[MS-SQP2]:
MSSearch Query
Version 2 Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
07/13/2009	0.1	Major	Initial Availability
08/28/2009	0.2	Editorial	Revised and edited the technical content
11/06/2009	0.3	Editorial	Revised and edited the technical content
02/19/2010	1.0	Major	Updated and revised the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.05	Minor	Clarified the meaning of the technical content.
09/27/2010	1.05	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.06	Editorial	Changed language and formatting in the technical content.
12/17/2010	1.06	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	1.06	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	1.06	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	1.06	No change	No changes to the meaning, language, or formatting of the technical content.
04/11/2012	1.06	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	1.06	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the MSSearch Query Version 2 Protocol that enables a protocol client to communicate with a server to issue search queries.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

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

Coordinated Universal Time (UTC)
GUID
handle
HRESULT
language code identifier (LCID)
little-endian
named pipe
Unicode

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

basic scope index key
binary large object (BLOB)
column
command tree
document identifier
full-text index catalog
index key
index key string
inflectional form
item
natural language query
noise word
property identifier
query expansion
query result
query server
rank
ranking
ranking model
restriction
row
scope index key
search query
sort order
stemming
token
user profile

The following terms are specific to this document:

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

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the technical documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

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

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

[MS-CIFO] Microsoft Corporation, "Content Index Format Structure Specification".

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

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

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2 Protocol Specification".

[MS-SQLPADM2] Microsoft Corporation, "SQL Administration Version 2 Protocol Specification".

[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

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

1.2.2 Informative References

[MSDN-FULLPROPSPEC] Microsoft Corporation, "FULLPROPSPEC", 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

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

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

[MS-SQP] Microsoft Corporation, "MSSearch Query Protocol Specification".

1.3 Protocol Overview (Synopsis)

The MSSearch Query Protocol facilitates for a protocol client that issues search queries to communicate with a protocol server hosting a search service. Specifically the activity carried out via this protocol is to provide the search results that match the client's search requests from the search catalog. The search service application running on a **query server** helps by efficiently organizing the extracted features of a collection of **items**.

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(1)** 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. The values stored in these tables eventually manifest themselves as search results to the end users.

1.3.1 Remote Querying

The MSSearch Query Protocol 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, including the search catalog name and configuration information.
   - The **restriction (1)** to specify what items are to be included and what items are to be 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(1)** that are to be returned for the search query.
   - The maximum time for query execution.
3. After the protocol server has acknowledged the protocol client's request to initiate the search query, the protocol client can request status information about the search query.
4. The protocol client requests a result set from the protocol server, and the protocol server responds by sending the property values for the items that were included in 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.
5. After the protocol server has released the search query, the protocol client sends a request to disconnect from the protocol server. The protocol client might 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 SMB2 protocol, as described in [MS-SMB2], for message transport. No other protocol depends directly on the MSSearch Protocol.<1>

This protocol is similar to, and shares a number of structures with, the previous version of the protocol described in [MS-SQP]. The two versions are incompatible and do not rely on each other.

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-SMB2].

1.6 Applicability Statement

The MSSearch Query protocol is designed for querying search catalogs on a remote server from a client. Typical size of the rowset is expected in the range of zero to 5000, with up to 4 columns.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

This protocol uses **HRESULT** values that are vendor extensible. Vendors are free to choose their own values for this field as long as the C bit (0x20000000) is set as specified in [MS-ERREF] section 2, indicating that the value is a customer code.

This protocol also uses NTSTATUS values taken from the NTSTATUS number space specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meaning<2> Choosing any other value runs the risk of a collision in the future.

Property Specification

Properties are represented by **property specifications,** which consist of a **GUID** representing a collection of properties called a property set, plus a 32-bit **property identifier** to identify the property within the set. The property identifier value MUST NOT be "0x00000000", "0xFFFFFFFF", or "0xFFFFFFFE". Vendors can guarantee that their properties are uniquely defined by placing them in a property set defined by their own GUIDs.<3>

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-SMB2]; the assignments are shown in the following table.

Parameter	Value	Reference
Pipe name	\pipe\OSearch14	[MS-SMB2]
Pipe name	\pipe\SPSearch4	[MS-SMB2]

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-SMB2]. The following pipe names are used:

- \pipe\OSearch14
- \pipe\SPSearch4

This protocol uses the underlying SMB2 named pipe protocol to retrieve the identity of the caller that made the connection, as specified in [MS-SMB2]. The protocol client MUST set SECURITY_IDENTIFICATION as the **ImpersonationLevel** in the request, specified in [MS-SMB2], to open the named pipe.

2.2 Message Syntax

2.2.1 Structures

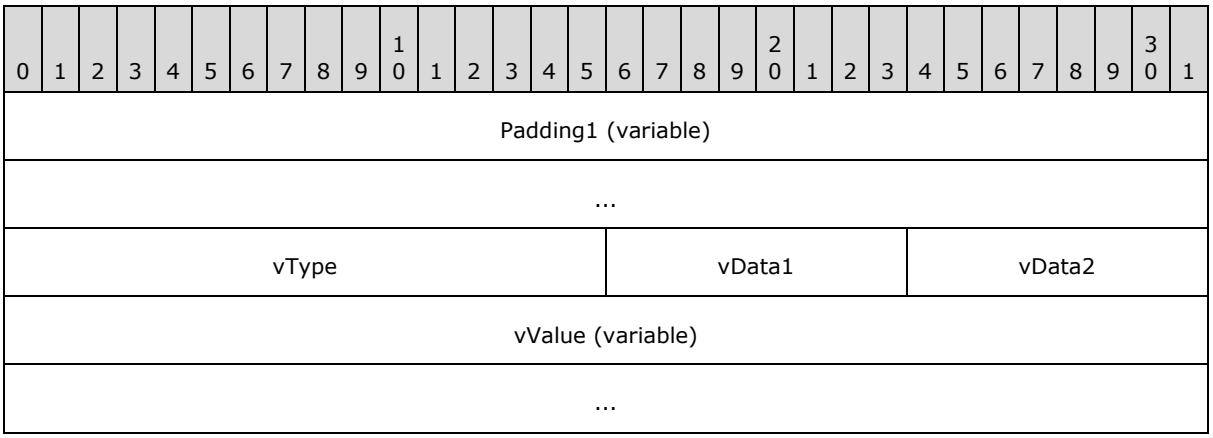
This section details data structures that are defined and used by the MSSearch Query Protocol. The following table summarizes the data structures defined in this section.

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 for full-text 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 (1) for a search query.
CPropertyRestriction	Contains a string for exact match for a property.
CPropertyRangeRestriction	Contains two property values to compare for a property.
CSort	Identifies a column to sort.
CRestriction	Stores a restriction (1) structure with its type and other generic information.
CColumnSet	Describes the columns to return.
CDbColId	Contains a column identifier.

Structure	Description
CDbProp	Contains a rowset property.
CDbPropSet	Contains a set of rowset properties.
CPidMapper	Contains an array of property specifications and serves to map from a property offset (sequential number) 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(1)** for a search query.
CTableColumn	Contains a column for the CPMSetBindingsIn message.
QUERYMETADATA	Contains information about a search query.
CKey	Contains information about a single **index key**.
CSynKey	Contains a confidence value to apply to a single index key.
CDocSetRestriction	Contains a list of **document identifiers** to restrict the result set to.
CInternalPropertyRestriction	Contains a property identifier to match with an operation.
COccRestriction	A restriction (1) that carries a position and another restriction (1) that defined how this position is interpreted.
CExactPositionWordRestriction	A restriction (1) that matches an index key in a specified position.
CSynRestriction	A restriction (1) that matches several keys or sub-restrictions.
CRangeRestriction	A restriction (1) that matches a range of keys.
CScopeRestriction	A restriction (1) that matches a **scope index key**.
CPhraseRestriction	A restriction (1) that matches a sequence of keys.
CNotRestriction	A restriction (1) that inverts the matching of all restrictions (1) it contains.
CWordRestriction	A restriction (1) that matches a single index key.
CProbRestriction	A restriction (1) that performs ranking on the items that match its sub-restrictions.
CScopeRangeRestriction	A restriction (1) that matches a range of scope index keys.
CRestrictionChildren	Contains a group of CRestriction structures.

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 zero 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 the length of this field is 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 that this value contains either "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].

Value	Meaning
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, which is defined in the following table; the value chosen is a choice made by the higher layer that provides it to the MSSearch Query Protocol, but the MSSearch Query Protocol treats VT_UINT and VT_UI4 as identical, with the preceding exception.
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 0x0007	A 64-bit floating point number, as specified in [IEEE754], representing the number of days after 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 after 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 table 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. For more information, 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.

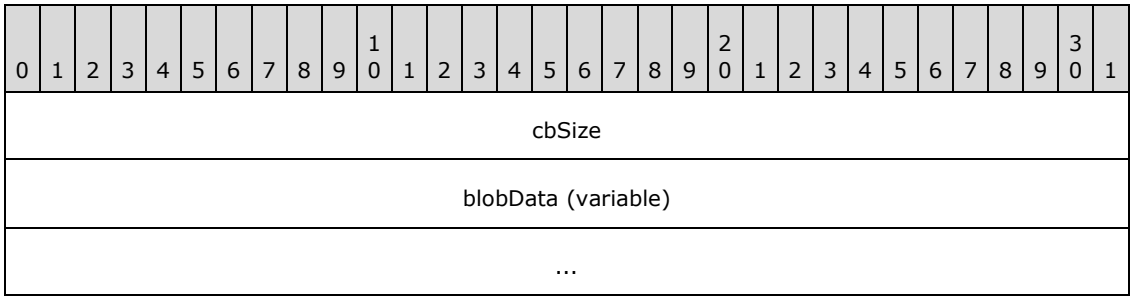
vData1 (1 byte): The value of this field MUST be set to "0x00".

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.

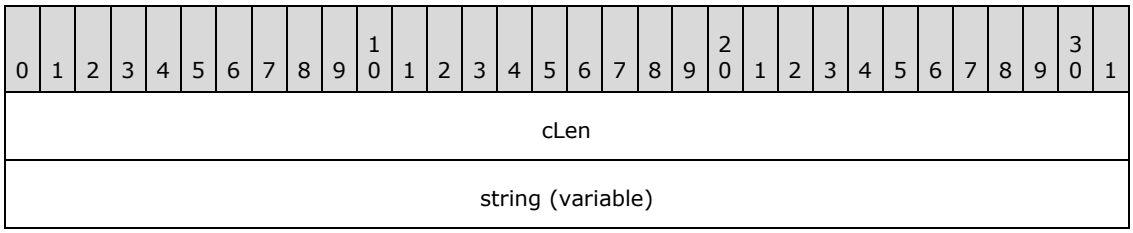


cbSize (4 bytes): A 32-bit unsigned integer. Indicates the size of the **blobData** field in bytes. If **vType** is set to VT_BSTR, **cbSize** MUST be set to "0x00000000" when the string represented is an empty string.

blobData (variable): MUST be of length **cbSize** in bytes. For a **vType** set to VT_BLOB, this field is opaque binary BLOB data. For a **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.

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



cLen: 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. 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.

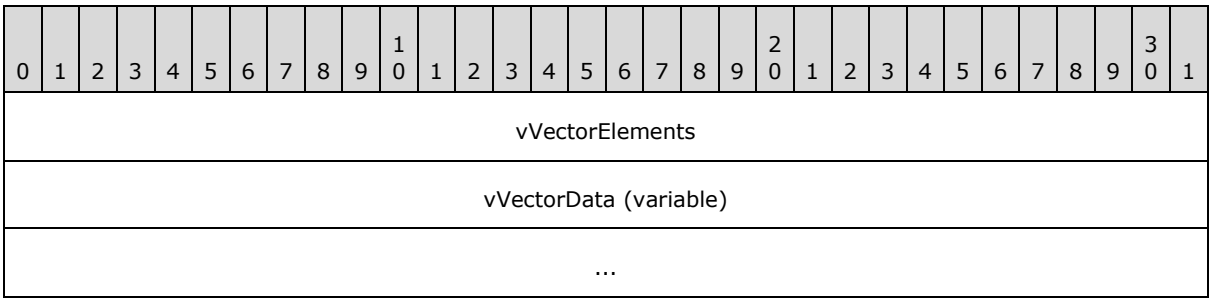
string: Null-terminated string. This field MUST be absent if **cLen** is set to "0x00000000".

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.

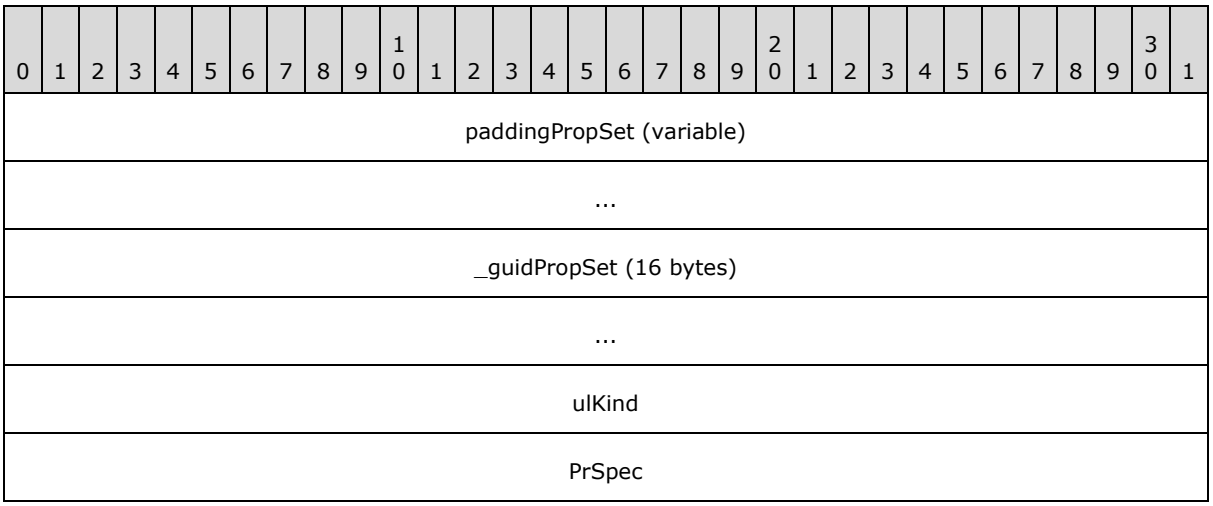
The elements in the **vVectorData** field MUST be separated by zero to 3 padding bytes such that each element begins at an offset that is a multiple of 4 bytes from the beginning of the

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. For properties to match, the **CFullPropSpec** structure MUST match the column identifier in the **full-text index catalog**.

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



paddingPropSet (variable): This field MUST be zero to 7 bytes in length. If the structure is a part of **QUERYMETADATA** structure (section 2.2.1.19), 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 **QUERYMETADATA** structure. Otherwise, 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 the length of this field is nonzero, the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

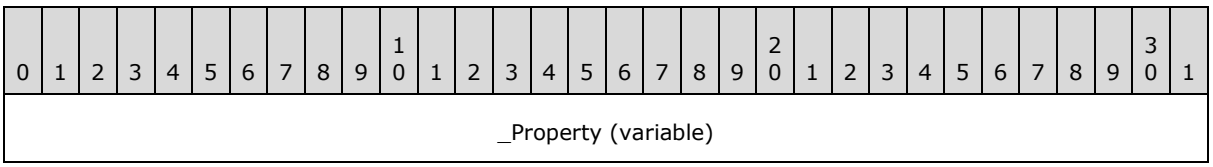
_guidPropSet (16 bytes): The GUID of the property set to which the property belongs.

ulKind (4 bytes): A 32-bit unsigned integer that 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.



