[MS-WSP]:
Windows Search Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- This protocol 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 protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact protocol@microsoft.com.
- 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.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
04/10/2007	1.0		Version 1.0 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.
07/10/2007	1.2.2	Editorial	Revised and edited the technical content.
08/17/2007	1.3	Minor	Revised technical content based on feedback.

Date	Revision History	Revision Class	Comments
09/21/2007	1.4	Minor	Revised technical and editorial content based on feedback.
10/26/2007	1.4.1	Editorial	Revised and edited the technical content.
01/25/2008	1.5	Minor	Updated the technical content.

Table of Contents

1 Introduction 7
1.1 Glossary 7
1.2 References 8
1.2.1 Normative References 8
1.2.2 Informative References 9
1.3 Protocol Overview (Synopsis) 9
1.3.1 Remote Administration Tasks 10
1.3.2 Remote Querying 10
1.4 Relationship to Other Protocols 11
1.5 Prerequisites/Preconditions 11
1.6 Applicability Statement 11
1.7 Versioning and Capability Negotiation 11
1.8 Vendor-Extensible Fields 11
1.8.1 Property IDs 11
1.9 Standards Assignments 11

2 Messages 12
2.1 Transport 12
2.2 Message Syntax 12
2.2.1 Structures 12
2.2.1.1 CBaseStorageVariant 14
2.2.1.1.1 CBaseStorageVariant Structures 18
2.2.1.1.1.1 DECIMAL 18
2.2.1.1.1.2 VT_VECTOR 19
2.2.1.1.1.3 SAFEARRAY 19
2.2.1.1.1.4 SAFEARRAYBOUND 21
2.2.1.1.1.5 SAFEARRAY2 21
2.2.1.1.1.6 VT_COMPRESSED_LPWSTR 21
2.2.1.2 CFullPropSpec 22
2.2.1.3 CContentRestriction 24
2.2.1.4 CKey 25
2.2.1.5 CInternalPropertyRestriction 26
2.2.1.6 CNatLanguageRestriction 27
2.2.1.7 CNodeRestriction 28
2.2.1.8 CPropertyRestriction 29
2.2.1.9 CScopeRestriction 32
2.2.1.10 CSort 33
2.2.1.11 CVectorRestriction 34
2.2.1.12 CCoercionRestriction 35
2.2.1.13 CRelDocRestriction 35
2.2.1.14 CProbRestriction 36
2.2.1.15 CFeedbackRestriction 38
2.2.1.16 CRestrictionArray 38
2.2.1.17 CRestriction 39
2.2.1.18 CColumnSet 41
2.2.1.19 CCategorizationSet 41
2.2.1.20 CCategorizationSpec 42
2.2.1.21 CCategSpec 43
2.2.1.22 CRangeCategSpec 44
2.2.1.23 RANGEBOUNDARY 45
2.2.1.24 CAggregSet 47
2.2.1.25 CAggregSpec 47

2.2.1.26 CSortAggregSet 48
2.2.1.27 CAggregSortKey 49
2.2.1.28 CInGroupSortAggregSets 49
2.2.1.29 CInGroupSortAggregSet 50
2.2.1.30 CDbColId 51
2.2.1.31 CDbProp 52
2.2.1.31.1 Database Properties 52
2.2.1.32 CDbPropSet 54
2.2.1.33 CPidMapper 56
2.2.1.34 CColumnGroupArray 56
2.2.1.35 CColumnGroup 57
2.2.1.36 SProperty 57
2.2.1.37 CRowSeekAt 57
2.2.1.38 CRowSeekAtRatio 58
2.2.1.39 CRowSeekByBookmark 58
2.2.1.40 CRowSeekNext 59
2.2.1.41 CRowsetProperties 59
2.2.1.42 CRowVariant 61
2.2.1.43 CSortSet 61
2.2.1.44 CTableColumn 62
2.2.1.45 SERIALIZEDPROPERTYVALUE 64
2.2.2 Message Headers 65
2.2.3 Messages 66
2.2.3.1 CPMCiStateInOut 66
2.2.3.2 CPMUpdateDocumentsIn 69
2.2.3.3 CPMForceMergeIn 70
2.2.3.4 CPMConnectIn 71
2.2.3.5 CPMConnectOut 74
2.2.3.6 CPMCreateQueryIn 74
2.2.3.7 CPMCreateQueryOut 76
2.2.3.8 CPMGetQueryStatusIn 77
2.2.3.9 CPMGetQueryStatusOut 77
2.2.3.10 CPMGetQueryStatusExIn 78
2.2.3.11 CPMGetQueryStatusExOut 78
2.2.3.12 CPMSetBindingsIn 79
2.2.3.13 CPMGetRowsIn 80
2.2.3.14 CPMGetRowsOut 82
2.2.3.15 CPMRatioFinishedIn 84
2.2.3.16 CPMRatioFinishedOut 85
2.2.3.17 CPMFetchValueIn 86
2.2.3.18 CPMFetchValueOut 87
2.2.3.19 CPMGetNotify 88
2.2.3.20 CPMSendNotifyOut 88
2.2.3.21 CPMGetApproximatePositionIn 88
2.2.3.22 CPMGetApproximatePositionOut 89
2.2.3.23 CPMCompareBmkIn 89
2.2.3.24 CPMCompareBmkOut 90
2.2.3.25 CPMRestartPositionIn 90
2.2.3.26 CPMStopAsynchIn 91
2.2.3.27 CPMFreeCursorIn 91
2.2.3.28 CPMFreeCursorOut 91
2.2.3.29 CPMDisconnect 92
2.2.4 Errors 92
2.2.5 Standard Properties 92
2.2.5.1 Query Properties 92

2.2.5.2 Common Open Properties 93
3 Protocol Details 102
3.1 Server Details....103
3.1.1 Abstract Data Model....103
3.1.2 Timers 104
3.1.3 Initialization....104
3.1.4 Higher-Layer Triggered Events 104
3.1.5 Message Processing and Sequencing Rules 104
3.1.5.1 Remote Windows Search Service Catalog Management....106
3.1.5.1.1 Receiving a CPMCiStateInOut Request 106
3.1.5.1.2 Receiving a CPMUpdateDocumentsIn Request 106
3.1.5.1.3 Receiving a CPMForceMergeIn Request....106
3.1.5.2 Remote Windows Search Service Querying....107
3.1.5.2.1 Receiving a CPMConnectIn Request 107
3.1.5.2.2 Receiving a CPMCreateQueryIn Request 107
3.1.5.2.3 Receiving a CPMGetQueryStatusIn Request 108
3.1.5.2.4 Receiving a CPMGetQueryStatusExIn Request 108
3.1.5.2.5 Receiving a CPMRatioFinishedIn Request 108
3.1.5.2.6 Receiving a CPMGetRowsIn Request 109
3.1.5.2.7 Receiving a CPMFetchValueIn Request 110
3.1.5.2.8 Receiving a CPMSetBindingsIn Request 111
3.1.5.2.9 Receiving a CPMGetNotify Request 111
3.1.5.2.10 Receiving a CPMGetApproximatePositionIn Request 111
3.1.5.2.11 Receiving a CPMCompareBmkIn Request....112
3.1.5.2.12 Receiving a CPMRestartPositionIn Request....112
3.1.5.2.13 Receiving a CPMStopAsynchIn Request 113
3.1.5.2.14 Receiving a CPMFreeCursorIn Request 113
3.1.5.2.15 Receiving a CPMDisconnect Request 113
3.1.6 Timer Events....113
3.1.7 Other Local Events....113
3.2 Client Details....114
3.2.1 Abstract Data Model....114
3.2.2 Timers 114
3.2.3 Initialization....114
3.2.4 Higher-Layer Triggered Events 114
3.2.4.1 Remote Windows Search Service Catalog Management....114
3.2.4.1.1 Sending a CPMCiStateInOut Request....115
3.2.4.1.2 Sending a CPMUpdateDocumentsIn Request 115
3.2.4.1.3 Sending a CPMForceMergeIn Request....115
3.2.4.2 Remote Windows Search Service Catalog Query Messages 115
3.2.4.2.1 Sending a CPMConnectIn Request 116
3.2.4.2.2 Sending a CPMCreateQueryIn Request 116
3.2.4.2.3 Sending a CPMSetBindingsIn Request 117
3.2.4.2.4 Sending a CPMGetRowsIn Request 117
3.2.4.2.5 Sending a CPMFetchValueIn Request 118
3.2.4.2.6 Sending a CPMFreeCursorIn Request 118
3.2.4.2.7 Sending a CPMDisconnect Message....118
3.2.5 Message Processing and Sequencing Rules 118
3.2.5.1 Receiving a CPMCreateQueryOut Response 118
3.2.5.2 Receiving a CPMGetRowsOut Response....119
3.2.5.3 Receiving a CPMFetchValueOut Response....120
3.2.5.4 Receiving a CPMFreeCursorOut Response....120
3.2.6 Timer Events....120
3.2.7 Other Local Events....120

4 Protocol Examples 121
4.1 Example 1 121
5 Security 136
5.1 Security Considerations for Implementers 136
5.2 Index of Security Parameters 136
6 Appendix A: Windows Behavior 137
7 Index 138

1 Introduction

This document specifies the Windows Search Protocol, which allows a client to issue queries to a server hosting a Windows Search service (WSS). The protocol is primarily intended to be used for full-text queries. It also allows an administrator to remotely manage the WSS.

This document specifies both remote querying and remote administration of Windows Search service (WSS) catalogs.

1.1 Glossary

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

GUID
HRESULT
Named Pipe
Path

The following terms are specific to this document:

Binding: A request to include a particular **column** in a returned **rowset**. The binding specifies a property to be included in the search results.

Bookmark: A marker that uniquely identifies a **row** within a set of **rows**.

