  - **vData2** is set to 0x00.
  - **vValue** is set to 0x00000000, meaning it is a regular query.

- The **cExtPropSet** field is set to 0x00000001.
- The **ExtPropertySets** [0] field is of type CDbPropSet. The **CDbPropSet** structure comprising **ExtPropertySets** [0] field is populated as follows:
  - The **GuidPropSet** field is set to A9BD1526-6A80-11D0-8C9D-0020AF1D740E (DBPROPSET_FSCIFRMWRK_EXT).
  - The **cProperties** field is set to 0x00000001.
  - The **aProps** field is an array of CDbProp structures.
    - For the **aProps**[0] element:
      - **DBPROPID** is set to 0x00000002 (DBPROP_CI_CATALOG_NAME)**.**

- **DBPROPOPTIONS** is set to 0x0000000.
- **DBPROPSTATUS** is set to 0x00000000.

- For the **ColId** element:
  - **eKind** is set to 0x00000001.
  - **GUID** is null (all zeros), meaning the value applies to the query, not just a single column.
  - **ulID** is set to 0x00000000.
- For the **vValue** element:
  - **vType** is set to 0x0008 (VT_BSTR).
  - **vData1** is set to 0x00.
  - **vData2** is set to 0x00.
  - **vValue** is set to "SYSTEM", the name of the search catalog being requested.

3. Various padding fields are filled in as needed. The message is sent to the protocol server.

4. The protocol server verifies that the **_ulChecksum** is correct, verifies that the user is authorized to make this request, and responds with a CPMConnectOut message.

   1. The header of the message is populated as follows:
      - **_msg** is set to 0x000000C8, indicating that this is a **CPMConnectOut** message.
      - **_status** is set to 0x0000000 indicating SUCCESS.
      - **_ulChecksum** is set to 0.
      - **_ulReserved2** is set to 0x00000000.
   2. The body of the message is populated as follows:
      - The **_serverVersion** field is set to 0x00000102.
      - The **_reserved** fields are filled with arbitrary data.

5. The protocol client prepares a CPMCreateQueryIn message.

   1. The header of the message is populated as follows:
      - **_msg** is set to 0x000000CA, indicating that this is a **CPMCreateQueryIn** message.
      - **_status** is set to 0x00000000.
      - **_ulChecksum** contains the checksum, computed according to section 3.2.4.
      - **_ulReserved2** is set to 0x00000000.
   2. The body of the message is populated as follows:
      - The **Size** field is set to the size of the rest of the message.
      - The **CColumnSetPresent** field is set to 0x01.

- The **ColumnSet** field is of type CColumnSet. The **CColumnSet** structure comprising this field is set as follows:
  - The **count** field is set to 0x00000001 indicating one column is returned.
  - The **indexes** array contains one element with value 0x00000000, indicating the first entry in **_aPropSpec**.
- The **CRestrictionPresent** field is set to 0x01, indicating the **Restriction** field is present.
- The **Restriction** field is of type CRestriction and is set to:
  - **_ulType** is set to 0x00000004 (RTContent).
  - **Weight** is set to 0x00000000.
  - The **Restriction** field contains a CContentRestriction structure:
    - **_Property** is set to GUID 012357BD-1113-171D-1F25-292BB0B0B0B0 / 0x00000001 / 0x00000001, which represents the document body on the particular protocol server implementation.
    - **Cc** is set to 0x00000009.
    - **_pwcsphrase** is set to the string "Microsoft".
    - **Lcid** is set to 0x00000409 (for English).
    - **_ulGenerateMethod** is set to 0x00000000 (exact match).
- **CSortSetPresent** is set to 0x00.
- **RowSetProperties** is set as follows:
  - **_uBooleanOptions** is set to 0x00008001 (sequential, ignore noise-only).
  - **_ulMaxOpenRows** is set to 0x00000000.
  - **_ulMemoryUsage** is set to 0x00000000.
  - **_cMaxResults** is set to 0x00000100 (return at most 256 rows).
  - **_cCmdTimeOut** is set to 0x00000000 (never time out).
- **PidMapper** is set to:
  - **count** is set to 0x00000001.
  - **_aPropSpec** is set to GUID 012357BD-1113-171D-1F25-292BB0B0B0B0 / 0x00000001 / 0x0000002F, which represents the document identifier property on the particular protocol server implementation.
- **Reserved1** is set to 0x00000000.
- **LCID** is set to 0x409 (for English).