...
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 zero 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 the length of this field is 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 non-null-terminated Unicode string 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 zero 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 the length of this field is 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].

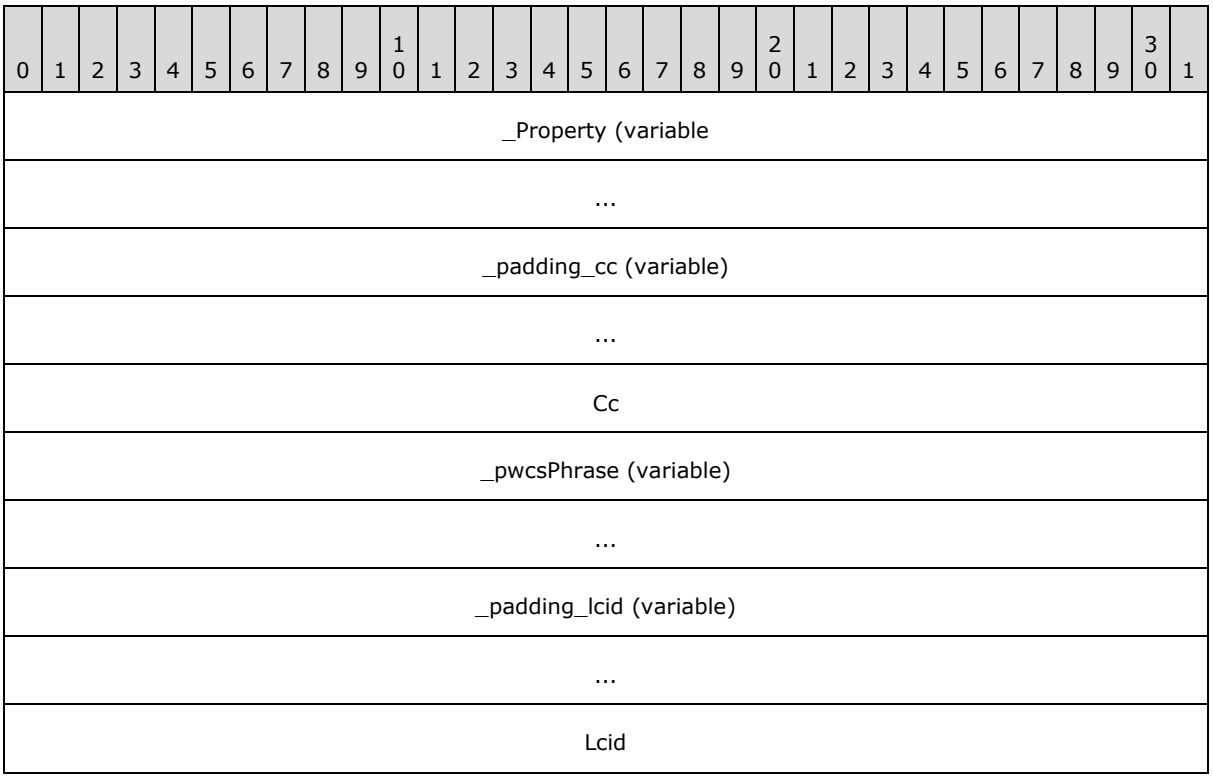
_ulGenerateMethod (4 bytes): A 32-bit unsigned integer, specifying the method to use when generating alternate word forms. The following table specifies the possible values for this field along with their meanings.

Value	Meaning
GENERATE_METHOD_EXACT 0x00000000	Exact match. Each word in the phrase MUST match exactly in the search catalog.

Value	Meaning
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, "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".
GENERATE_METHOD_THESAURUS 0x00000004	Matches synonyms of a word using a user-defined thesaurus. For example, if the words "glad" and "happy" were defined as synonyms in the thesaurus and the word "glad" was crawled, "happy" would match when performing a thesaurus match.

2.2.1.4 CNatLanguageRestriction

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



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

_padding_cc (variable): This field MUST be zero 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 the length of this field is nonzero, the value it contains is arbitrary. The contents 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.

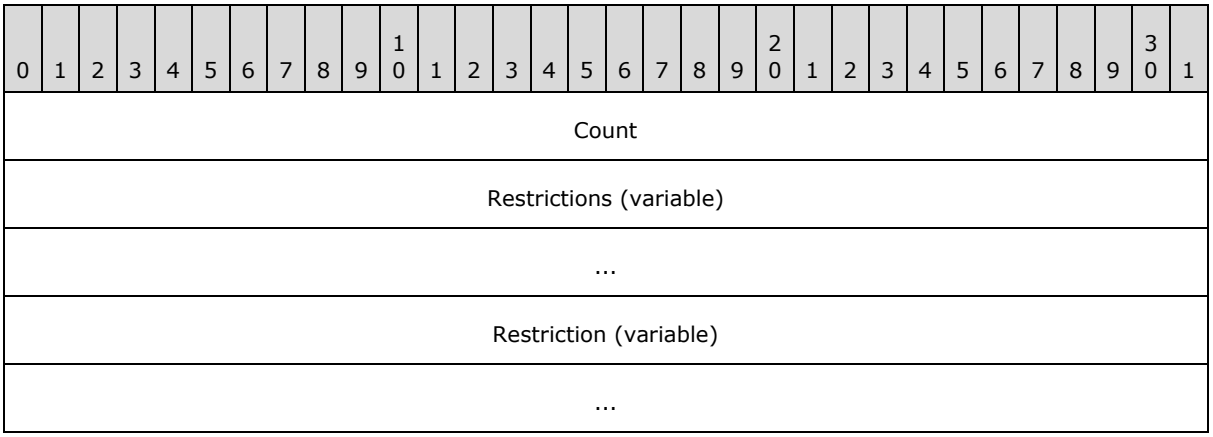
_pwcsPhrase (variable): A non-null-terminated Unicode string with the text to be searched for within the specific property. This string MUST NOT be empty. The **Cc** field contains the length of the string. The protocol server MUST interpret the string as specified in [MS-SEARCH] section 2.2.11.8, treating the whole string as text-expression.

_padding_lcid (variable): This field MUST be zero 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 the length of this field is 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 (1) nodes for constraining the results of a search query.



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

Restrictions (variable): An array of CRestriction structures. Structures in the array MUST be separated by zero 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.

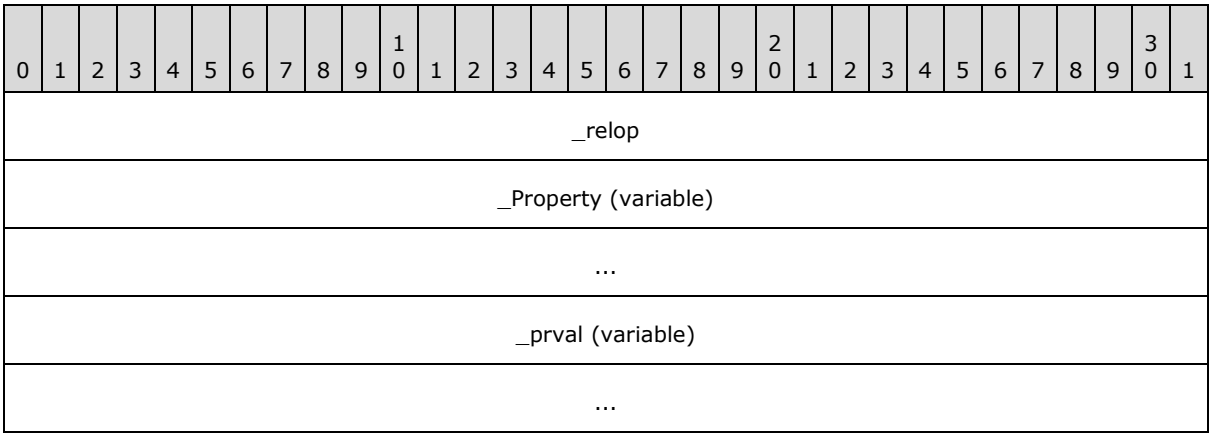
Restriction (variable): The format of this field depends on the value of the **Type** field of the enclosing CRestriction structure:

Value of the Type field	Meaning
RTAnd	The field MUST be empty.

Value of the Type field	Meaning
0x00000001	
RTOr 0x00000002	The field MUST be empty.
RTProximity 0x00000006	The field MUST be empty.
RTPhrase 0x00000013	The field contains a CPhraseRestriction. All the CRestriction structures in the **Restrictions** field MUST have a type of RTWord or RTSynonym.
RTProb 0x0000000D	The field contains a CProbRestriction. All the CRestriction structures in the **Restrictions** field MUST have a type of RTWord, RTSynonym or RTPhrase.

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:

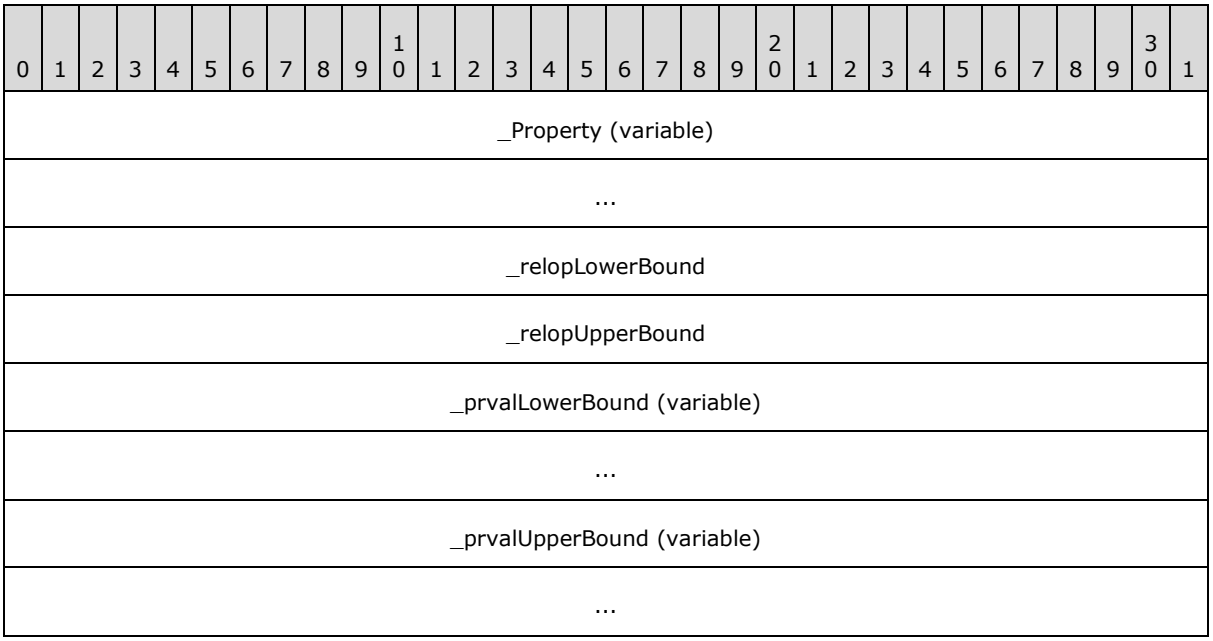
Value	Meaning
PRAny 0x00000200	The restriction (1) is true if any element in the property value has the relationship with some element in the **_prval** field.

_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 CPropertyRangeRestriction

The **CPropertyRangeRestriction** structure contains a property to get from each row, two comparison operators, and two constants. For each row, the value returned by the specific property in the row is compared against the constants to see if it belongs to the range specified by the constants. Comparison operators specify whether the constants themselves are included in the range. The property MUST have VT_FILETIME type.



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

_relopLowerBound (4 bytes): A 32-bit unsigned integer specifying whether **_prvalLowerBound** is considered a part of range. The lower 8 bits of **_relopLowerBound** MUST be one of the following values.

Value	Meaning
PRGT 0x02	The operator is "greater", **_prvalLowerBound** is excluded from the range.
PRGE	The operator is "greater or equal", **_prvalLowerBound** is included in the range.

Value	Meaning
0x03	

The higher 24 bits MUST have the value "0x000000", "0x000001", "0x000002", or "0x000004", and MUST be ignored.

_relopUpperBound (4 bytes): A 32-bit unsigned integer specifying whether **_prvalUpperBound** is considered a part of range. The lower 8 bits of **_relopUpperBound** MUST be one of the following values.

Value	Meaning
PRLT 0x00	The operator is "less", **_prvalUpperBound** is excluded from the range.
PRLE 0x01	The operator is "less or equal", **_prvalUpperBound** is included in the range.

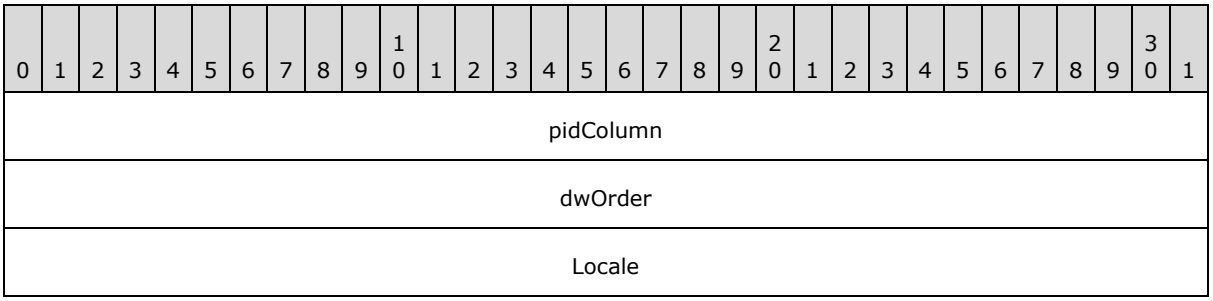
The higher 24 bits MUST have the value "0x000000", "0x000001", "0x000002", or "0x000004", and MUST be ignored.

_prvalLowerBound (variable): A CBaseStorageVariant structure containing the lower bound of the range. The **vType** field value MUST be set to "0x0040".

_prvalUpperBound (variable): A CBaseStorageVariant structure containing the upper bound of the range. The **vType** field value MUST be set to "0x0040".

2.2.1.8 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.