Catalog: The highest-level unit of organization in the **Windows Search service**. It represents a set of indexed documents against which queries can be executed by using the Windows Search Protocol.

Category: A hierarchical grouping of **rows**. For example, a query result that contains author and title **columns** can be categorized based on author. Each group of **rows** containing the same value for author would constitute a category.

Chapter: A range of **rows** within a set of **rows**.

Column: The container for a single type of information in a **row**. Columns map to property names and specify which properties are used for the search query's **command tree** elements.

Command Tree: A combination of **restrictions**, **categories**, and **sort orders** that are specified for the search query.

Cursor: An entity that is used as a mechanism to work with one **row** or a small block of **rows** at a time in a set of data returned in a result set. A cursor is positioned on a single **row** within the result set. After the cursor is positioned on a **row**, operations can be performed on that **row**, or on a block of **rows** starting at that position.

Handle: A token that can be used to identify and access **cursors**, **chapters**, and **bookmarks**.

Indexing: The process of extracting text and properties from files and storing the extracted values into the **indexes** (for text), and the **property cache** (for properties).

Inverted Index: A persistent structure that contains the text content pulled out of files by the **Windows Search service**. The text is organized into an inverted index that maps from a word in a property to a list of the documents and locations within a document that contain that word.

Locale: An identifier, as specified in [MS-LCID], that specifies preferences related to language. These preferences indicate how dates and times are to be formatted, how items are to be sorted alphabetically, how strings are to be compared, and so on.

Natural Language Query: A query constructed using human language instead of query syntax. The **Windows Search service** is free to interpret the query in order to determine the best results. The interpretation is explicitly not specified in order to allow improvements over time.

Noise Word: A word that is ignored by the **Windows Search service** when present in the **restrictions** specified for the search query, because it has little discriminatory value. English examples include "a," "and," and "the."

Object: A file, e-mail, e-mail attachment, contact, calendar appointment or any other self-contained item that can be **indexed** and searched for by the **Windows Search service**.

Property Cache: A cache of **object** properties extracted by a **Windows Search service**.

Restriction: A set of conditions that a file must meet to be included in the search results returned by the **Windows Search service** in response to a search query. A restriction narrows the focus of a search query, limiting the files that the **Windows Search service** will include in the search results to only those that match the conditions.

Row: The collection of **columns** containing the property values that describe a single result from the set of **objects** that matched the **restrictions** specified in the search query submitted to the **Windows Search service**.

Rowset: A set of **rows** returned in the search results.

Sort Order: The set of rules in a search query that define the ordering of **rows** in the search result. Each rule consists of a property (for example, name or size) and a direction for the ordering (ascending or descending). Multiple rules are applied sequentially.

Windows Search Service (WSS): A service that creates **indexed catalogs** for the contents and properties of file systems. Applications can search the **catalogs** for information from the files on the indexed file system.

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

1.2 References

1.2.1 Normative References

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

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://grouper.ieee.org/groups/754/

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", July 2006.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

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

[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference", March 2007.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2006.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2006.

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

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

If you have any trouble finding [SALTON], please check here.

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

1.2.2 Informative References

[JONES] Sparck Jones, K., Walker, S., and Robertson, S.E., "A Probabilistic Model of Information and Retrieval: Development and Status", September 1998, University of Cambridge Technical Report UCAM-CL-TR-446.

[MSDN-FULLPROPSPEC] Microsoft Corporation, "FULLPROPSPEC", http://msdn2.microsoft.com/en-us/library/ms690996.aspx

[MSDN-OLEDBP] Microsoft Corporation, "OLE DB Provider for Indexing Service", http://msdn2.microsoft.com/en-us/library/ms690319.aspx

[MSDN-QUERYERR] Microsoft Corporation, "Query-Execution Values", http://msdn2.microsoft.com/en-us/library/ms690617.aspx

1.3 Protocol Overview (Synopsis)

A **Windows Search service** helps to efficiently organize the extracted features of a collection of documents. The Windows Search Protocol allows a client to communicate with a server hosting a Windows Search service, both to issue queries and to allow an administrator to manage the **indexing** server.

When processing files, a Windows Search service analyzes a set of documents, extracts useful information, and then organizes the extracted information in such a way that properties of those documents can be efficiently returned in response to queries. A collection of documents that can be queried comprises a **catalog**. A catalog may contain an **inverted index** (for quick word matching) and a **property cache** (for quick retrieval of property values).

Conceptually, a catalog consists of a logical "table" of properties with the text or value and corresponding **locale** stored in **columns** of the table. Each **row** of the table corresponds to a separate document in the scope of the catalog, and each column of the table corresponds to a property.

The specific tasks performed by the Windows Search Protocol are grouped into two functional areas:

- Remote administration of Windows Search service catalogs.
- Remote querying of Windows Search service catalogs.

1.3.1 Remote Administration Tasks

The Windows Search Protocol enables the following Windows Search service catalog management tasks from a client:

- Query the current state of a Windows Search service catalog on the server.
- Update the state of a Windows Search service catalog.
- Launch the indexing process for a particular set of files.
- Initiate optimization of an index to improve query performance.

All remote administration tasks follow a simple request/response model. No state is maintained on the client for any administration call, and administrative calls can be made in any order.

1.3.2 Remote Querying

The Windows Search Protocol enables clients to perform search queries against a remote server hosting a Windows Search service.

Sending a search query is a multi-step process initiated by the client. The steps are as follows:

1. The client requests a connection to a server hosting a Windows Search service.
2. The client sends the parameters for the search query, which include:
   - **Rowset** properties like the catalog name and configuration information.
   - The **restrictions** to specify which documents are to be included and/or excluded from the search results.
   - The order in which the search results are to be returned.
   - The columns to be returned in the result set.
   - The maximum number of rows that should be returned for the query.
   - The maximum time for query execution.
3. After the server has acknowledged the client's request to initiate the query, the client can request status information about the query, but this is not a required step.
4. The client then specifies which properties the server should include in the search results.
5. The client requests a result set from the server, and the server responds by sending the client the property values for files that were included in the results for the client's search query. If the value of a property is too large to fit in a single response buffer, the server will not send the property; instead, it will set the property status to deferred. The client then requests the property value separately using a series of requests for successive chunks of the value and then resumes requesting other values.
6. After the client is finished with the search query and no longer requires additional results, the client contacts the server to release the query.
7. After the server has released the query, the client may send a request to disconnect from the server. The connection is then closed. Alternatively, the client may issue another query and repeat the sequence from the step 2.

1.4 Relationship to Other Protocols

The Windows Search Protocol relies on the SMB Protocol (as specified in [MS-SMB]) for message transport. No other protocol depends directly on the Windows Search Protocol.<1>

1.5 Prerequisites/Preconditions

It is assumed the client has obtained the name of the server and a catalog name before this protocol is invoked.<2> It is also assumed that the client and server have a security association usable with **named pipes** (see [MS-SMB]).

1.6 Applicability Statement

The Windows Search Protocol is designed for querying and managing catalogs on a remote server from a client.<3>

1.7 Versioning and Capability Negotiation

This protocol has no versioning or capability negotiation mechanisms.

1.8 Vendor-Extensible Fields

This protocol uses **HRESULTs** 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 the value is a customer code.

This protocol also uses **NTSTATUS** values taken from the NTSTATUS number space defined in [MS-ERREF]. Vendors SHOULD<4> reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

1.8.1 Property IDs

Properties are represented by IDs known as property IDs. Each property must have a **globally unique identifier (GUID)**. This identifier consists of a GUID representing a collection of properties called a property set plus either a string or a 32-bit integer to identify the property within the set. If the integer form of ID is used, then the values 0x00000000, 0xFFFFFFFF, and 0xFFFFFFFE are considered invalid.

Vendors can guarantee that their properties are uniquely defined by placing them in a property set defined by their own GUIDs.

1.9 Standards Assignments

This protocol has no standards assignments, only private assignments made by Microsoft using allocation procedures specified in other protocols.

Microsoft has allocated this protocol a named pipe as specified in [MS-SMB]. The assignment is:

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

2 Messages

The following sections specify how messages are transported and provide details of message syntax, including common structures, certificate requirements, and common error codes.

2.1 Transport

All messages MUST be transported using a named pipe, as specified in [MS-SMB]. The following pipe name is used:

- \pipe\MSFTEWDS

This protocol uses the underlying SMB named pipe protocol to retrieve the identity of the caller that made the connection as specified in [MS-SMB] section 2.2.8. The client MUST set SECURITY_IDENTIFICATION as the **ImpersonationLevel** in the request to open the named pipe.

2.2 Message Syntax

Several structures and messages in the following sections refer to **chapter** or **bookmark handles**. A handle is a 32-bit long opaque structure that uniquely identifies a chapter or bookmark. Typically, client applications receive handle values via method calls. However, there are several well-known values that need not be obtained from a server, the meaning of which is specified in the following table.

Value	Meaning
DB_NULL_HCHAPTER 0x00000000	A chapter handle to the unchaptered rowset that contains all query results.
DBBMK_FIRST 0x00000001	A bookmark handle to a bookmark that identifies the first row in the rowset.
DBBMK_LAST 0x00000002	A bookmark handle to a bookmark that identifies the last row in the rowset.

2.2.1 Structures

This section details data structures that are defined and used by the Windows Search Protocol.

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

The following table summarizes the data structures defined in this section.