6. The protocol server processes it and responds with a CPMCreateQueryOut message.
   1. The header of the message is populated as follows:
      - **_msg** is set to 0x000000CA, indicating that this is a **CPMCreateQueryOut** message.
      - **_status** is set to SUCCESS.

- **_ulChecksum** is set to 0x00000000 (or any other arbitrary value).
- **_ulReserved2** is set to 0x00000000.

2. The body of the message is populated as follows:
   - **_fTrueSequential** is set to 0x00000000.
   - **_fWorkIdUnique** is set to 0x00000001.
   - The **Cursor** field contains a cursor handle to this query. The value depends on the state of the protocol server, assuming that the returned value is 0xAAAAAAAA.

7. The protocol client issues a CPMSetBindingsIn request message to define the format of a row.
   1. The header of the message is populated as follows:
      - **_msg** is set to 0x000000D0, indicating that this is a **CPMSetBindingsIn** message.
      - **_status** is set to SUCCESS.
      - **_ulChecksum** contains the checksum, computed according to section 3.2.4.
      - **_ulReserved2** is set to 0x00000000.
   2. The body of the message is populated as follows:
      - **_hCursor** is set to 0xAAAAAAAA.
      - **_cbRow** is set to 0x10 (big enough to fit columns).
      - **_cbBindingDesc** is set to the size of the **_cColumns** and **_aColumns** fields combined.
      - **_dummy** is set to 0x00000000 (or any other arbitrary value).
      - **_cColumns** is set to 0x00000001.
      - The **_aColumns** array is set to contain one CTableColumn structure containing:
        - **_PropSpec** is set to GUID 012357BD-1113-171D-1F25-292BB0B0B0B0 / 0x00000001 / 0x0000002F, which represents the document identifier property on the particular server implementation.
        - **_vType** is set to 0x000C (VT_VARIANT).
        - **_ValueUsed** is set to 0x01 (column transferred in row).
        - **_ValueOffset** is set to 0x0008 (at beginning of row).
        - **_ValueSize** is set to 0x10 (size of a **CRowVariant**).
        - **_StatusUsed** is set to 0x01.
        - **_StatusOffset** is set to 0x02.
        - **_LengthUsed** is set to 0x01.
        - **_LengthOffset** is set to 0x04.

8. The protocol server processes it and responds with a CPMSetBindingsIn message.
   1. The header of the message is populated as follows:

- **_msg** is set to 0x000000D0.
- **_status** is set to SUCCESS.
- **_ulChecksum** is set to 0x00000000 (or any other arbitrary value).
- **_ulReserved2** is set to 0x00000000.

9. The protocol client issues a CPMGetRowsIn request message, assuming that the protocol client is prepared to accept 100 rows at this point, in ascending order.
    1. The header of the message is populated as follows:
        - **_msg** is set to 0x000000CC, indicating that this is a **CPMGetRowsIn** message.
        - **_status** is set to 0x00000000.
        - **_ulChecksum** contains the checksum, computed as specified in section 3.2.4.
        - **_ulReserved2** is set to 0x00000000.
    2. The body of the message is populated as follows:
        - **_hCursor** is set to 0xAAAAAAAA.
        - **_cRowsToTransfer** is set to 0x00000064.
        - **_cRowWidth** is set to 0x00000030 (from bindings).
        - **_cbSeek** is set to 0x0000000C.
        - **_cbReadBuffer** is set to 0x4000 (the maximum value for this field).
        - **_ulClientBase** is set to 0x00000000.
        - **Reserved1** is set to 0x00000000.
        - **Reserved2** is set to 0x00000001.
        - **Reserved3** is set to 0x00000000.
        - **Reserved4** is set to 0x00000000.
10. The protocol server processes it and responds with a CPMGetRowsOut message, assuming the protocol server found 100 items that contain the word "Microsoft".
    1. The header of the message is populated as follows:
        - **_msg** is set to 0x000000CC, indicating that this is a **CPMGetRowsOut** message.
        - **_status** is set to SUCCESS.
        - **_ulChecksum** is set to 0x00000000.
        - **_ulReserved2** is set to 0x00000000.
    2. The body of the message is populated as follows:
        - **_CRowsReturned** is set to 0x00000012. (18 results returned).
        - **Rows** contains the 18 items that contain the word "Microsoft". Because this is fixed-size data, it is simply structured as a list of 18, 48-byte CRowVariants that contain document identifiers.