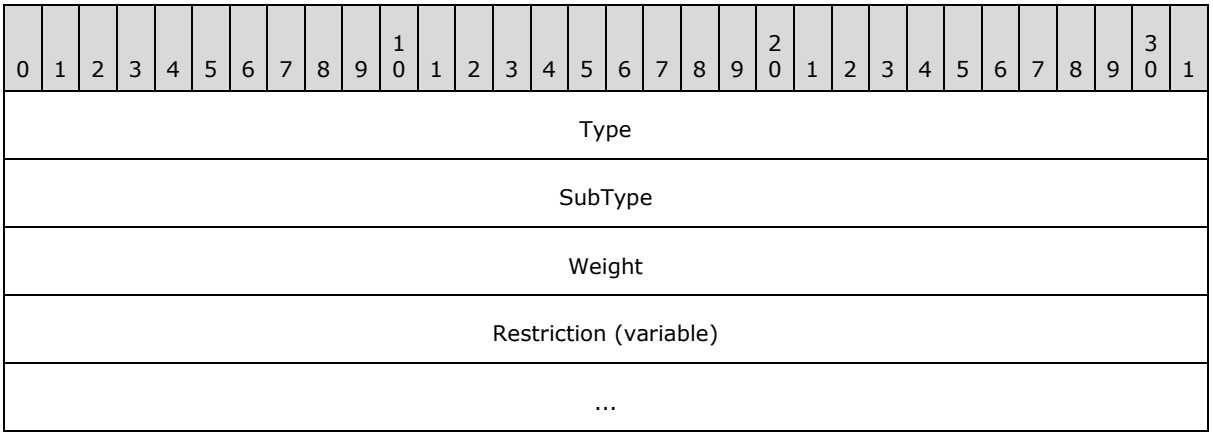
Value	Meaning
QUERY_SORTASCEND 0x00000000	The rows are sorted in ascending order based on the values in the column specified.
QUERY_SORTDESCEND	The rows are sorted in descending order based on the values in the

Value	Meaning
0x00000001	column specified.

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

2.2.1.9 CRestriction

The **CRestriction** structure stores a restriction (1) with its type and other generic information.



Type (4 bytes): A 32-bit unsigned integer indicating the restriction (1) type used for the command tree node. The type determines the format and interpretation of the **Restriction** field of the structure, as specified in the following table. This field MUST be set to one of the following values:

Value	Meaning
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.
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 (1) is undefined. The restriction (1) 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.
RTNatLanguage 0x00000008	The node contains a CNatLanguageRestriction.
RTPropertyRange	The node contains a CPropertyRangeRestriction.

Value	Meaning
0x0000001C	
RTDocSet 0x00000019	The node contains a CDocSetRestriction.
RTInternalProp 0x00000016	The node contains a CInternalPropertyRestriction.
RTExactWord 0x00000018	The node contains a COccRestriction.
RTSynonym 0x00000012	The node contains a COccRestriction.
RTRange 0x00000014	The node contains a CRangeRestriction.
RTScope 0x0000001A	The node contains a CScopeRestriction.
RTPhrase 0x00000013	The node contains a CNodeRestriction.
RTNot 0x00000003	The node contains a CNotRestriction.
RTWord 0x00000011	The node contains a COccRestriction.
RTProb 0x0000000D	The node contains a CNodeRestriction
RTScopeRange 0x0000001D	The node contains a CScopeRangeRestriction.

SubType (4 bytes): A 32-bit unsigned integer that specifies the inclusion behavior for child restrictions (1) of a CProbRestriction. Child restrictions (1) are restrictions (1) stored in the **Restrictions** array of a CNodeRestriction whose **Type** field value is RTProb. This value of the field MUST be set to zero ("0") and MUST be ignored for all other restrictions (1). The value MUST be one of the values listed in the following table.

Value	Meaning
rstUndefined 0x00000000	The inclusion behavior does not apply to this type of restriction (1).
rstInclude 0x10000000	Documents that match this restriction (1) MUST be returned unless they do not match one of the restrictions (1) with **SubType** rstMustInclude or match any of the restrictions (1) with **SubType** rstExclude among the child restrictions (1) of the restriction's parent CProbRestriction.
rstMustInclude 0x20000000	Documents that do not match this restriction (1) MUST NOT be returned.

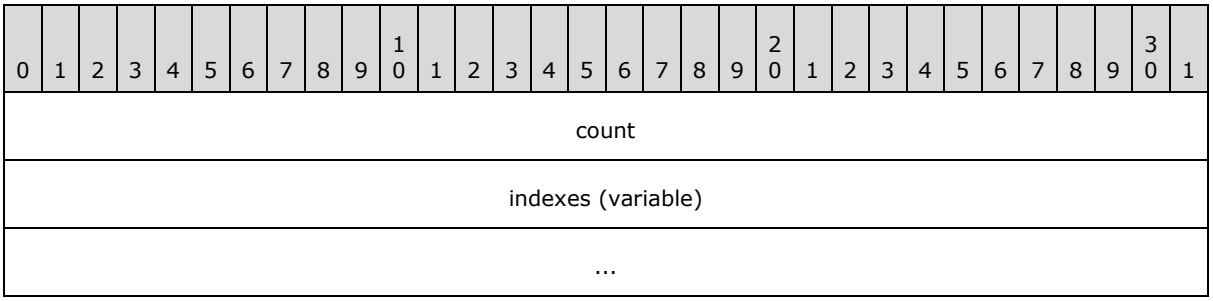
Value	Meaning
rstExclude 0x30000000	Documents that match this restriction (1) MUST NOT be returned.

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 (1) type for the command tree node. The format MUST be as indicated by the **Type** field.

2.2.1.10 CColumnSet

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

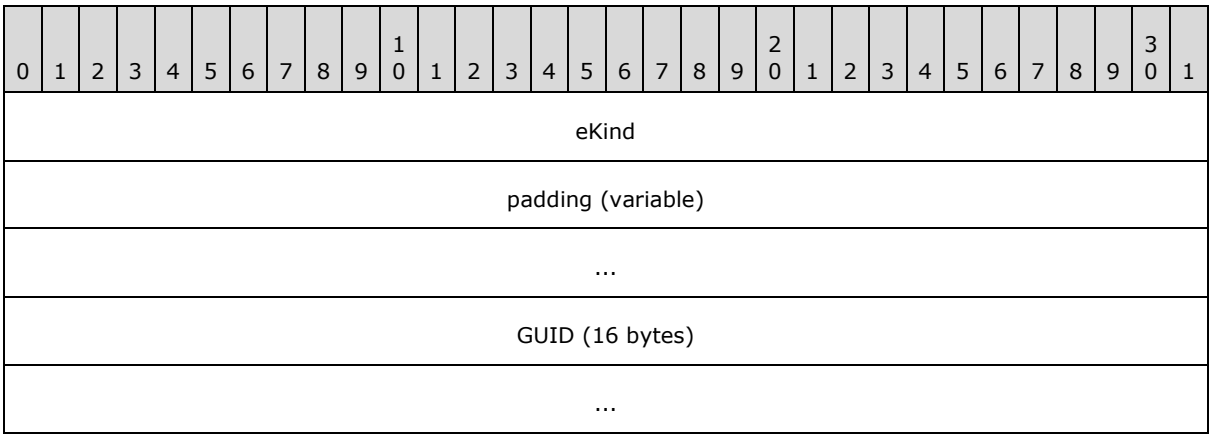


count (4 bytes): A 32-bit unsigned integer specifying the number of elements in the indexes array.

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.



ulId

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

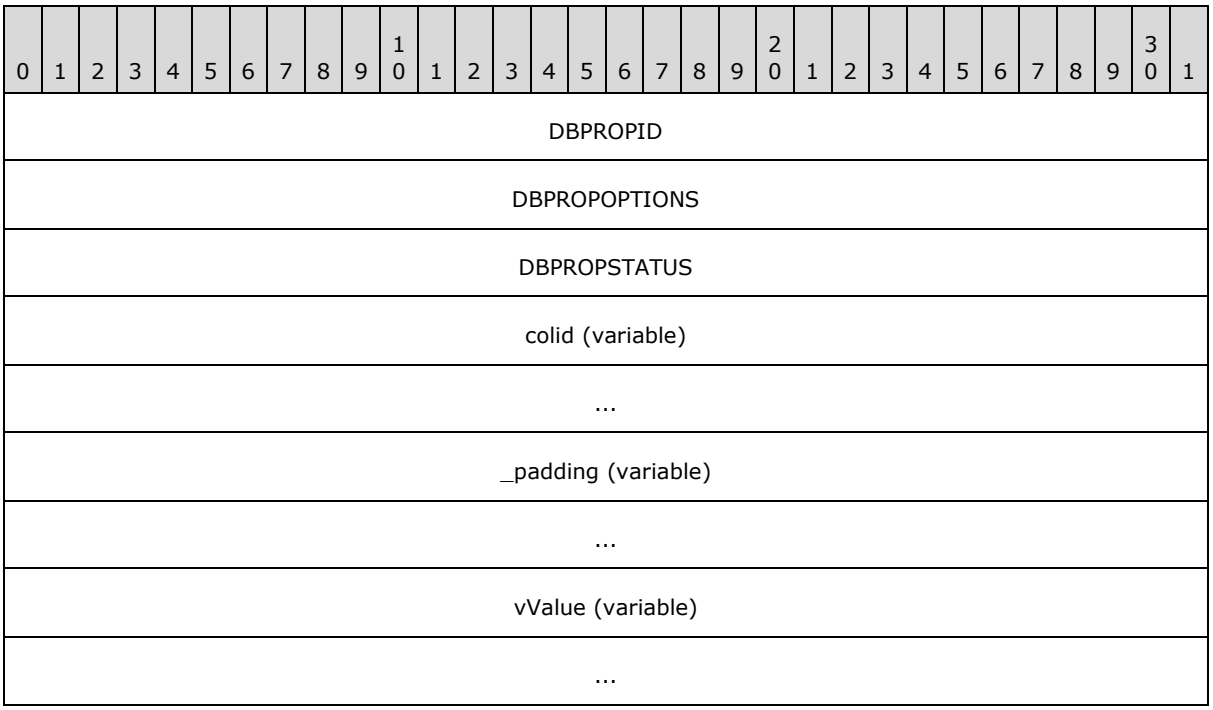
padding (variable): This field MUST be zero to 7 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 8 bytes from the beginning of the message that contains this structure. If the length of this field is nonzero, the value it contains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



DBPROPID (4 bytes): A 32-bit unsigned integer indicating the property identifier. The value MUST be a part of the **DBPROPSET_FSCIFRMWRK_EXT** property set, as specified in section 2.2.1.12.1.

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

DBPROPSTATUS (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 zero 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 the length of this field is nonzero, the value it contains is arbitrary. The contents of this field MUST be ignored by the receiver.

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

2.2.1.12.1 Database Properties

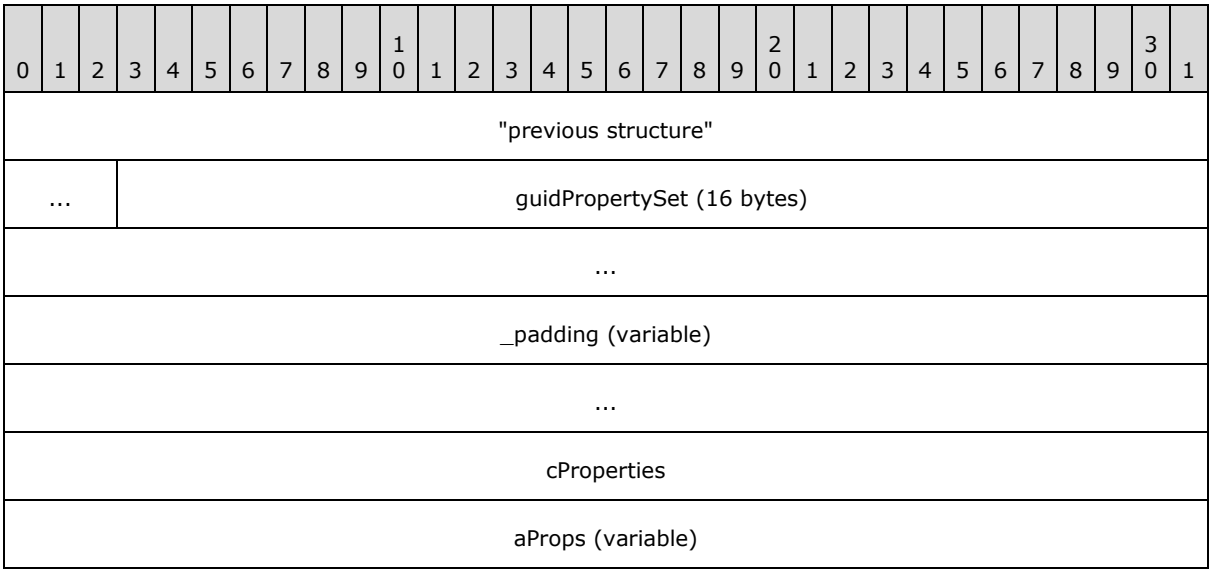
This protocol supports the following database properties to control the behavior of the query server. These properties are grouped into three property sets identified in the **guidPropertySet** field of the CDbPropSet structure.

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. The value MUST be a VT_LPWSTR or a VT_BSTR. The structure MUST be set so 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 so 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 starts 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** begins 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. 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 zero 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 the length of this field is nonzero, the value it contains is arbitrary. The contents 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 zero to 3 padding bytes so 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.

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
paddingCount (variable)																															
...																															
count																															
aPropSpec (variable)																															
...																															

paddingCount (variable): This field MUST be zero 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. 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																															
_padding (variable)																															
...																															
_guidRankingModelId (16 bytes)																															
...																															
_guidUserid (16 bytes)																															
...																															
_guidCorrelationId (16 bytes)																															
...																															

_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 (1 bit): MUST be set to "1" and MUST be ignored.

B through C - U1, U2 (1 bit each): MUST be set to "0" and MUST be ignored.

D - A1 (1 bit): Indicates whether the query is executed in an asynchronous way. Its value MUST be "0" if the query is executed in a synchronous way and "1" otherwise. If the value is "1", the

protocol client MUST receive the **CPMSendNotify** response from the protocol server before sending **CPMFetchValueIn** and **CPMGetRowsIn** messages, as specified in section 3.1.5.

E - A2 (1 bit): MUST be equal to **A1**.

F through O - U3, U4, U5, U6, U7, U8, U9, U10, U11, U12 (1 bit each): MUST be set to "0" and MUST be ignored.

P - IN (1 bit): 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 (1 bit each): MUST be set to "0" and MUST be ignored.

V - AT (1 bit): 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 (1 bit): 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 - EP (1 bit): Specifies the desired phonetics query expansion behavior. If the value is "1", the server MUST use phonetic query expansion, which means that phonetically similar tokens MUST be used for query expansion. If the value is "0", the server MUST NOT use phonetic query expansion. The logic used to identify phonetic similarity of tokens depends on server implementation.

Y - EN (1 bit): Specifies the desired nicknames query expansion behavior. If the value is "1", the server MUST use nicknames query expansion. If the value is "0", the server MUST NOT use nicknames query expansion.

Z - IT (1 bit): Indicates whether all server-side query execution time limits are ignored. The value MUST be "1" if the query is allowed to be executed indefinitely and "0" otherwise.

AA through AF - U18, U19, U20, U21, U22, U23 (1 bit each): 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", the server assumes all rows 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 will not time out.

_padding (variable): This field MUST be zero 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 the length of this field is nonzero, the value it contains is arbitrary. The contents of this field MUST be ignored by the receiver.