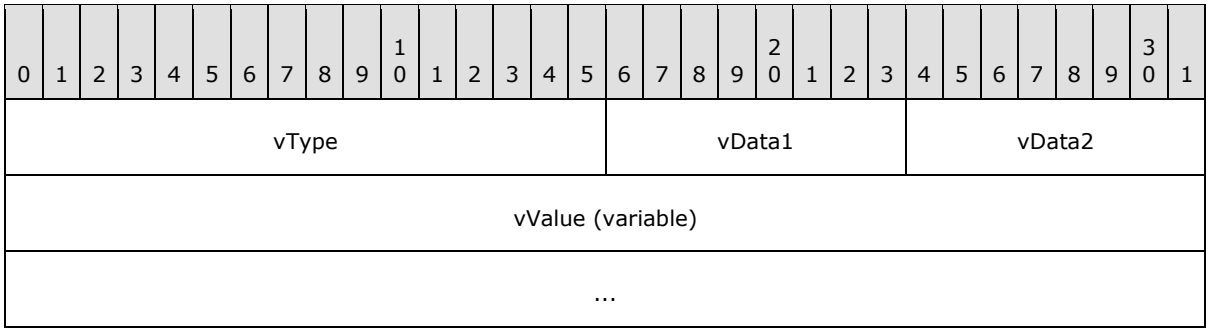
Structure	Description
CBaseStorageVariant	Contains the value on which to perform a match operation for a property that is specified in a CPropertyRestriction structure.
SAFEARRAY, SAFEARRAY2	Contains a multidimensional array.
SAFEARRAYBOUND	Represents the bounds for a dimension of an array specified in a SAFEARRAY structure.

Structure	Description
CFullPropSpec	Contains a property specification.
CContentRestriction	Contains a string to match for a property value in the property cache.
CKey	Contains a property value.
CInternalPropertyRestriction	Contains a property value to match with an operation.
CNatLanguageRestriction	Contains a **natural language query** match for a property.
CNodeRestriction	Contains an array of **command tree** nodes specifying the restrictions for a query.
CPropertyRestriction	Contains a property value to match with an operation.
CScopeRestriction	Contains a restriction on the files to be searched.
CSort	Identifies a column to sort.
CVectorRestriction	Contains an array of command tree nodes specifying the restrictions for a vector space array query. For details, see [SALTON].
CRestriction	A restriction node in a query command tree.
CRestrictionArray	Contains a counted array of restrictions.
CColumnSet	Describes the columns to return.
CCategorizationSet	A set of CCategorizationSpec structures where each describes the grouping property for one level in a hierarchical result set.
CCategorizationSpec	Specifies the **categorization** property used to **categorize** results at one level in a hierarchical result set.
CCategSpec	Contains a grouping specification.
CRangeCategSpec	Contains range information for grouping by ranges.
CDbColId	Contains a column identifier.
CDbProp	Contains a rowset property.
CDbPropSet	Contains a set of rowset properties.
CPidMapper	Maps from message internal property IDs (PIDs) to full property specifications.
CRowSeekAt	Contains the offset at which to retrieve rows in a CPMGetRowsIn message.
CRowSeekAtRatio	Identifies the approximate point expressed as a ratio at which to begin retrieval for a CPMGetRowsIn message.
CRowSeekByBookmark	Identifies the bookmarks from which to retrieve rows for a CPMGetRowsIn message.
CRowSeekNext	Contains the number of rows to skip in a CPMGetRowsIn message.
CRowsetProperties	Contains the configuration information for a query and is specified as OLE-

Structure	Description
	DB rowset properties.
CSortSet	Contains the **sort orders** for a query.
CTableColumn	Contains a column for the CPMSetBindingsIn message.
SERIALIZEDPROPERTYVALUE	Contains a serialized value.
CCoercionRestriction	Contains a coercion restriction that affects ranking of query results.
CFeedbackRestriction	Contains a set of feedback documents for relevance feedback queries (for more information, see [JONES]).
CProbRestriction	Contains probabilistic rank parameters (for more information, see [JONES]).
CRelDocRestriction	Contains a relevant document for relevance feedback queries (for more information, see [JONES]).
CAggregSet	Contains a set of aggregate specifications.
CAggregSpec	Contains a single aggregate or column specification.
CAggregSortKey	Contains a single grouping sort key.
CSortAggregSet	Contains a set of grouping sort keys.
CInGroupSortAggregSet	Contains sorting information for a group with regards to a parent group.
CInGroupSortAggregSets	Contains sorting information for a group with regards to one or more parent groups.

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.



vType: 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	**vValue** is not present.

Value	Meaning
0x0000	
VT_NULL 0x0001	**vValue** is not present.
VT_I1 0x0010	A 1-byte signed integer.
VT_UI1 0x0011	A 1-byte unsigned integer.
VT_I2 0x0002	A 2-byte signed integer.
VT_UI2 0x0012	A 2-byte unsigned integer.
VT_BOOL 0x000B	A Boolean value; a 2-byte integer. **Note** Contains 0x0000 (FALSE) or 0xFFFF (TRUE).
VT_I4 0x0003	A 4-byte signed integer.
VT_UI4 0x0013	A 4-byte unsigned integer.
VT_R4 0x0004	An IEEE 32-bit floating point number, as defined in [IEEE754].
VT_INT 0x0016	A 4-byte signed integer.
VT_UINT 0x0017	A 4-byte unsigned integer. Note that this is identical to VT_UI4, except that VT_UINT cannot be used with VT_VECTOR (defined below); the value chosen is up to the higher layer that provides it to the Windows Search Protocol Specification, but the Windows Search Protocol Specification treats VT_UINT and VT_UI4 as identical with the exception noted above.
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 defined 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 representing the number of days since 00:00:00 on December 31, 1899 (Coordinated Universal Time).
VT_FILETIME	A 64-bit integer representing the number of 100-nanosecond intervals

Value	Meaning
0x0040	since 00:00:00 on January 1, 1601 (Coordinated Universal Time).
VT_DECIMAL 0x000E	A DECIMAL structure as specified in section 2.2.1.1.1.1.
VT_CLSID 0x0048	A 16-byte binary value containing a GUID.
VT_BLOB 0x0041	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 below under **vValue**.
VT_LPSTR 0x001E	A null-terminated ANSI string.
VT_LPWSTR 0x001F	A null-terminated Unicode (for details, see [UNICODE]) string.
VT_COMPRESSED_LPWSTR 0x0023	A compressed version of a null-terminated Unicode (for details, see [UNICODE]) string as specified in section 2.2.1.1.1.6.
VT_VARIANT 0x000C	CBaseStorageVariant.

The following table specifies the type modifiers for **vType**. Type modifiers can be binary ORed with **vType** to change the meaning of **vValue** to indicate it is one of two possible array types.

Value	Meaning
VT_VECTOR 0x1000	If the type indicator is combined with VT_VECTOR by using an OR operator, **vValue** is a counted array of values of the indicated type. See section 2.2.1.1.1.2. This type modifier MUST NOT be combined with the following types: VT_INT, VT_UINT, VT_DECIMAL, VT_BLOB, and VT_BLOB_OBJECT.
VT_ARRAY 0x2000	If the type indicator is combined with VT_ARRAY by an OR operator, the value is a SAFEARRAY containing values of the indicated type. This type modifier MUST NOT be combined with the following types: VT_I8, VT_UI8, VT_FILETIME, VT_CLSID, VT_BLOB, VT_BLOB_OBJECT, VT_LPSTR, and VT_LPWSTR.

vData1: When **vType** is VT_DECIMAL, the value of this field is specified as the **Scale** field in section 2.2.1.1.1.1. For all other **vTypes**, the value MUST be set to 0x00.

vData2: When **vType** is VT_DECIMAL, the value of this field is specified as the **Sign** field in section 2.2.1.1.1.1. For all other **vTypes**, the value MUST be set to 0x00.

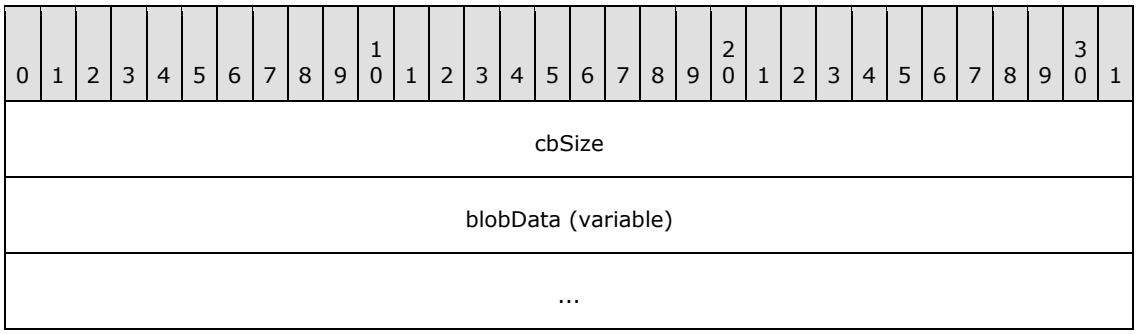
vValue: The value for the match operation. The syntax MUST be as indicated in the **vType** field.

The following table summarizes sizes for the **vValue** field, dependent upon the **vType** field for fixed-length data types. The size is in bytes.

vType	Size
VT_I1, VT,_UI1	1

vType	Size
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_DECIMAL, VT_CLSID	16

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



cbSize: A 32-bit unsigned integer.

Note Indicates the size of the **blobData** field in bytes. If **vType** is set to VT_BSTR or VT_LPSTR, cbSize MUST be set to 0x00000000 when the string represented is an empty string.

blobData: MUST be of length **cbSize** in bytes.