11. The protocol client sends a CPMDisconnect message to end the connection.

    1. The header of the message is populated as follows:

        - **_msg** is set to 0x000000C9, indicating that this is a **CPMDisconnect** message.
        - **_status** is set to 0x00000000.
        - **_ulChecksum** is set to 0x00000000.
        - **_ulReserved2** is set to 0x00000000.

12. The protocol server processes the message and removes all client states for the protocol client.

5 Security

5.1 Security Considerations for Implementers

Crawling implementations that crawl secure content use the user context provided by SMB (as specified in [MS-SMB]) to enforce permissions on the named pipe used as the transport for this protocol.

5.2 Index of Security Parameters

Security Parameter	Section
Impersonation level	2.1

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 updates to those products.

- Microsoft Office SharePoint Server 2007
- Windows SharePoint Services 3.0

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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.

<1> Section 1.4: Applications interact with an OLE DB interface wrapper such as a protocol client, and not directly with the protocol. For more information, see [MSDN-OLEDBP-OI].

<2> Section 1.8: See [MSDN-PROPSET] for a list of supported property sets

<3> Section 2.1: Windows SharePoint Services 3.0 implementation uses \pipe\SPSearch name.

<4> Section 2.2.1.19: In Windows SharePoint Services 3.0 implementation alternative spellings are not generated and the **SpellingSuggestion** field contains an empty string.

<5> Section 2.2.1.19: In Microsoft Office SharePoint Server 2007 for Search, Windows SharePoint Services 3.0 and Office SharePoint Server 2007 implementations, the **vVectorData** field is set to the values of internal identifiers for query terms.

<6> Section 2.2.2: In Office SharePoint Server 2007 for Search, Office SharePoint Server 2007, and Windows SharePoint Services 3.0 implementations, the protocol client sets the **_status** field to 0x00000000.

<7> Section 2.2.3.1: In Office SharePoint Server 2007 for Search, Office SharePoint Server 2007, and Windows SharePoint Services 3.0 implementation, the **_iClientVersion** is set as specified in the following table.

Version	Value
32-bit, no SP1	0x00000102
64-bit, no SP1	0x00010102
32-bit, SP1 or higher	0x00000103
64-bit, SP1 or higher	0x00010103

<8> Section 2.2.3.2: The Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server 2007 SP1 do not initialize the value of the _reserved field and send arbitrary data

<9> Section 2.2.3.8: This field is set to 0x00004000 for all versions of Windows

<10> Section 2.2.4: The same pipe connection is used for the following messages except when the error is returned in a CPMConnectOut message. In the latter case, the pipe connection is terminated

by the client by closing the named pipe handle. Whenever the client end of pipe is closed the server releases all resources associated with the connection including the named pipe instance.

<11> Section 3.2.4.1.4: For a 32-bit protocol client talking to a 32-bit protocol server or a 64-bit protocol client talking to a 64-bit protocol server, this value is set to a memory address of the receiving buffer in the application process. This allows for pointers received in the Rows field of CPMGetRowsOut to be correct memory pointers in a client application process. Otherwise, it is set to 0x00000000.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
1.4 Relationship to Other Protocols	Updated product behavior note for dependencies on the MSSearch Protocol.	Minor
2.1 Transport	Updated product behavior note for using \pipe\SPSearch name.	Minor
2.2.1.10 CColumnSet	Updated description for CColumnSet structure.	Minor
2.2.1.19 QUERYMETADATA	Updated product behavior note of SpellingSuggestion field.	Minor
2.2.2 Message Headers	Updated product behavior note of the _status field 0x00000000.	Minor

8 Index

N

O

P

Q

R

S

T

V