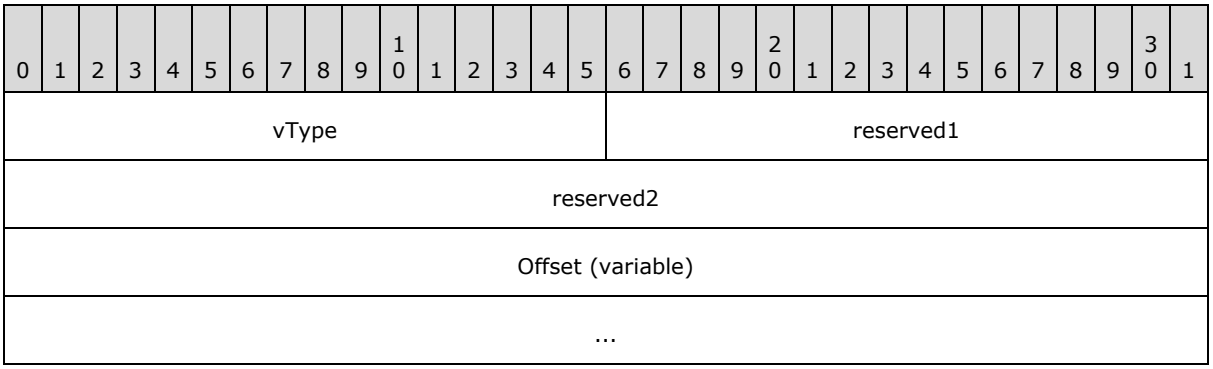
_guidRankingModelId (16 bytes): A 16-byte GUID value, specifying the **identifier** of the **ranking model** to be used for query execution. The GUID value "00000000-0000-0000-0000-000000000000" specifies the default ranking model. For details, see [MS-SQLPADM2] section 3.1.1.5.1.

_guidUserId (16 bytes): A 16-byte GUID value, specifying the **user profile** identifier of the user to be used for personalized ranking features.

_guidCorrelationId (16 bytes): A 16-byte GUID value that identifies the query for debugging purposes. There are no constraints on this value.

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.

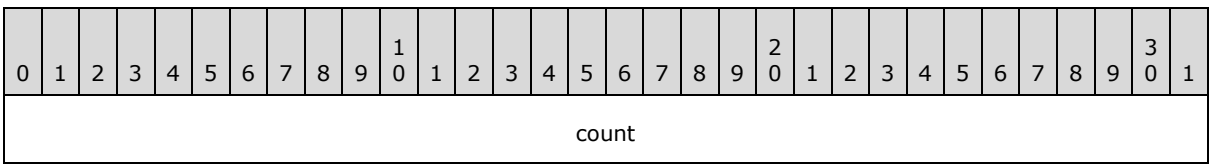
reserved1 (2 bytes): Not used. MUST be ignored on receipt.

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

Offset (variable): An offset to variable-length data such as a string. This MUST be a 32-bit value, or4-bytes long, if 32-bit offsets are being used, as specified in section 2.2.3.7, or a 64-bit value, or8-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.



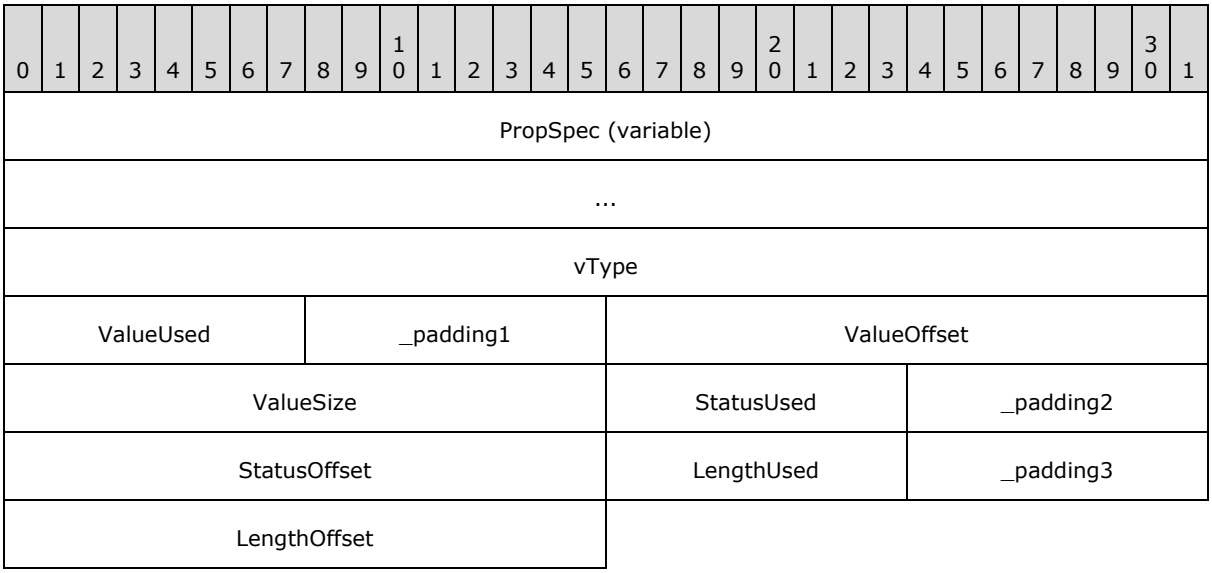
sortArray (variable)
...

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 zero 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. This field MUST be set to "0x0000000C".

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

_padding1 (1 byte): A 1-byte field that MUST be inserted before the **ValueOffset** field if, without that byte, **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. This field MUST be set to "0x01".

_padding2 (1 byte): A 1-byte field that MUST be inserted before the **StatusOffset** field if, without that byte, **StatusOffset** 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 at the offset specified in the **StatusOffset** request field. The status byte MUST be set to "0x00".

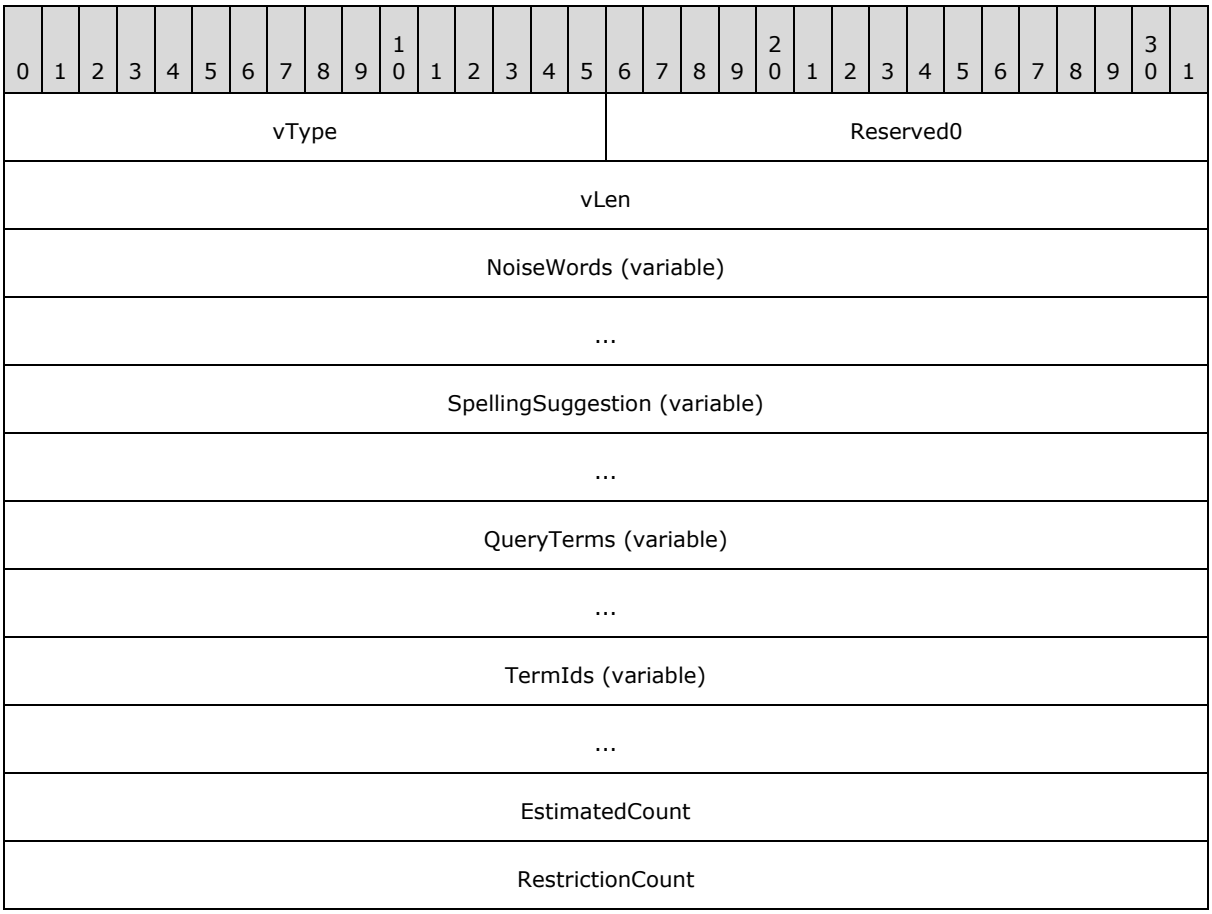
LengthUsed (1 byte): A reserved 1-byte field. MUST be set to "0x01".

_padding3 (1 byte): A 1-byte field that MUST be inserted before the **LengthOffset** field if, without that byte, **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 at 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.



Restrictions (variable)
...

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

Reserved0 (2 bytes): A reserved 16-bit field. This field MUST be set to "0x00".

vLen (4 bytes): A 32-bit field specifying the total length in bytes of the remaining fields of the structure (excluding the **vType**, **Reserved0**, and **vLen** fields).

NoiseWords (variable): A **CBaseStorageVariant** containing terms which were treated as ignored 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 zero or more query terms which were treated as ignored words by the query. For information about serialization for **vValue**, see section 2.2.1.1.

SpellingSuggestion (variable): A **CBaseStorageVariant** containing terms that have been determined by the server to be 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 spelling suggestions. If all spelling suggestions are the same as the original terms, the **vValue** MUST contain a null-terminated empty VT_LPWSTR. For information about serialization for **vValue**, see 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 zero or more query terms. For information about serialization for **vValue**, see 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.** For information about serialization for **vValue**, see section 2.2.1.1.

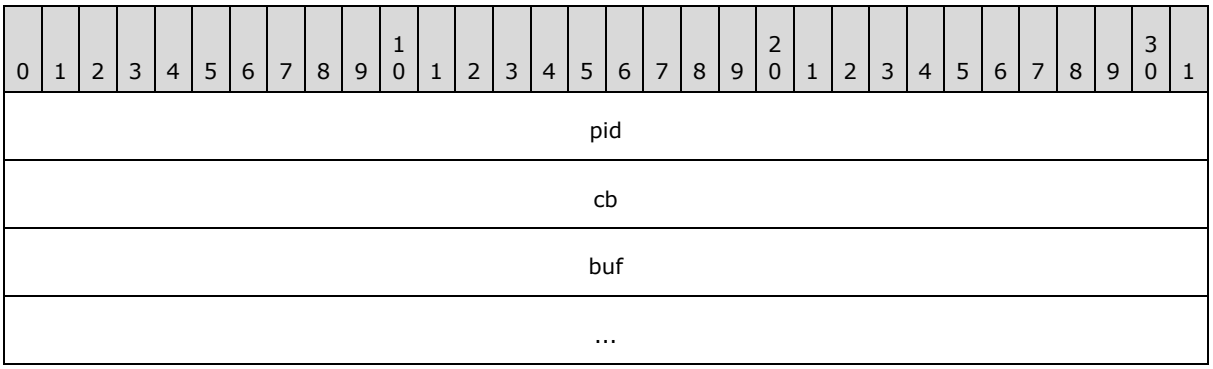
EstimatedCount (4 bytes): A 32-bit field containing the estimated number of total results, regardless of the number of rows requested by the protocol client.

RestrictionCount (4 bytes): A 32-bit field containing the number of restrictions (1) that follow in the **Restrictions** field. This value MUST be set to "1".

Restrictions (variable): A sequence of serializations of **CRestrictions**, as specified in section 2.2.1.9. There MUST be as many serializations of **CRestriction** as indicated in the **RestrictionCount** field. These MUST be the restrictions (1) that the protocol server used to generate the row set containing the results of the query. This set of restrictions (1) can be the same as the restrictions (1) specified in the **CPMCreateQueryIn** message, as specified in section 2.2.3.3, but it can also be different if the protocol server modified the restriction (1) before executing the query.

2.2.1.20 CKey

The **CKey** structure stores an index key used in restrictions (1).



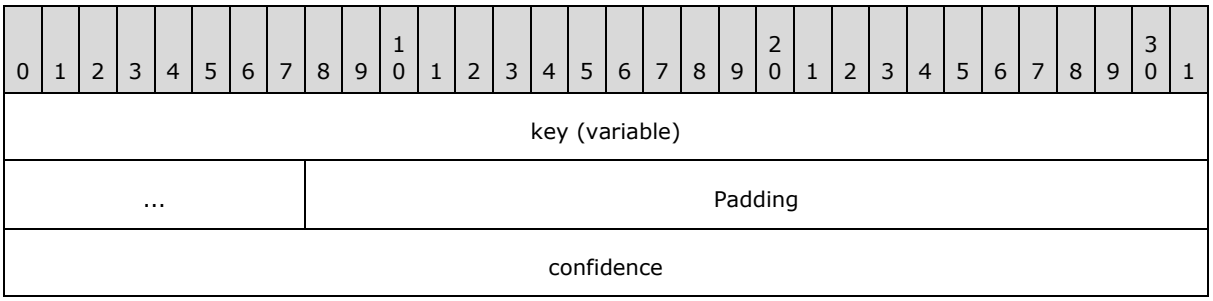
pid (4 bytes): The property identifier of a property referenced by the index key.

cb (4 bytes): The size of the **buf** field in bytes.

buf (variable): The **index key string** of the index key.

2.2.1.21 CSynKey

The **CSynKey** structure pairs a CKey structure with a confidence value.



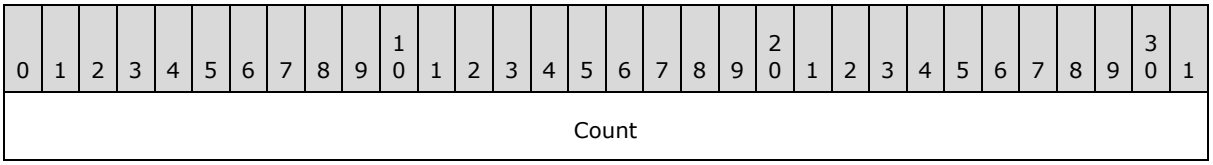
key (variable): The CKey structure.

Padding (variable): A field that MUST be zero 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 the length of this field is not zero, the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

confidence (4 bytes): The confidence applied to this index key. The confidence is a 32-bit floating point value. The value MAY be used during **ranking** of search results to put a higher weight on some keys, and a lower weight on others. It MAY also be ignored.