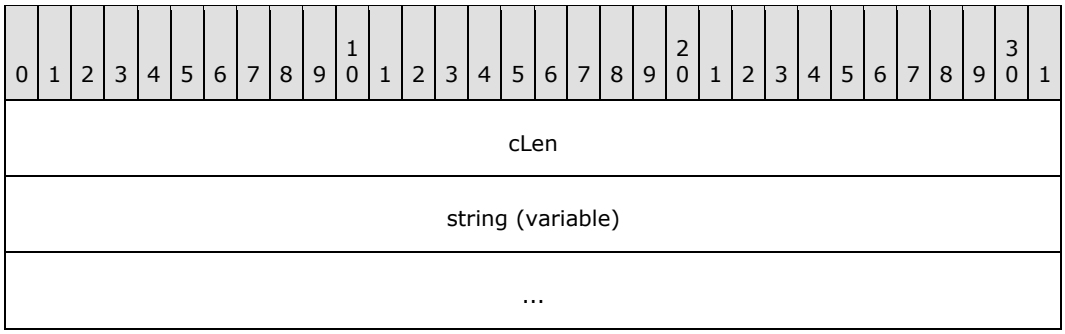
For **vType** set to VT_BLOB, this field is opaque binary blob data.

For **vType** set to VT_BSTR, this field is a set of characters in an OEM-selected character set. The client and server MUST be configured to have interoperable character sets. 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 diagram below, with the following caveats:

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

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



cLen: A 32-bit unsigned integer, indicating the size of the **string** field including the terminating null.

Note A value of 0x00000000 indicates that no such string is present.

string: Null-terminated string.

Note This field MUST be absent if **cLen** equals 0x00000000.

2.2.1.1.1 CBaseStorageVariant Structures

The following structures are used in the CBaseStorageVariant structure.

2.2.1.1.1.1 DECIMAL

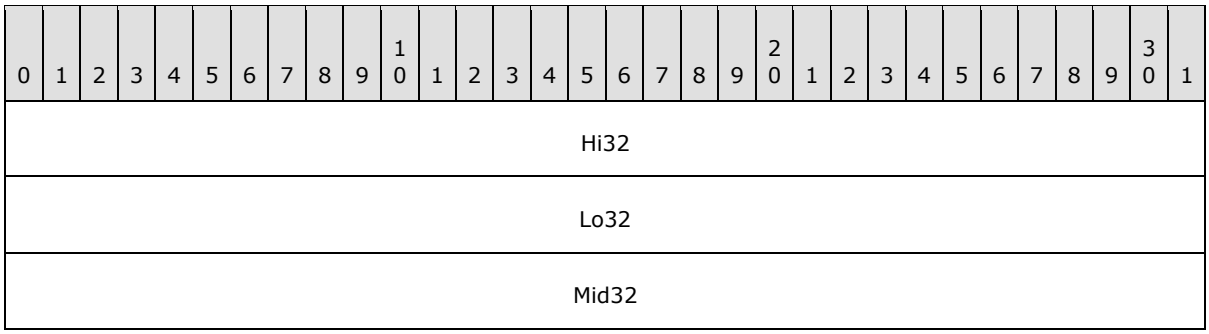
DECIMAL is used to represent an exact numeric value with a fixed precision and fixed scale.

When **vType** is set to VT_DECIMAL (0x0000E), the **vData1** and **vData2** fields of CBaseStorageVariant MUST be interpreted as follows:

vData1: The number of digits to the right of the decimal point. MUST be in the range 0 to 28.

vData2: The sign of the numeric value. Set to 0x00 if positive; set to 0x80 if negative.

When **vType** is set to VT_DECIMAL, the format of the **vValue** field is specified in the following diagram.



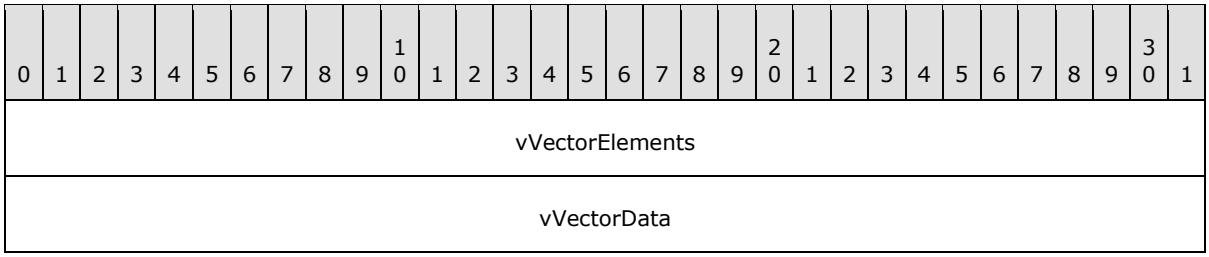
Hi32: The highest 32 bits of the 96-bit integer.

Lo32: The lowest 32 bits of the 96-bit integer.

Mid32: The middle 32 bits of the 96-bit integer.

2.2.1.1.1.2 VT_VECTOR

VT_Vector is used to pass one-dimensional arrays.



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

vVectorData: 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.

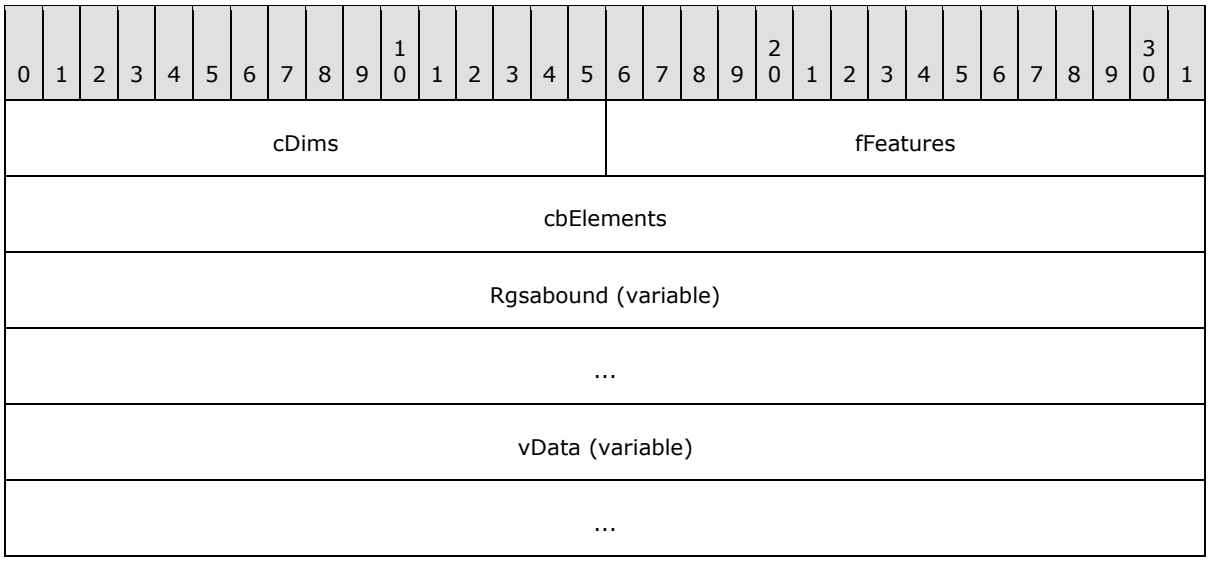
For variable-length data types, vVectorData contains a sequence of consecutively marshaled simple types where the type is indicated by **vType** with the 0x1000 bit cleared. This includes a special case indicated by **vType** VT_ARRAY | VT_VARIANT (that is, 0x100C).

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

For a **vType** set to VT_ARRAY | VT_VARIANT, the type for items in this sequence is CBaseStorageVariant.

2.2.1.1.1.3 SAFEARRAY

SAFEARRAY is used to pass multidimensional arrays. The structure contains array size information as well as the data in the array.



cDims: Unsigned 16-bit integer, indicating the number of dimensions of the multidimensional array.

fFeatures: A 16-bit bitfield. The values represent features, defined by upper-layer applications, and MUST be ignored.

cbElements: A 32-bit unsigned integer specifying the size of each element of the array.

Rgsabound: An array that contains one SAFEARRAYBOUND structure per dimension in the SAFEARRAY. This array has left-most dimension first, and right-most dimension last.

vData: A vector of marshaled items of a particular type, indicated by the **vType** of the containing CBaseStorageVariant, with the bit 0x2000 cleared.

vData is marshaled similarly to VT_Vector, as specified in section 2.2.1.1.1.2, with the difference that the number of items is not stored in front of the vector. Rather, the number of items is calculated by multiplying the **cElements** value with all safe array bounds given in the **Rgsabound** field. Elements are stored in a vector in order of dimensions, iterating beginning with the right-most dimension.

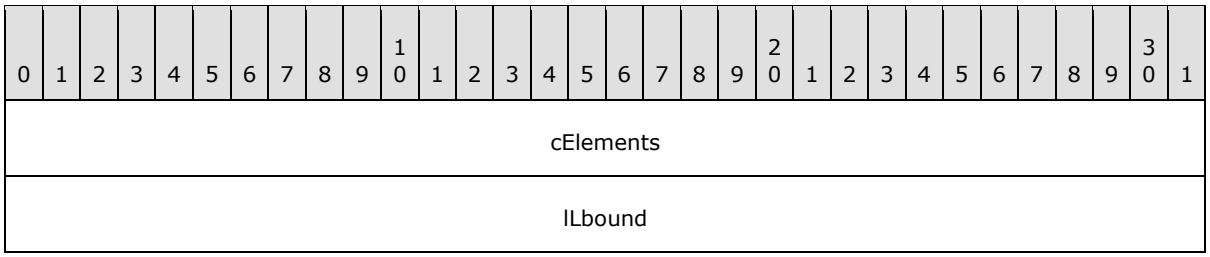
The following diagram visually represents a sample two-dimensional array. The first dimension has **cElements** equal to 4 (represented horizontally) and **lLbound** equal to 0, and the second dimension has **cElements** equal to 2 (represented vertically) and **lLbound** equal to 0.

cElement 0	cElement 1	cElement 2	cElement 3
0x00000001	0x00000002	0x00000003	0x00000005
0x00000007	0x00000011	0x00000013	0x00000017

Using the previous diagram, vData will contain the following sequence: 0x00000001, 0x00000007, 0x00000002, 0x00000011, 0x00000003, 0x00000013, 0x00000005, 0x00000017 (iterating through the rightmost dimension first, and then incrementing the next dimension). The preceding **Rgsabound** (which records **cElements** and **lLbound**) would be: 0x00000004, 0x00000000, 0x00000002, and 0x00000000.