2.2.1.22 CDocSetRestriction

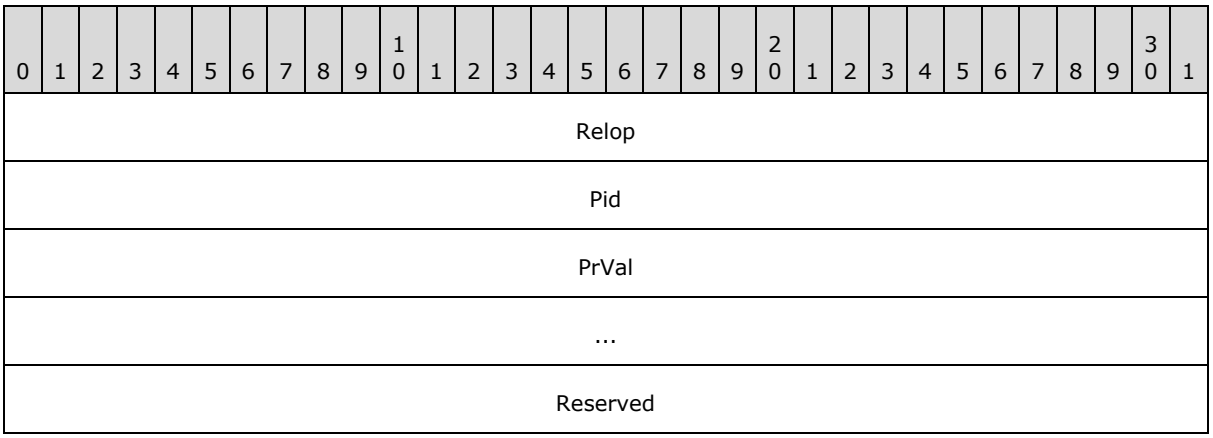
The **CDocSetRestriction** structure contains the number of document identifiers used for matching.



Count (4 bytes): A 32-bit unsigned integer indicating how many document identifiers are following.

2.2.1.23 CInternalPropertyRestriction

The **CInternalPropertyRestriction** 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 matches the relationship specified by the **Relop** field.



Relop (4 bytes): A 32-bit unsigned integer specifying the relation to perform on the property. **Relop** MUST be one of the following values.

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

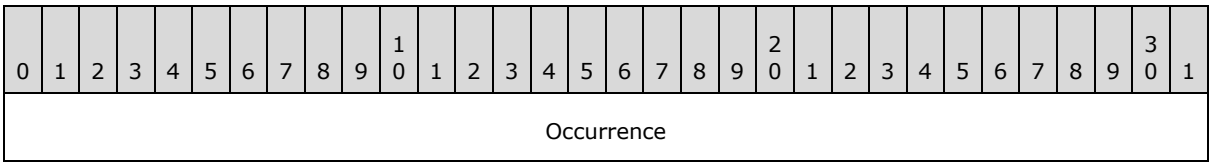
Pid (4 bytes): A 32-bit unsigned integer indicating the property identifier of the property.

PrVal (variable): A CBaseStorageVariant structure containing the value to relate to the property. The type of data stored in **PrVal** MUST match the data type of the property specified by the **Pid** field.

Reserved (4 bytes): A reserved 32-bit unsigned integer that MUST be set to "0".

2.2.1.24 COccRestriction

The **COccRestriction** structure contains a restriction (1) that carries occurrence information.



PrevNoiseWordCount
PostNoiseWordCount
OccRestriction (variable)
...

Occurrence (4 bytes): A 32-bit unsigned integer indicating the occurrence of the restriction (1).

PrevNoiseWordCount (4 bytes): A 32-bit unsigned **integer** indicating how many noise words occur before the position indicated in the **Occurrence** field.

PostNoiseWordCount (4 bytes): A 32-bit unsigned **integer** indicating how many noise words occur after the position indicated in the **Occurrence** field.

OccRestriction (variable): The format of this field is dependent on the value of the **Type** field of the enclosing CRestriction structure:

Value of the Type field	Meaning
RTExactWord 0x00000018	The field contains a CExactPositionWordRestriction.
RTSynonym 0x00000012	The field contains a CSynRestriction.
RTWord 0x00000011	The field contains a CWordRestriction.

Whether this restriction (1) matches or not is determined by the contents of the **OccRestriction** field.

2.2.1.25 CExactPositionWordRestriction

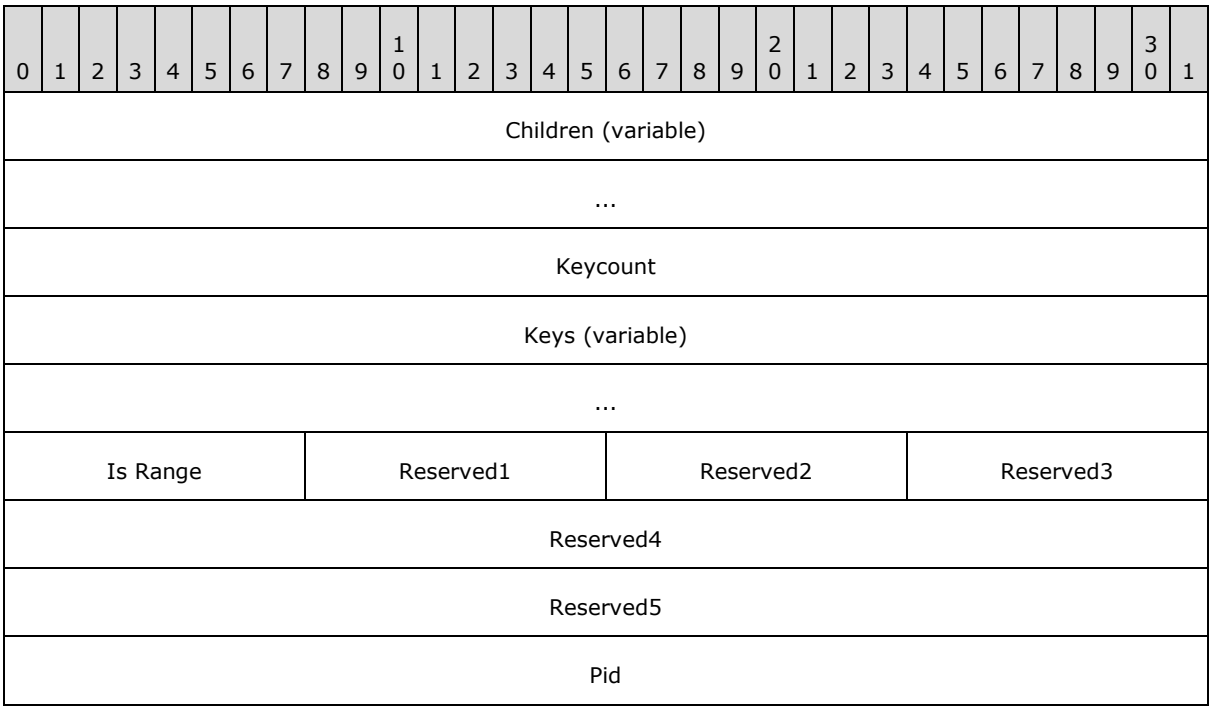
The **CExactPositionWordRestriction** imposes stricter rules for matching a CWordRestriction.

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
OccWindow																															

OccWindow (4 bytes): A 32-bit unsigned integer. The restriction (1) MUST only match if the enclosing CWordRestriction matches, and if the position of any of the matching index keys in the document modulo this value is equal to the value of the **Occurrence** field of the enclosing COccRestriction structure.

2.2.1.26 CSynRestriction

The **CSynRestriction** structure contains several **CSynKey** structures and other CRestriction structures to match.



Children (variable): A CRestrictionChildren structure. This restriction (1) MUST match if any of the restrictions (1) in this field match.

KeyCount (4 bytes): A 32-bit unsigned integer storing the number of **CSynKey** structures in the **Keys** field.

Keys (variable): An array of **CSynKey** structures. Structures in the array MUST be separated by zero to 3 padding bytes so 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. This restriction (1) MUST match an item if the item contains any of the index keys that are contained in this structure.

IsRange (1 byte): An 8-bit unsigned integer. If this value is "0", a document MUST match the restriction (1) if it contains an index key that is equal to any of the keys in the **Keys** field. Otherwise, a document MUST match the restriction (1) if it contains an index key of which any of the keys in the **Keys** field is a prefix.

Reserved1 (1 byte): An 8-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved2 (1 byte): An 8-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved3 (1 byte): An 8-bit unsigned integer. This value is reserved and MUST be ignored.

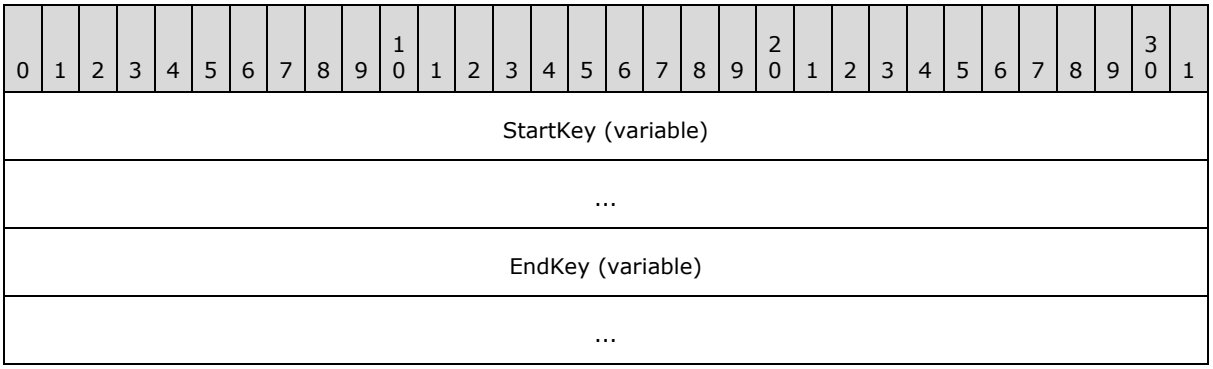
Reserved4 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to zero ("0").

Reserved5 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to "1".

Pid (4 bytes): A 32-bit unsigned integer indicating the property identifier of the property to be compared.

2.2.1.27 CRangeRestriction

The **CRangeRestriction** matches a range of index keys.



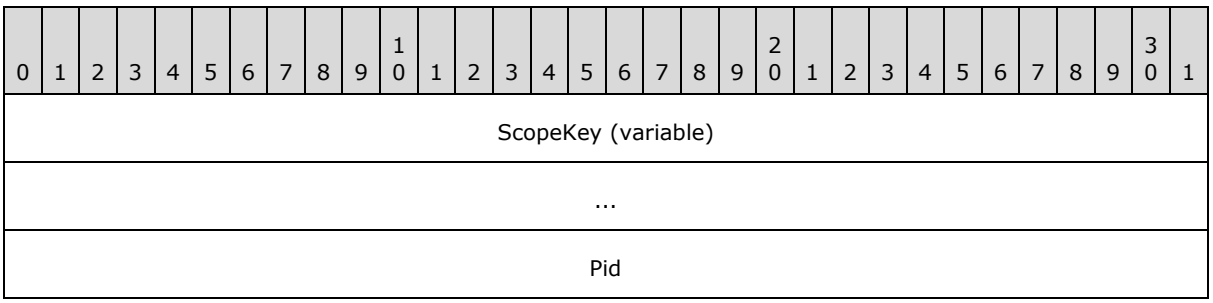
StartKey (variable): A CKey structure representing the first index key to match on.

EndKey (variable): A CKey structure representing the first index key to not match on.

The **CRangeRestriction** matches if a document contains an index key that is greater than or equal to the key in the **StartKey** field, and smaller than the key in the **EndKey** field.

2.2.1.28 CScopeRestriction

The **CScopeRestriction** contains a **basic scope index key** and a property identifier. For each row, the basic scope index key created from the value returned by the specific property in the row is compared against this value.



ScopeKey (variable): A CKey structure that contains the basic scope index key against which the properties are compared.

Pid (4 bytes): A 32-bit unsigned integer, specifying the property identifier of the property to be compared.

2.2.1.29 CPhraseRestriction

The **CPhraseRestriction** structure defines the behavior of the CNodeRestriction that contains it. The restrictions (1) in the enclosing CNodeRestriction have to match in the right order for the **CPhraseRestriction** to match.

Reserved1 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to "0".

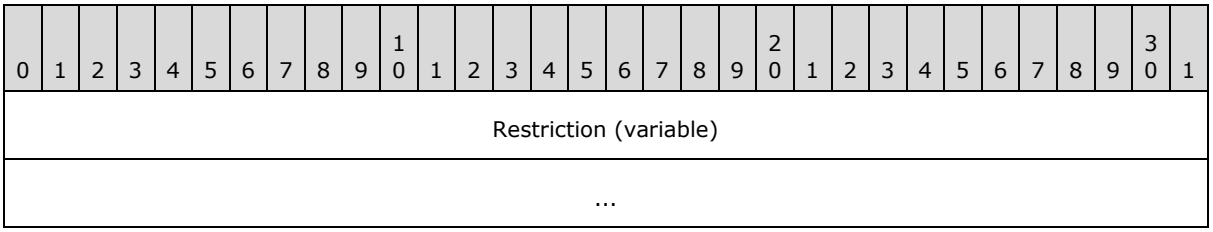
Reserved2 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to "1".

MaxInexactDistance (4 bytes): A 32-bit unsigned integer. This value determines the tolerance for inexact matching.

This restriction (1) MUST only match if all of the child restrictions (1) in the enclosing CNodeRestriction match, and if the difference between the **Occurrence** field of the child restriction (1) and the position of the corresponding index key is smaller than or equal to the value of the **MaxInexactDistance** field.

2.2.1.30 CNotRestriction

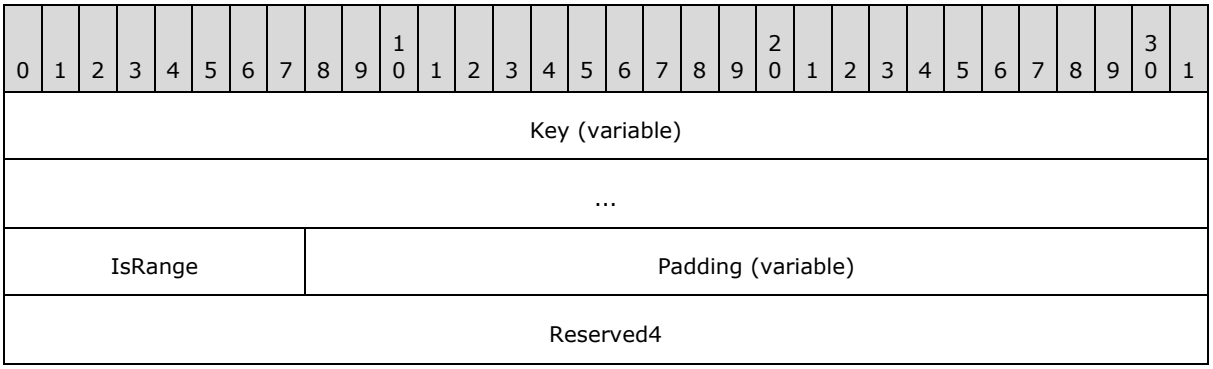
The **CNotRestriction** reverses the matching of the restriction (1) it contains.



Restriction (variable): A CRestriction structure. If this restriction (1) matches, the **CNotRestriction** MUST NOT match. If this restriction (1) does not match, the **CNotRestriction** MUST match.

2.2.1.31 CWordRestriction

The **CWordRestriction** structure contains a key to match.



Reserved5

Key (variable): A CKey structure, as specified in section 2.2.1.20, describing the index key to match.

IsRange (1 byte): An 8-bit unsigned integer. If this value is "0", a document MUST only match the restriction (1) if it contains an index key that is equal to the key in the **Key** field. Otherwise, a document MUST only match the restriction (1) if it contains an index key of which the key in the **Key** field is a prefix.

Padding (variable): This field MUST be zero 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 the length of this field is nonzero, the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

Reserved4 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to "0".

Reserved5 (4 bytes): A 32-bit unsigned integer. This value is reserved. It MUST be set to "1".

2.2.1.32 CProbRestriction

The **CProbRestriction** advises the server to perform probabilistic ranking on the restrictions (1) it contains.

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)																															
...																															
Reserved1																															
Reserved2																															
Reserved3																															
Reserved4																															
Reserved5																															
Reserved6																															

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

Padding1 (variable): This field MUST be zero 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 the length of this field is nonzero, the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

Reserved1 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved2 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved3 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

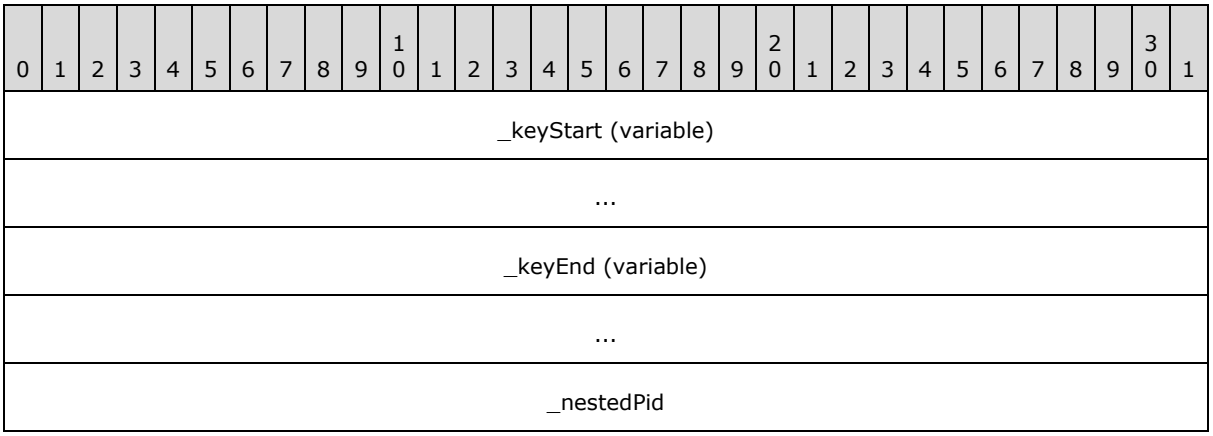
Reserved4 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved5 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

Reserved6 (4 bytes): A 32-bit unsigned integer. This value is reserved and MUST be ignored.

2.2.1.33 CScopeRangeRestriction

The **CScopeRangeRestriction** structure contains two basic scope index keys and a property identifier. The row matches the restriction (1) if the basic scope index key created from the value returned by the specific property is greater than or equal to the lower bound basic scope index key and less than or equal to the upper bound basic scope index key. Ordering of index keys is specified in [MS-CIFO] section 2.2.3.



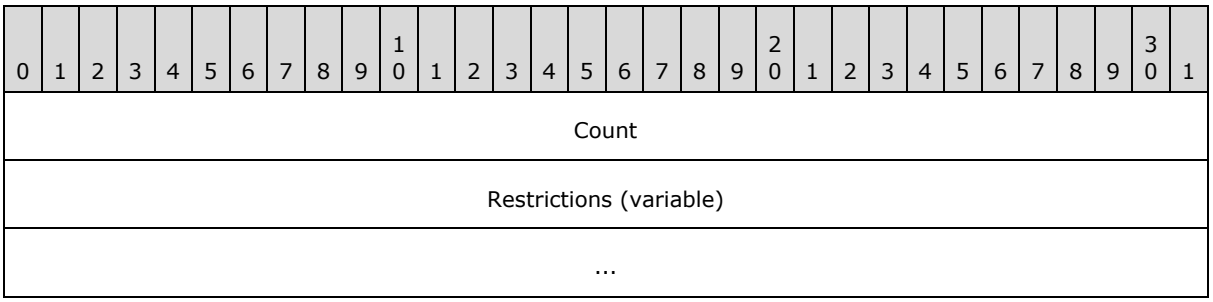
_keyStart (variable): A CKey structure that contains the lower bound basic scope index key.

_keyEnd (variable): A CKey structure that contains the upper bound basic scope index key.

_nestedPid (4 bytes): A 32-bit unsigned integer, specifying the property identifier of the property to be compared.

2.2.1.34 CRestrictionChildren

The **CRestrictionChildren** structure holds a group of CRestriction structures.



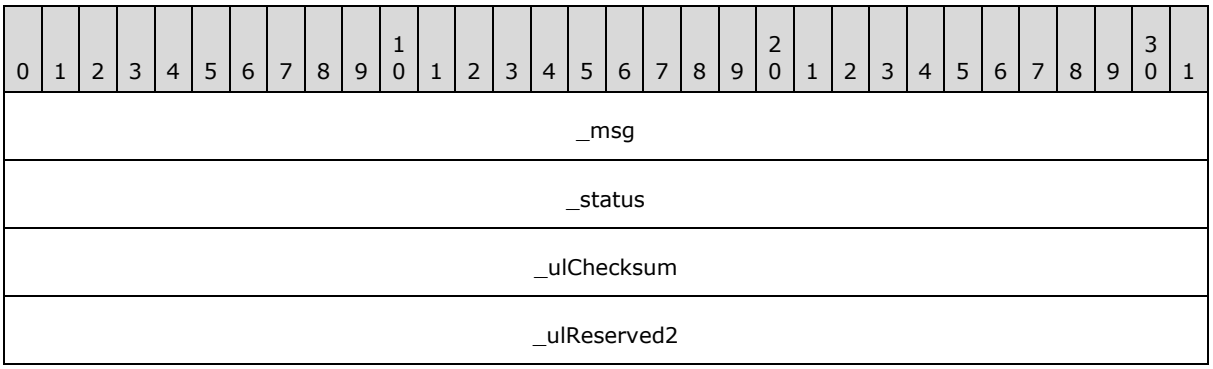
Count (4 bytes): A 32-bit unsigned integer storing the number of CRestriction structures in the **Restrictions** field.

Restrictions (variable): An array of CRestriction structures. Structures in the array MUST be separated by zero 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.

2.2.2 Message Headers

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

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



_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 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. The value of **_msg** MUST be set to one of the following values.

Value	Meaning
0x000000C8	CPMConnectIn or CPMConnectOut
0x000000C9	CPMDisconnect

Value	Meaning
0x000000CA	CPMCreateQueryIn or CPMCreateQueryOut
0x000000CB	CPMFreeCursorIn or CPMFreeCursorOut
0x000000CC	CPMGetRowsIn or CPMGetRowsOut
0x000000D0	CPMSetBindingsIn
0x000000D1	CPMGetNotifyIn or CPMGetNotifyOut
0x000000D2	CPMSendNotifyOut
0x000000E4	CPMFetchValueIn or CPMFetchValueOut

_status (4 bytes): An HRESULT or NTSTATUS value, indicating the status of the requested operation. When sent by the protocol client, this field MUST be set to "0" and the protocol server MUST ignore the value<5>.

_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

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 Otherwise the value MUST be set to "0x00000000". The _**ulClientBase** field of the CPMGetRowsIn message contains the lower 32 bits of the base value.

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																															
_cbBlob2																															
_padding (12 bytes)																															
...																															
MachineName (variable)																															
...																															
UserName (variable)																															
...																															
_paddingcPropSets (variable)																															
...																															
cPropSets																															
PropertySets																															
...																															
_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<6>. 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.
0x00010102	The protocol client is capable of handling 64-bit offsets in **CPMGetRowsOut** messages.

_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 (4 bytes): 4 bytes of padding that MUST be ignored.

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

_padding (12 bytes): 12 bytes of padding that 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 name of the account running the application that called 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 zero 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**.

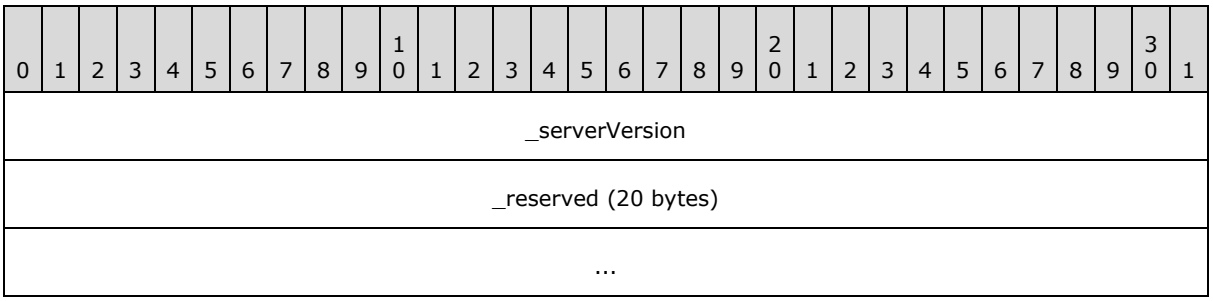
_paddingExtPropset (variable): This field MUST be zero to 7 bytes in length. The number of bytes MUST be the number required to make the byte offset of the **cExtPropSets** field from the beginning of the message that contains this structure equal to a multiple of 8. The value of the bytes can be any 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**. The **aProps** field of the first element of the array MUST contain a **CDbProp** structure with **DBPROPID** set to "0x00000002" (DBPROP_CI_CATALOG_NAME).

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.



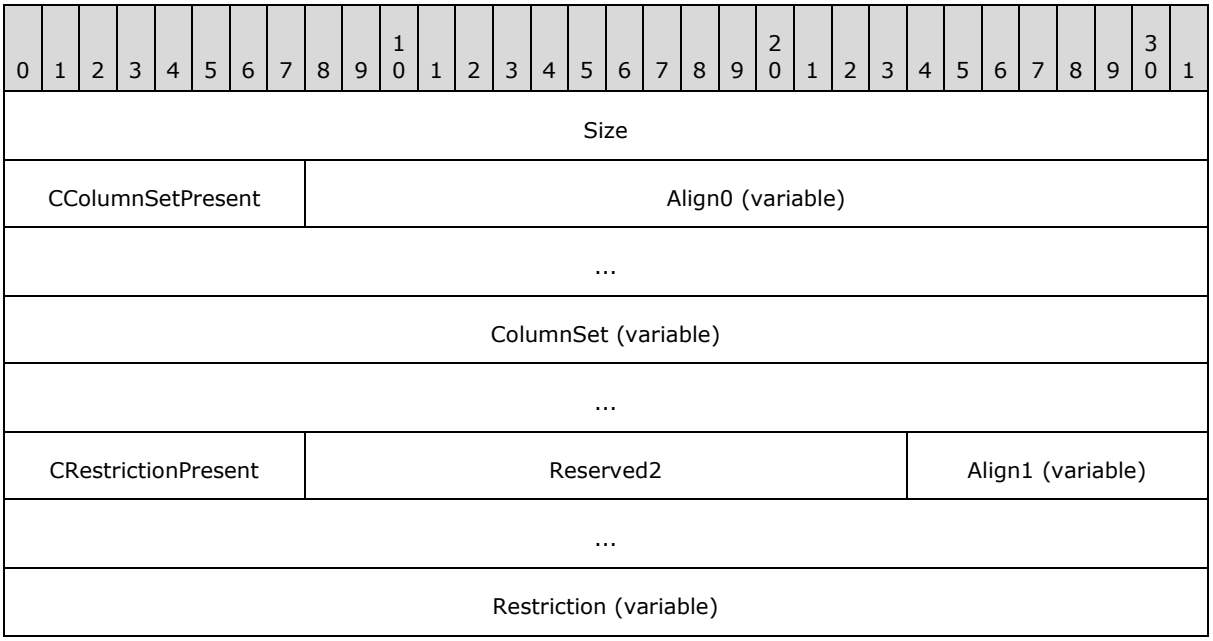
_serverVersion (4 bytes): A 32-bit integer, indicating whether the server can support 64-bit offsets, as specified in section 2.2.3.7.

Value	Meaning
0x00000102	The protocol server can only send 32-bit offsets.
0x00010102	The protocol server can send 32-bit or 64-bit offsets.

_reserved (20 bytes): A 20 byte reserved field. The protocol client MUST ignore these values.

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.



...	
CSortSetPresent	Align2 (variable)
...	
SortSet	
...	
Reserved0	Align3 (variable)
...	
RowSetProperties (variable)	
...	
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.

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 zero, 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.

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

If this field is set to any nonzero value, the **Restriction** field MUST be present. If this field is set to "0x00", the **Restriction** field MUST NOT be present.

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

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

Restriction (variable): A CRestriction structure containing the command tree of the search query. The tree MUST only contain CRestriction structures with **Type** equal to RTAnd, RTOr, RTNot, RTContent, RTProperty, RTProximity, RTNatLanguage, RTPropertyRange or RTPhrase.

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

If this field is set to any nonzero value, the **SortSet** field MUST be present. If set to "0x00", the **SortSet** field MUST NOT be present.

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

Align3 (variable): A field structure of zero, 1, 2 or 3 bytes that is used to align the next field to a 4-byte boundary. This field 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.

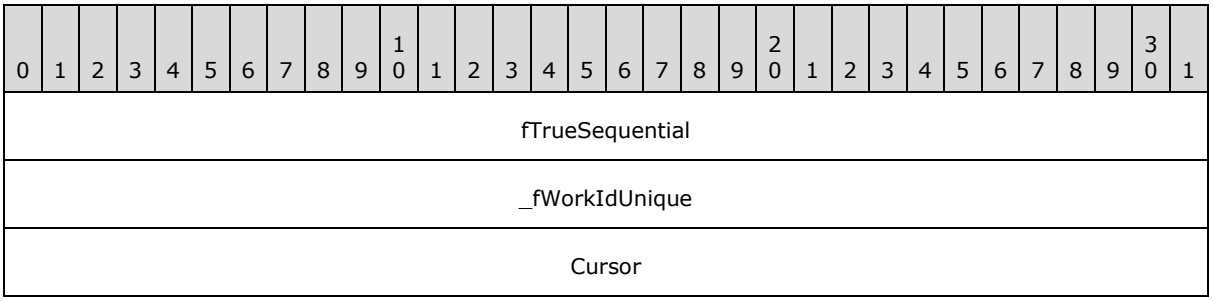
Reserved1 (4 bytes): A 32-bit reserved field. This field MUST be set to "0x00000000".

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 optimizations for the search query. This field MUST be set to one of the values in the following table.

Value	Meaning
0x00000000	For the search query provided in **CPMCreateQueryIn**, the protocol server is unable to use optimizations and there is larger latency in delivering query results.
0x00000001	For the search query provided in **CPMCreateQueryIn**, the protocol server is able to use optimizations and the query results can be expected to be delivered faster.