2.2.1.1.1.4 SAFEARRAYBOUND

The SAFEARRAYBOUND structure represents the bounds of one dimension of a SAFEARRAY or SAFEARRAY2. Its format is the following.

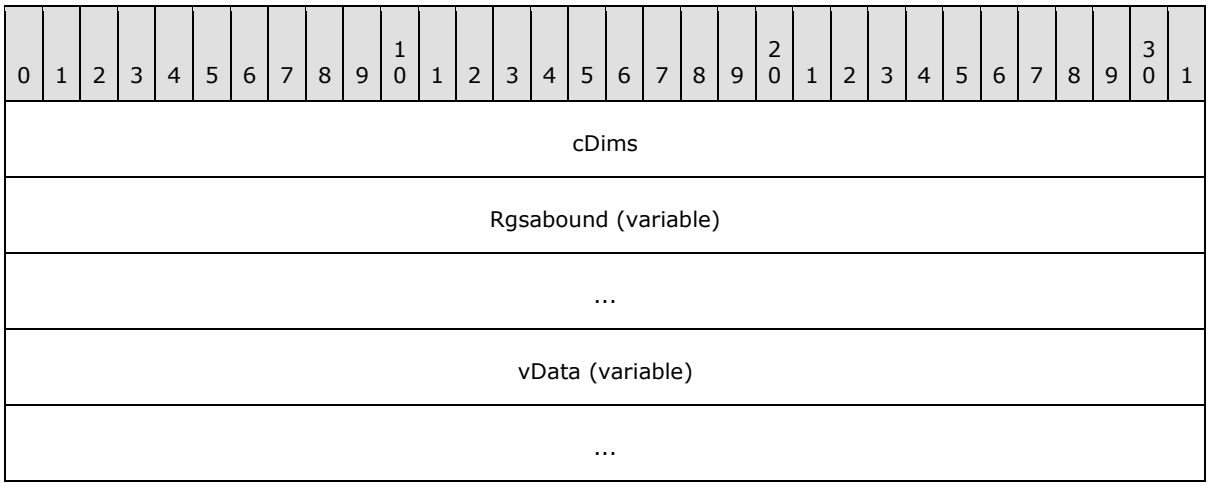


cElements: A 32-bit unsigned integer, specifying the number of elements in the dimension.

lLbound: A 32-bit unsigned integer, specifying the lower bound of the dimension.

2.2.1.1.1.5 SAFEARRAY2

SAFEARRAY2 is used to pass multidimensional arrays in SERIALIZEDPROPERTYVALUE. The structure contains boundary information as well as the data.



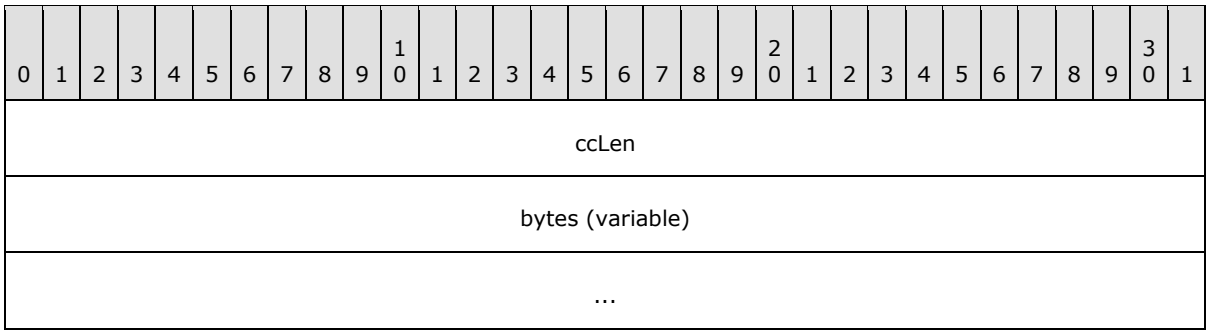
cDims: Unsigned 32-bit integer, indicating the number of dimensions of the SAFEARRAY2.

Rgsabound: An array that contains one SAFEARRAYBOUND structure per dimension in the SAFEARRAY2. This array has the left-most dimension first and the right-most dimension last.

vData: A vector of marshaled items of a particular type, indicated by the **dwType** of the containing SERIALIZEDPROPERTYVALUE, with bit 0x2000 cleared. The format of vData is the same as that specified for the vData field of SAFEARRAY.

2.2.1.1.1.6 VT_COMPRESSED_LPWSTR

The VT_COMPRESSED_LPWSTR structure contains a compressed version of a null-terminated Unicode string, as specified in [UNICODE].



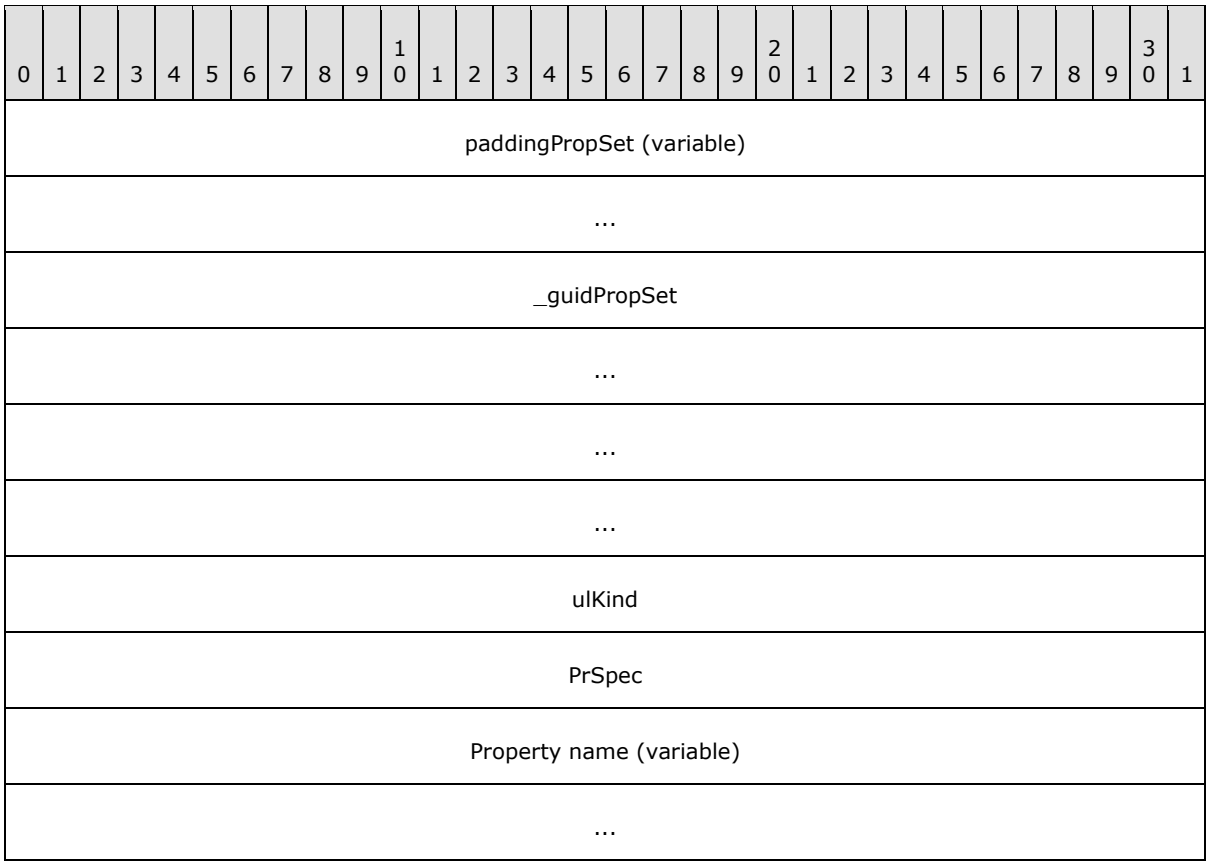
ccLen: A 32-bit unsigned integer, indicating the number of characters in the compressed Unicode string, excluding the terminating NULL character. A value of 0x00000000 indicates that no such string is present.

bytes: A sequence of bytes, each representing the lower byte of a two-byte Unicode character, where the higher byte of the character is always set to zero. Note that only the first 255 Unicode characters can be represented with this encoding scheme. This field MUST be absent if **ccLen** is set to 0x00000000.

2.2.1.2 CFullPropSpec

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

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



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

_guidPropSet: The GUID of the property set to which the property belongs.

ulKind: A 32-bit unsigned integer. MUST be one of the following values that indicates the contents of **PrSpec**.