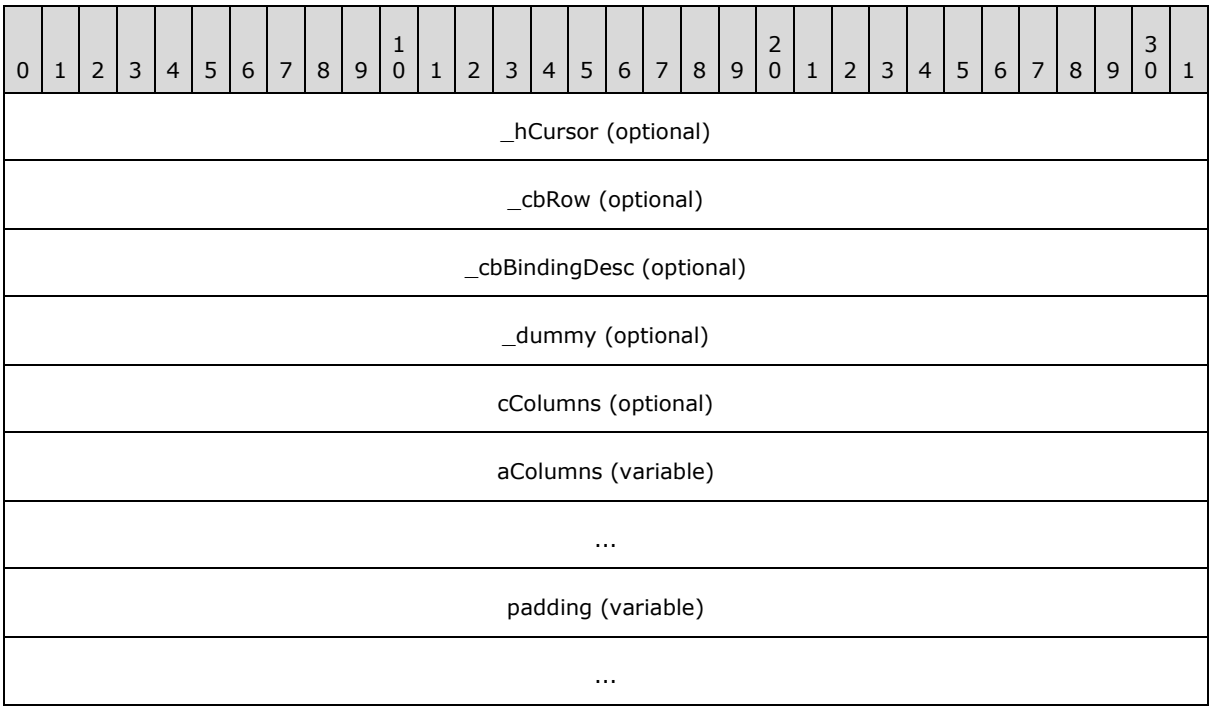
_fWorkIdUnique (4 bytes): A Boolean value indicating if the document identifiers pointed to by the cursors are unique throughout query results. This field MUST be set to one of the following values.

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 replies 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 NOT be present 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 NOT be present 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 NOT be present 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 NOT be present when the message is sent by the protocol server.

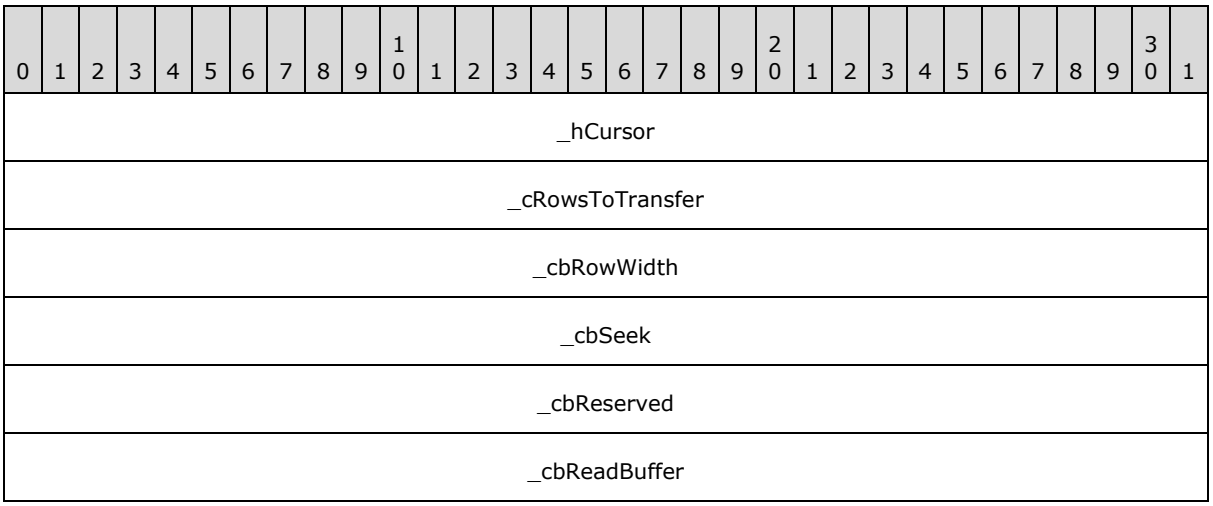
cColumns (4 bytes, optional): A 32-bit unsigned integer indicating the number of elements in the **aColumns** array. This field MUST be present when the message is sent by the protocol client, and MUST NOT be present when the message is sent by the protocol server.

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 NOT be present when the message is sent by the protocol server. Structures in the array MUST be separated by zero to 3 padding bytes such that each structure has a 4-byte alignment from the beginning of a message. Padding bytes can be any arbitrary value, and MUST be ignored on receipt.

padding (variable): This field MUST be zero to 3 bytes long to pad the message 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 NOT be present 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.



_ulClientBase
Reserved1
Reserved2
Reserved3
Reserved4

_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. This 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 so that the **Rows** field follows the **Reserved1** field.

_cbReadBuffer (4 bytes): A 32-bit unsigned integer that MUST be set to the maximum of the value of **_cbRowWidth**, or 1,000 times the value of **_cRowsToTransfer**, rounded up to the nearest 512-byte multiple. 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 the **_ulClientBase** field is used as the lower 32 bits of a 64-bit value. For more information, see section 2.2.3.7.

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

Reserved2 (4 bytes): A 32-bit reserved field. This field MUST be set to 0x00000001.

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

Reserved4 (4 bytes): A 32-bit reserved filed. This field MUST be set to 0x00000000.

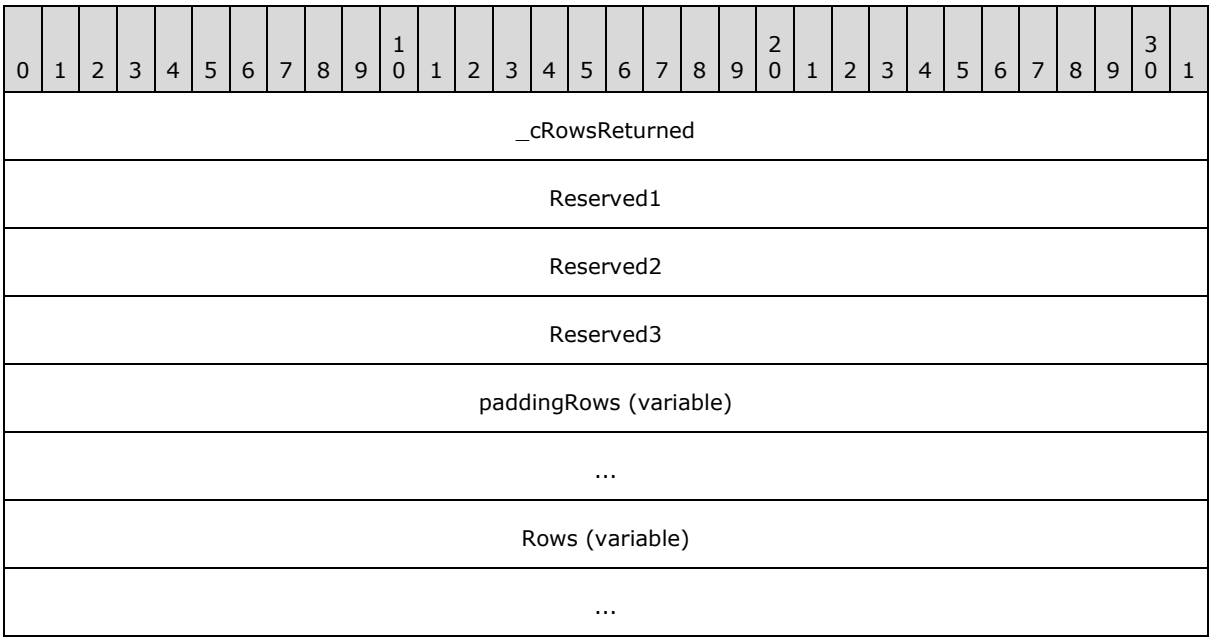
2.2.3.7 CPMGetRowsOut

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 by putting "0x00000102" in the **_iClientVersion** field of CPMConnectIn. Offsets are 32-bit integers.
- The protocol client indicated it was a 64-bit system by setting **_iClientVersion** to "0x00010102" in **CPMConnectIn**, and the protocol server indicated that it was a 32-bit system because **_serverVersion** is set to 0x00010102 in CPMConnectOut. Offsets are 32-bit integers.

- The protocol client indicated it was a 64-bit system because **_iClientVersion** is set to "0x00010102" in **CPMConnectIn**, and the protocol server indicated that it was a 64-bit system because **_serverVersion** is set to 0x00010102 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.

Reserved1 (4 bytes): A 32-bit reserved field. This field MUST be set to 0x000000.

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

Reserved3 (4 bytes): A 32-bit reserved field. This field MUST be ignored by the receiver.

paddingRows (variable): This field MUST be of sufficient length, from zero 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.

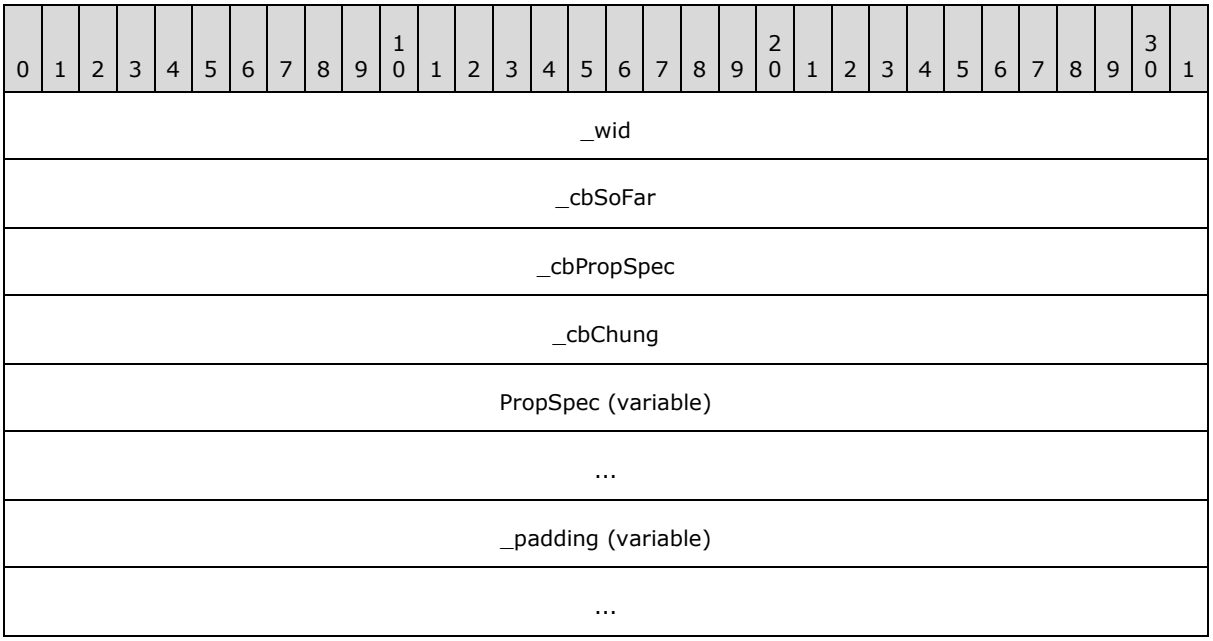
Rows (variable): Row data is formatted as prescribed by column information in the most recent CPMSetBindingsIn message. Rows MUST be stored in forward order. For example, row 1 must be stored before row 2. Fixed-sized columns MUST be stored at the offsets specified by the most recent **CPMSetBindingsIn**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

_cbChunk (4 bytes): A 32-bit unsigned integer containing the maximum number of bytes that the sender can accept in a CPMFetchValueOut message. 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", the following field values MUST be set on this structure. Otherwise, this structure MUST be omitted:

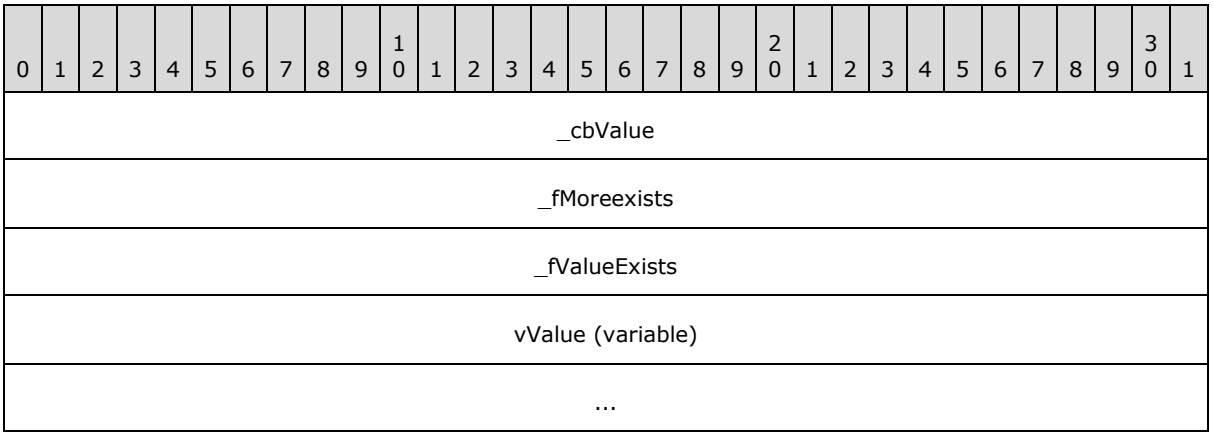
Field	Value
_guidPropSet	E83758B4-0C6E-435B-BCC6-268021EFAD6C
ulKind	PRSPEC_PROPID (0x00000001)
PrSpec	0x00000000

_padding (variable): This field MUST be zero to 3 bytes long to pad the message 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 3.1.5.5, this message is sent after each **CPMFetchValueIn** message until all bytes of the metadata are transferred. The message length, including the 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 total size of the **CPMFetchValueOut** message MUST be equal to the value of the **_cbChunk** field in **CPMFetchValueIn**.

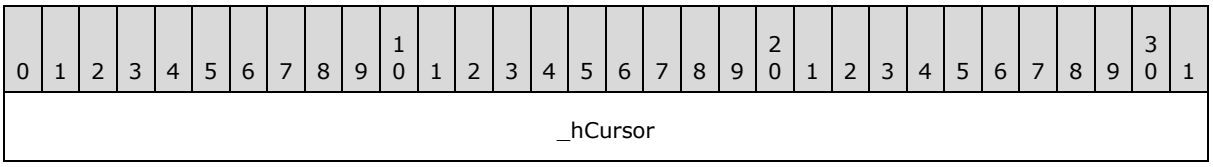
Value	Meaning
0x00000000	There is no additional data available.
0x00000001	There is additional data available.