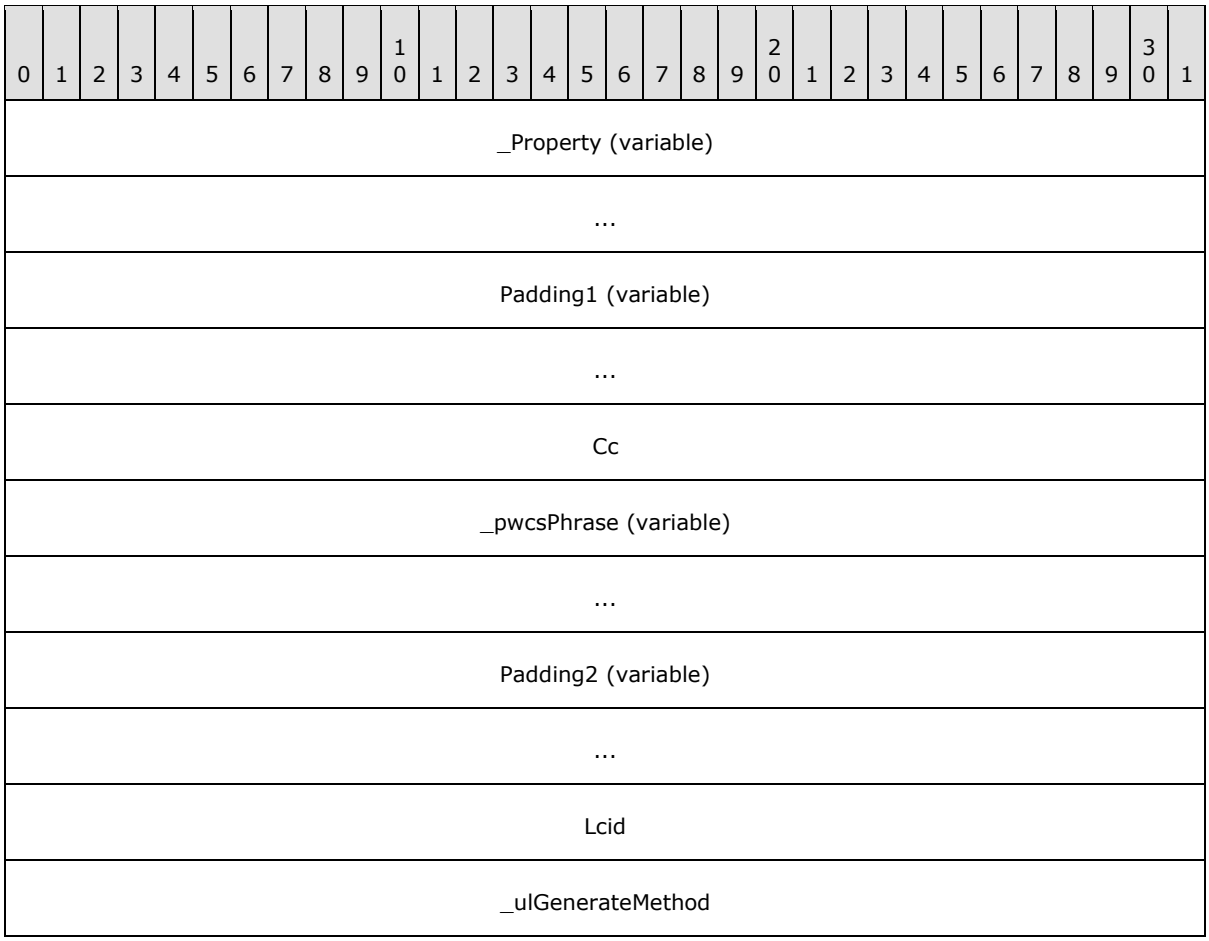
Value	Meaning
PRSPEC_LPWSTR 0x00000000	The **PrSpec** field specifies the number of non-NULL characters in the **Property name** field.
PRSPEC_PROPID 0x00000001	The **PrSpec** field specifies the property ID (PROPID).

PrSpec: A 32-bit unsigned integer with a meaning as indicated by the **ulKind** field.

Property name: If **ulKind** is set to PRSPEC_PROPID, this field MUST NOT be present. If **ulKind** is set to PRSPEC_LPWSTR, this field MUST contain a case-insensitive array of **PrSpec** non-null Unicode characters that contains the name of the property.

2.2.1.3 CContentRestriction

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



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

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

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

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

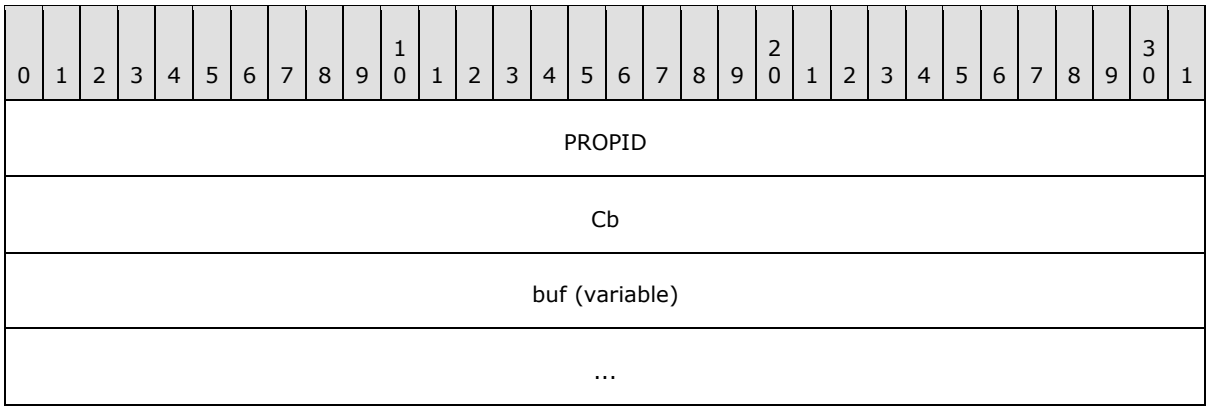
Lcid: A 32-bit unsigned integer, indicating the locale of **_pwcsPhrase**, as specified in [MS-LCID].

_ulGenerateMethod: A 32-bit unsigned integer, specifying the method to use when generating alternate word forms.

Value	Meaning
GENERATE_METHOD_EXACT 0x00000000	Exact match. Each word in the phrase must match exactly in the inverted index.
GENERATE_METHOD_PREFIX 0x00000001	Prefix match. Each word in the phrase is considered a match if the word is a prefix of an indexed string. For example, if the word "barking" is indexed, then "bar" would match when performing a prefix match.
GENERATE_METHOD_INFLECT 0x00000002	Matches inflections of a word. An inflection 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, inflections of the verb swim in English include swim, swims, swimming, and swam.

2.2.1.4 CKey

The CKey structure contains a property value.



PROPID: A 32-bit unsigned integer, representing the property ID as discussed in section 1.8.1. The following are well-known values.

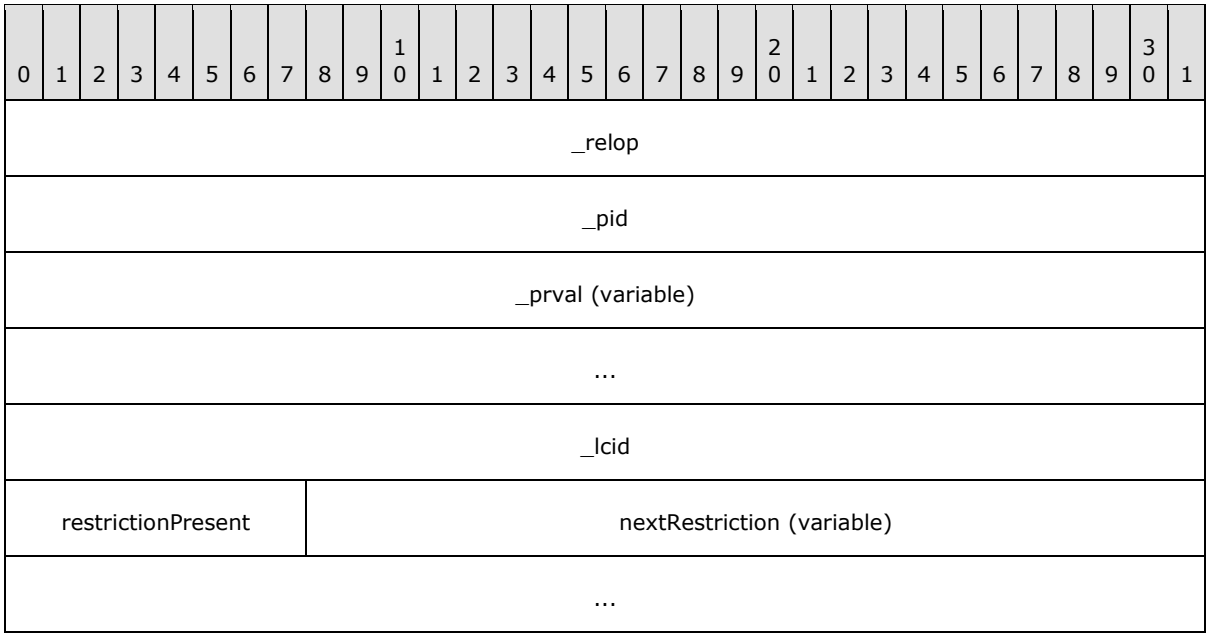
Value	Meaning
0xFFFFFFFF	Represents an invalid property ID and MUST NOT be used.
0xFFFFFFFE	Represents an invalid property ID and MUST NOT be used.
0x00000000	Represents "any" property ID.

Cb: A 32-bit unsigned integer containing the length, in bytes, of **buf**.

buf: A sequence of bytes representing the value of the property.

2.2.1.5 CInternalPropertyRestriction

The CInternalPropertyRestriction structure contains a property value to match with an operation.



_relop: A 32-bit integer specifying the relation to perform on the property. **_relop** MUST be one of the following values:

Value	Meaning
PRLT 0x00000000	A less-than comparison.
PRLE 0x00000001	A less-than or equal-to comparison.
PRGT 0x00000002	A greater-than comparison.
PRGE 0x00000003	A greater-than or equal-to comparison.
PREQ 0x00000004	An equality comparison.
PRNE 0x00000005	A not-equal comparison.
PRRE 0x00000006	A regular expression comparison.
PRAllBits 0x00000007	A bitwise AND that returns the right operand.
PRSomeBits 0x00000008	A bitwise AND that returns a nonzero value.

Value	Meaning
PRAll 0x00000100	The operation is to be performed on a column of a rowset and is only true if the operation is true for all rows.
PRAny 0x00000200	The operation is to be performed on a column of a rowset and is true if the operation is true for any row.

_pid: A 32-bit unsigned integer, representing the property ID (see PROPID in section 2.2.1.4).

_prval: A CBaseStorageVariant that contains the value to relate to the property.

If the **vType** of _prval is VT_BLOB and _Property refers to a property of a string type (VT_LPSTR, VT_LPWSTR, VT_COMPRESSED_LPWSTR, VT_BSTR, or VT_VECTORs or VT_ARRAYs of those base types) then the first byte of the blob SHOULD be the low-order byte of a valid RANGEBOUNDARY ulType value (see section 2.2.1.23) and be followed by a null-terminated Unicode string. The value should be interpreted in the same manner as RANGEBOUNDARY values (see section 2.2.1.23).

_lcid: A 32-bit unsigned integer, indicating the locale of a string, contained in **_prval** value, as specified in [MS-LCID].

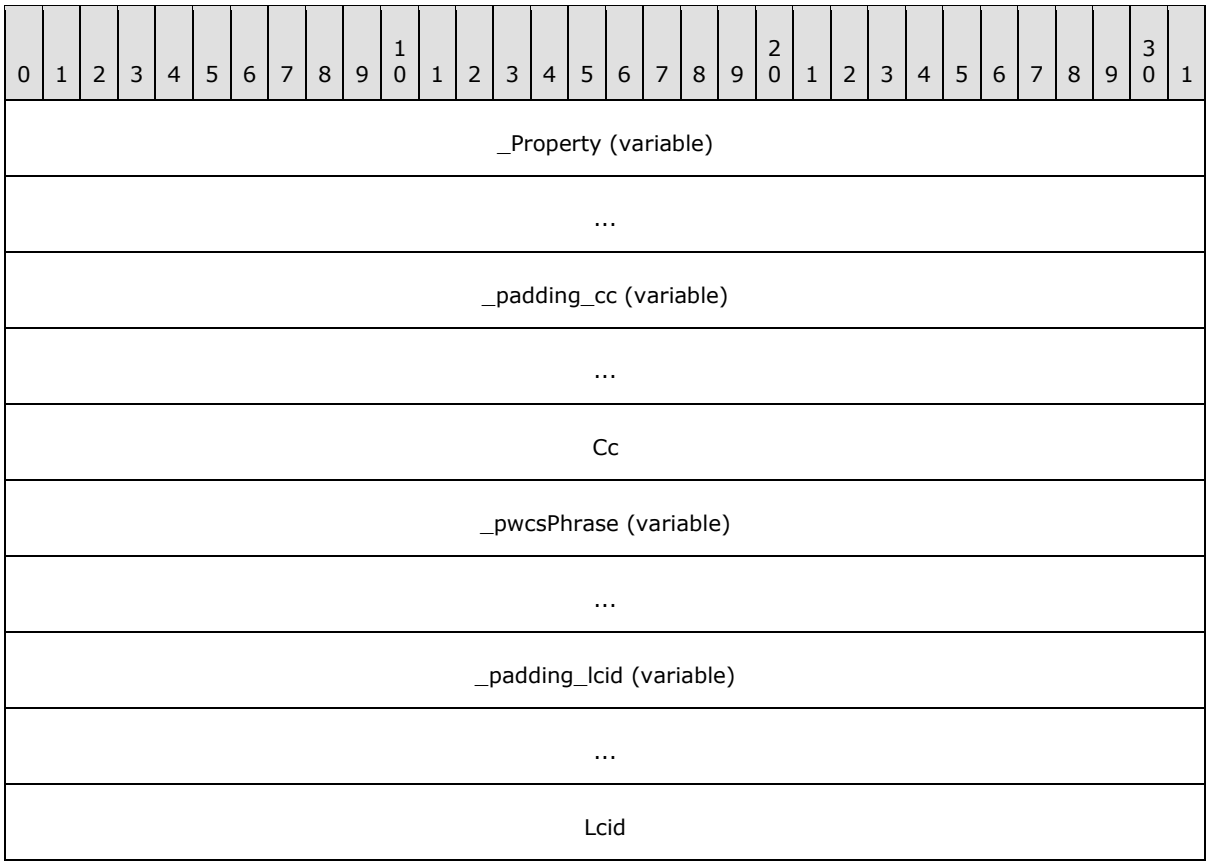
restrictionPresent: A byte value. MUST be set to one of the following values:

Value	Meaning
0x00	**restrictionPresent** indicates that the **nextRestriction** field is not present.
0x01	**restrictionPresent** indicates that the **nextRestriction** field is present.

nextRestriction: A CRestriction structure specifying a further restriction.

2.2.1.6 CNatLanguageRestriction

The CNatLanguageRestriction structure contains a natural language query match for a property. Natural language simply means that the string has no formal meaning. The Windows Search service is free to match on the string any way it wants. It can drop words, add alternate forms or make no changes.



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

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

Cc: A 32-bit unsigned integer. The number of characters in the **_pwcsPhrase** field.

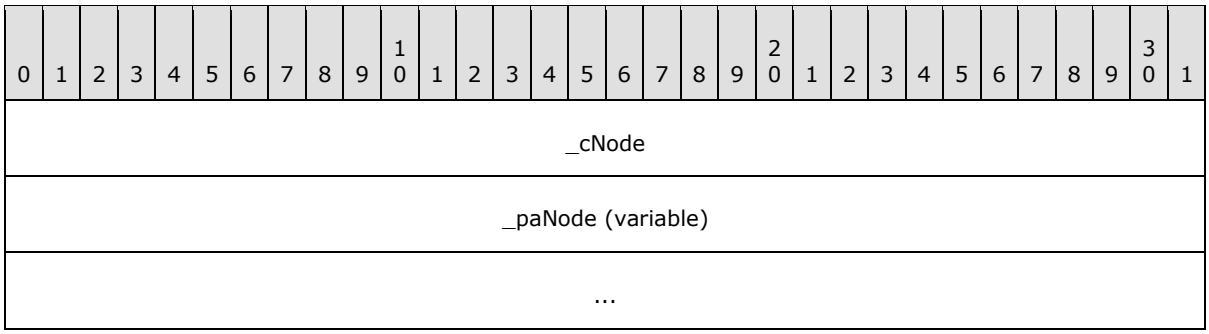
_pwcsPhrase: A non null-terminated Unicode string containing the text to search for within the specific property. MUST NOT be empty. The **Cc** field contains the length of the string.

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

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

2.2.1.7 CNodeRestriction

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

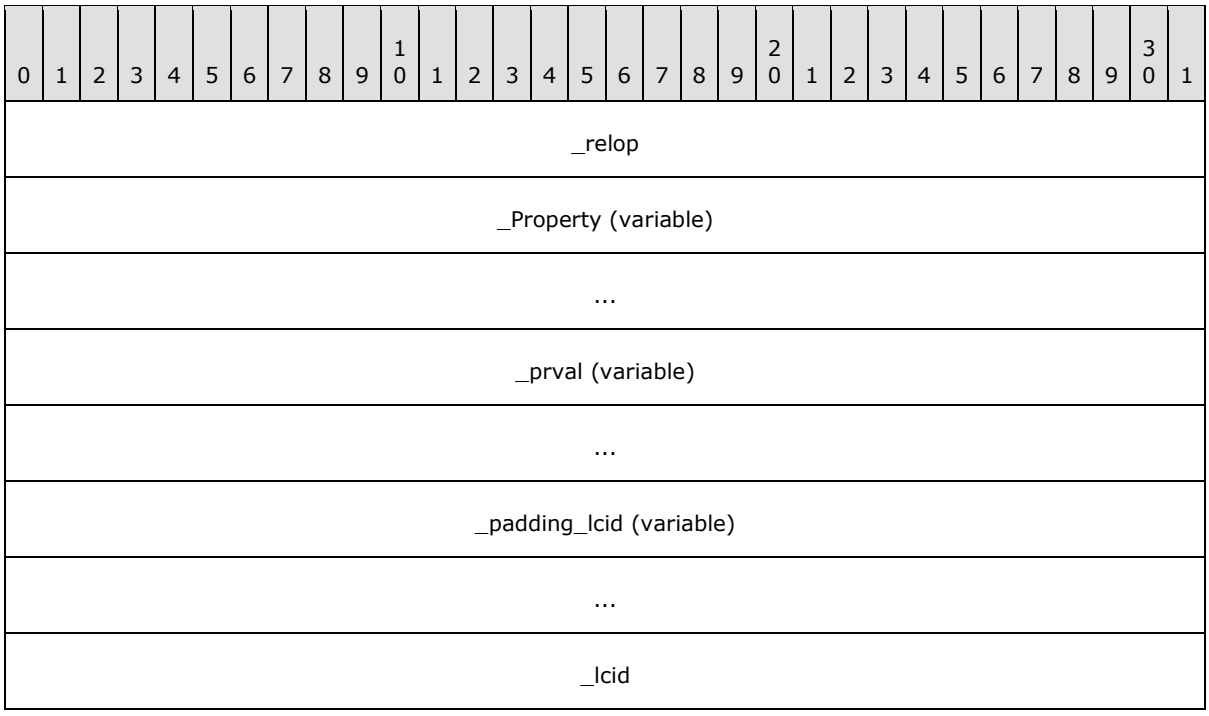


_cNode: A 32-bit unsigned integer specifying the number of CRestriction structures contained in the **_paNode** field.