_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

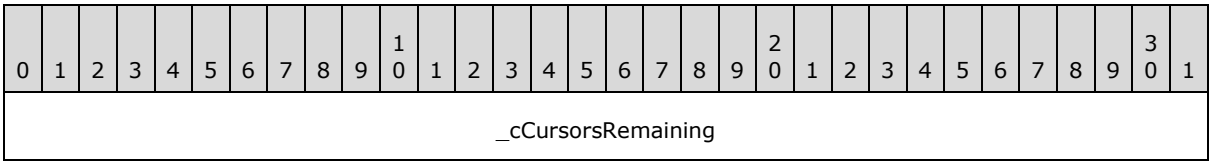
The **CPMFreeCursorIn** message requests the release of a cursor. The format of the **CPMFreeCursorIn** message that follows the header is shown in the following diagram.



_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 CPMGetNotifyIn

The **CPMGetNotifyIn** message requests a notification about query execution completion.

The message MUST NOT include a body; only the message header, as specified in section 2.2.2, is sent.

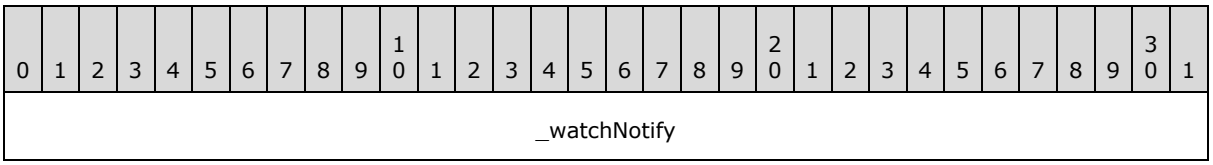
2.2.3.13 CPMGetNotifyOut

The **CPMGetNotifyOut** message replies to a CPMGetNotifyIn message indicating that the query execution is not complete yet. Once the query execution completes, the protocol server replies with an additional CPMSendNotifyOut message.

The message MUST NOT include a body; only the message header, as specified in section 2.2.2, is sent.

2.2.3.14 CPMSendNotifyOut

The **CPMSendNotifyOut** message replies to a CPMGetNotifyIn message indicating that the query execution has finished. The format of the **CPMSendNotifyOut** message that follows the header is shown in the following diagram.



_watchNotify (4 bytes): A 32-bit field. The value of the field can be any arbitrary value and MUST be ignored.

2.2.3.15 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 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, as described in section 1.8.

All error values MUST be treated the same; the error MUST be considered fatal and reported to the higher-layer caller. Future messages MAY be sent over the same pipe as if no error had occurred<7>.

3 Protocol Details

MSSearch Query Protocol message requests require only minimal sequencing. All messages MUST be preceded by an initial CPMConnectIn message. For example, a message is preceded by 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 responds 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.

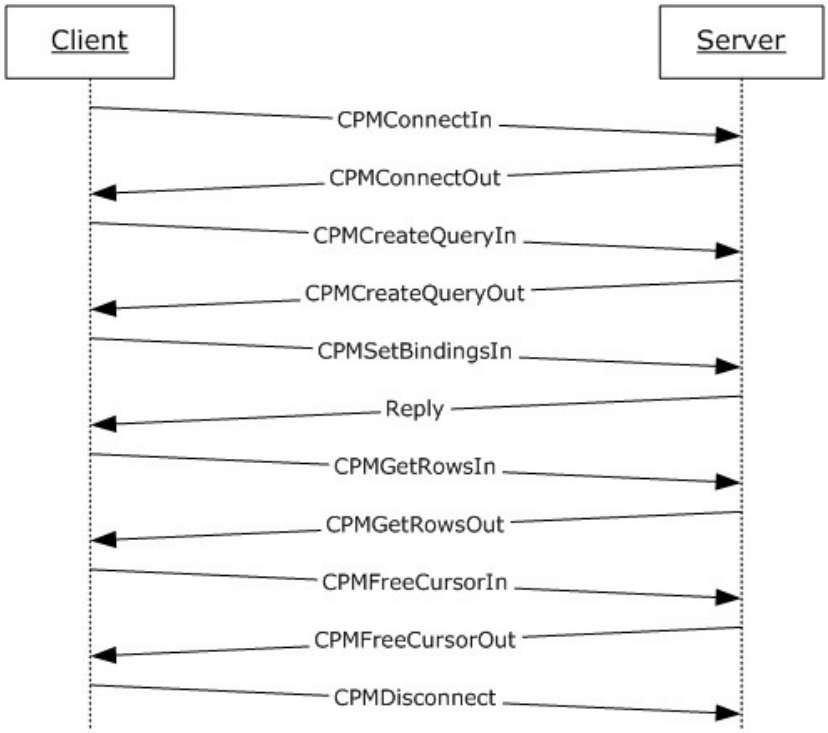


Figure 1: Typical message sequence for a simple query from protocol client to remote computer

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:
  - The protocol client's version, as specified in the CPMConnectIn message.
  - The search catalog associated with the protocol client by a **CPMConnectIn** message.
  - Additional client properties as specified in the database properties.
  - The protocol client's search query.
  - A list of cursor handles for the search query and the position in the result set for each handle.
  - The current set of bindings.
  - The current status of the search query, which includes for each cursor:
    - The number of rows in the query result.
    - The numerator and denominator of the query completion.
- The current state of the query server, which is "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 start listening for messages on the named pipe described 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 the 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, as specified in 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 it is 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 is determined 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 may be required as input to the later message. These requirements are detailed in the following sections. The following table summarizes the relationship between messages. An "X" in a column means that the protocol client MUST NOT send the message specified in the row before it receives the response specified in the column.

	CPMConnectOut	**CPMCreateQueryOut**	**CPMSetBindingsIn**	**CPMSendNotifyOut**
CPMConnectIn				
CPMCreateQueryIn	X			
CPMSetBindingsIn	X	X		
CPMGetNotifyIn	X	X		
CPMFetchValueIn	X	X	X	X
CPMGetRowsIn	X	X	X	X
CPMFreeCursorIn	X	X		

	CPMConnectOut	CPMCreateQueryOut	CPMSetBindingsIn	CPMSendNotifyOut
CPMDisconnect	X			

CPMSendNotifyOut is not required if the query was created as synchronous, as specified in **_uBooleanOptions** field of CRowsetProperties.

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. Check if the specified search catalog exists and is 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 (1) 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 a CPMCreateQueryOut response.
6. Initialize the number of rows to the currently calculated number of rows. This number can be zero if the search query did not start to execute, or some number if the search query is in a process of execution. Initialize the numerator and denominator of the 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 the binding information is valid. In order for it to be valid, the column at least specifies the value, length, or status to be returned, there is no overlap in bindings for value, length, or status, and value, length, and status fit in the row size specified. 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, 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 CPMGetNotifyIn Request

When the protocol server receives a CPMGetNotifyIn 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. If the query execution has not finished yet, prepare a CPMGetNotifyOut message and respond to the client with the message.
3. When query execution completes, prepare a CPMSendNotifyOut message and respond to the client with the message.

3.1.5.5 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, the protocol server MUST report the error encountered as an HRESULT or an NTSTATUS value, as described in 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 in 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. The string MUST contain space-delimited spelling

suggestions. If all spelling suggestions are the same as the original terms, vValue MUST contain a null terminated empty VT_LPWSTR.

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

For each query term the protocol server MUST determine a term identifier or 0x0000000. The protocol server MUST store the term identifiers in 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 the 32-bit value from steps 0-0 to a QUERYMETADATA structure and copy, starting from the **_cbSoFar** offset, at most **_cbChunk** bytes to the **vValue** field. Do not copy past the end of the serialized property. If this step fails, the protocol server MUST report an error.

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

3. If the length of the 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.

3.1.5.6 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 start of the query results. If this step fails, the protocol server MUST report the error encountered, which is an HRESULT or an NTSTATUS value, as described in 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, as specified in section 3.1.1. Store the actual number of rows fetched in **_cRowsReturned**.

6. Store fetched rows in the **Rows** field. For information about the structure of the Rows field, see section 2.2.3.7.

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

3.1.5.7 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 the associated resources for this cursor handle. For a complete list of resources, see section 3.1.1.
4. Remove the cursor from the list of cursors for that protocol client.
5. Respond with a CPMFreeCursorOut message, setting the **_cCursorsRemaining** field to 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 the associated resources, as specified in section 3.1.1.

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

The protocol server MAY use other timer events to interrupt execution of too expensive queries. When the protocol server receives a CPMCreateQueryIn request with the flag **IT** set in the **_uBooleanOptions** field of CRowsetProperties, these timer events MUST be ignored.

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. In other words, in the case of a failure to connect, it is the responsibility of the higher layer 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. 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 fails most of the messages. The higher layer 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. as specified in section 2.2.3.1, in **_iClientVersion**, **MachineName**, **UserName**, **PropertySets** and **ExtPropertySets**.

2. Set **_fClientIsRemote**, **_cbBlob1**, **_cbBlob2**, **cPropSets**, 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 on successful connection. This is not enforced by the protocol client.

- Use a CPMCreateQueryIn request to create a search query with the 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 (1) set, columns set, an optional sort order, 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 (1) 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 the 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, as specified in 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, as specified in 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 **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. This is not enforced by this protocol.

3.2.4.1.4 Sending a CPMGetNotifyIn Request

The client MAY request notification of query execution completion using a CPMGetNotifyIn message when it already has a valid cursor handle after successfully receiving CPMCreateQueryOut, as specified in section 3.2.5.1. If the **_uBooleanOptions** field of CRowsetProperties used in the CPMCreateQueryIn message indicates an asynchronous query, the client MUST request this notification before sending CPMFetchValueIn or CPMSetBindingsIn messages.

1. Prepare a CPMGetNotifyIn message and send it to the protocol server.

2. Wait to receive a CPMSendNotifyOut message from the protocol server, discarding other messages.

3. If the **_status** field of the response indicates an error, handle errors as specified in section 2.2.4.

The higher layers SHOULD then request a protocol client to send a CPMGetRowsIn message. This is not enforced by this protocol.

3.2.4.1.5 Sending a CPMGetRowsIn Request

When the higher layer is about to receive rows data, it provides 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.
   - The 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 the **_ulClientBase** field<8>.

3. Set the value of **_cbSeek**, which acts as an offset for **Rows** start, to "0x0000000C".

4. Send a CPMGetRowsIn message to the protocol server.

3.2.4.1.6 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.

3.2.4.1.7 Sending a CPMFreeCursorIn Request

After the higher layer 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 **CPMFreeCursorIn** message to the protocol server.

2. Wait to receive a CPMFreeCursorOut message from the protocol server, discarding other messages.

3.2.4.1.8 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 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 the 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 protocol client:

- Use CPMSetBindingsIn to set bindings for individual columns and do any subsequent actions on the query path.
- If the query CRowsetProperties indicated an asynchronous query, use a CPMGetNotifyIn message to wait for query execution completion.

3.2.5.2 Receiving a CPMGetNotifyOut Response

When the protocol client receives a CPMGetNotifyOut 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 step #2.
2. Keep waiting to receive a CPMSendNotifyOut message from the protocol server, discarding other messages.

3.2.5.3 Receiving a CPMSendNotifyOut Response

When the protocol client receives a CPMSendNotifyOut message response from the protocol server, the protocol client MUST return **_status** back to the higher layer.

3.2.5.4 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 step #2.
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 layer.

3.2.5.5 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 layer.

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.

The higher layers SHOULD do the following actions, but these are not enforced by the protocol client:

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

3.2.5.6 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 in **_cCursorsRemaining** is "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, 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 the catalog SYSTEM. Assume that both computer A and X are running a 64-bit version of the Windows Server® 2008 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 server X. The protocol client uses the named pipe \pipe\OSearch14 to connect to server X over SMB2.
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 **cPropSets** 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 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 "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:
     - **Type** is set to "0x00000004" (RTContent).
     - **SubType** is set to "0x00000000".
     - **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 "0x409" (English).
       - **_ulGenerateMethod** is set to "0x00000000" (exact match).
   - **CSortSetPresent** is set to "0x00".
   - **RowSetProperties** is set as follows:
     - **_uBooleanOptions** is set to "0x00008019" (asynchronous, 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).
     - **_guidRankingModelId** is set to GUID "00000000-0000-0000-0000-000000000000" (use the default ranking model).
     - **_guidUserId** is set to GUID "07C76175-DB7F-4850-AC6C-7C2711661B29" (the user's user profile identifier).

- **_guidCorrelationId** is set to GUID "C79B2F66-6B57-4028-8364-966F387DDDA4" (random GUID to identify the query).

- **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" (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. In this example, 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", which is big enough to fit the 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", which is the column transferred in the row.
  - **_ValueOffset** is set to "0x0008" (at beginning of row).
  - **_ValueSize** is set to "0x10", which is the 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.

   - 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 CPMGetNotifyIn request message to wait for query execution completion.

   - The header of the message is populated as follows:
     - **_msg** is set to "0x000000D1", indicating that this is a **CPMGetNotifyIn** message.
     - **_status** is set to "0x00000000".
     - **_ulChecksum** contains the checksum, computed as specified in section 3.2.4.
     - **_ulReserved2** is set to "0x00000000".

10. The protocol server processes it and, assuming that the query is still being executed, responds with a CPMGetNotifyOut message.

    - The header of the message is populated as follows:
      - **_msg** is set to "0x000000D1".
      - **_status** is set to "SUCCESS".
      - **_ulChecksum** is set to "0x00000000" or any other arbitrary value.
      - **_ulReserved2** is set to "0x00000000".

11. The client regularly polls the query server for query completion, as described in step 9. After the query execution completes, the protocol server responds with a CPMSendNotifyOut message.

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

        - **_msg** is set to "0x000000D2".
        - **_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:

        - **_watchNotify** is set to "0x00000000" or any other arbitrary value.

12. The protocol client issues a CPMGetRowsIn request message, assuming that the protocol client is prepared to accept 100 rows 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", which is 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".

13. 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 structured as a list of 18, 48-byte CRowVariants that contain document identifiers.

14. The protocol client sends a CPMDisconnect message to end the connection.
    - 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".
15. 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 SMB2, as specified in [MS-SMB2], 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 released service packs:

- Microsoft® FAST™ Search Server 2010
- Microsoft® SharePoint® Foundation 2010

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

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

<1> Section 1.4: Applications SHOULD 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: Search uses only the values specified in [MS-ERREF].

<3> Section 1.8: See [MSDN-PROPSET] for a list of supported property sets.

<4> Section 2.2.1.19: In SharePoint Foundation 2010 implementation, alternative spellings are not generated and the **SpellingSuggestion** field always contains an empty string.

<5> Section 2.2.2: In Search Server 2010, Search Server 2010 Express, SharePoint Foundation 2010 and SharePoint Server 2010 implementations, the protocol client always sets the **_status** field to 0x00000000.

<6> Section 2.2.3.1: In Search Server 2010, Search Server 2010 Express, SharePoint Foundation 2010 and SharePoint Server 2010 implementations, the **_iClientVersion** is always set to 0x00010102.

<7> 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 the pipe is closed, the server releases all resources associated with the connection, including the named pipe instance.

<8> Section 3.2.4.1.5: 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

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

8 Index

T

V