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

2.2.1.8 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 datatypes of the values must match.



_relop: A 32-bit unsigned integer specifying the relation to perform on the property. **_relop** MUST be one of the following values.

Value	Meaning
PRLT 0x00000000	A less-than comparison.
PRLE 0x00000001	A less-than or equal-to comparison.
PRGT 0x00000002	A greater-than comparison.
PRGE 0x00000003	A greater-than or equal-to comparison.
PREQ 0x00000004	An equality comparison.
PRNE 0x00000005	A not-equal comparison.
PRRE 0x00000006	A regular expression comparison (see below).
PRAllBits 0x00000007	A bitwise AND that returns the value equal to **_prval**.
PRSomeBits 0x00000008	A bitwise AND that returns a nonzero value.

For vector properties, the behavior of the relational operators depends on the result of a logical **OR** using a mask and the relational operator.

If there is no mask, the **restriction** is true if the relational operator holds between each element of a property value and the corresponding element in the **_prval** field. If, in addition, the two vectors have different lengths, then the vector lengths are compared using the relational operator.

If there is a mask, its possible values are:

Value	Meaning
PRAll 0x00000100	The restriction is true if every element in a property value has the relationship with some element in the **_prval** field.
PRAny 0x00000200	The restriction is true if any element in the property value has the relationship with some element in the **_prval** field.

For PRRE relations, regular expressions are expressed with a string that contains special symbols. Any character except an asterisk (*), period (.), question mark (?), or vertical bar (|) matches itself. A regular expression can be enclosed in matching quotes ("…"), and must be enclosed in quotes if it contains a space or closing parenthesis (the ")" character).

The asterisk matches any number of characters. The period matches end of string. The question mark matches any one character. The vertical bar (|) is an escape character, which indicates special behavior for the characters in the table below the ([) character. The following table explains the meanings of special characters in regular expressions.

Character	Meaning
(	An opening parenthesis opens a group. It must be followed by a matching closing parenthesis.
)	A closing parenthesis closes a group. It must be preceded by a matching opening parenthesis.
[	An opening square bracket preceded (escaped) by a vertical pipe character opens a character class. It must be followed by a matching (unescaped) closing square bracket.
{	An opening brace opens a counted match. It must be followed by a matching closing brace.
}	A closing brace closes a counted match. It must be preceded by a matching opening brace.
,	A comma separates OR clauses.
*	An asterisk matches zero or more occurrences of the preceding expression.
?	A question mark matches zero or one occurrence of the preceding expression.
+	A plus sign matches one or more occurrences of the preceding expression.
Other	All other characters match themselves.

The following table describes characters which, when located between square brackets ([ ]), have special meanings.

Character	Meaning
^	A caret matches everything but following classes. (It must be the first character in the string.)
]	A closing square bracket matches another closing square bracket. It may be preceded only by a caret (^); otherwise it closes the class.
-	A hyphen is a range operator. It is preceded and followed by normal characters.
Other	All other characters match themselves (or begin or end a range).

The following table describes the syntax used between braces ({ }).

Character	Meaning
{m}	Matches exactly m occurrences of the preceding expression (0 < m < 256).
{m,}	Matches at least m occurrences of the preceding expression (1 < m < 256).
{m, n}	Matches between m and n occurrences of the preceding expression, inclusive (0 < m < 256, 0 < n < 256).

To match the asterisk and question mark, enclose them within brackets. For example, [*]sample matches "*sample".

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

_prval: A CBaseStorageVariant structure containing the value to relate to the property.

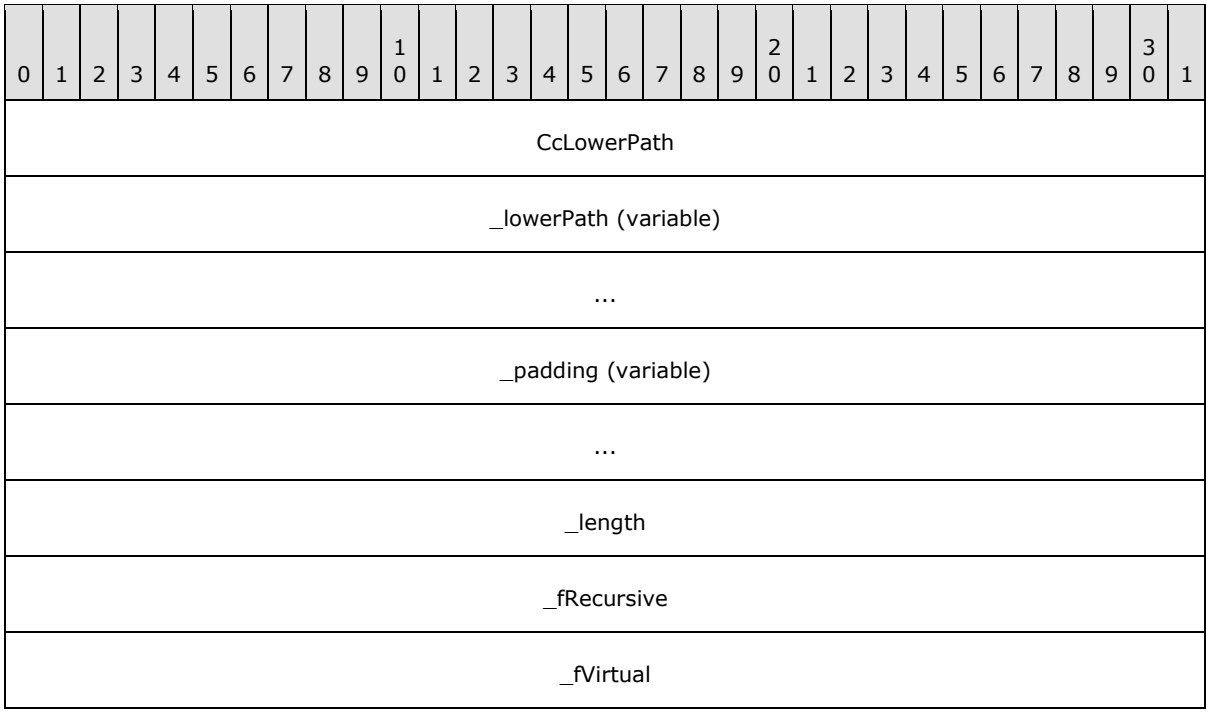
If the **vType** of _prval is VT_BLOB, and _Property refers to a property of a string type (VT_LPSTR, VT_LPWSTR, VT_COMPRESSED_LPWSTR, VT_BSTR, or VT_VECTORs or VT_ARRAYs of those base types), then the first byte of the blob SHOULD be the low-order byte of a valid RANGEBOUNDARY ulType value (see section 2.2.1.23) and be followed by a null-terminated Unicode string. The value should be interpreted in the same manner as RANGEBOUNDARY values (see section 2.2.1.23).

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

_lcid: A 32-bit unsigned integer representing locale for the string contained in **_prval**, as specified in [MS-LCID].

2.2.1.9 CScopeRestriction

The CScopeRestriction structure restricts the files to be returned to those that have a path that matches the restriction.



CcLowerPath: A 32-bit unsigned integer containing the number of Unicode characters in the **_lowerPath** field.

_lowerPath: A non null-terminated Unicode string representing the **path** to which the query should be restricted. The **CcLowerPath** field contains the length of the string.

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

_length: A 32-bit unsigned integer containing the length of **_lowerPath** in Unicode characters. This MUST be the same value as **CcLowerPath**.

_fRecursive: A 32-bit unsigned integer. MUST be set to one of the following values:

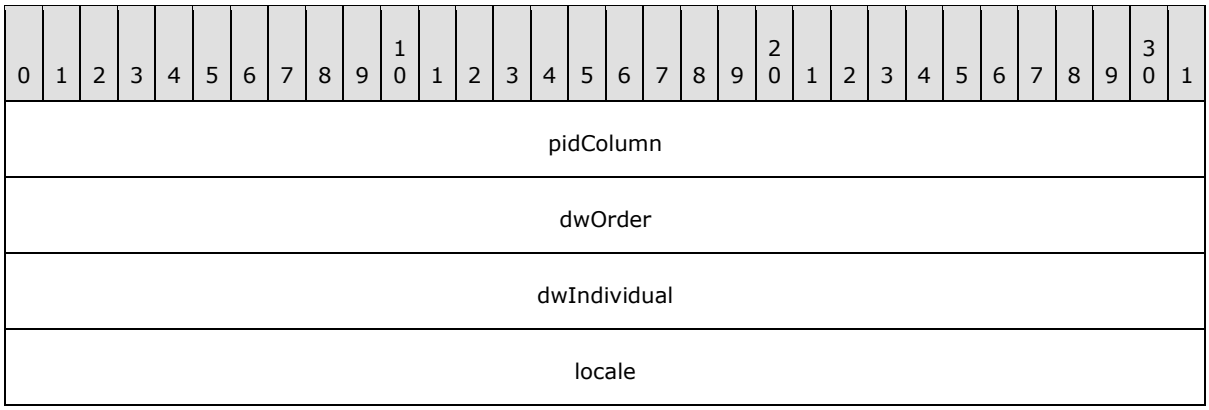
Value	Meaning
0x00000000	The server is not to examine any subdirectories.
0x00000001	The server is to recursively examine all subdirectories of the path.

_fVirtual: A 32-bit unsigned integer. MUST be set to one of the following values:

Value	Meaning
0x00000000	**_lowerPath** is a file system path.
0x00000001	**_lowerPath** is a virtual path (the Uniform Resource Locator associated with a physical directory on the file system) for a Web site.

2.2.1.10 CSort

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



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

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

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

Value	Meaning
0x00000001	column specified.

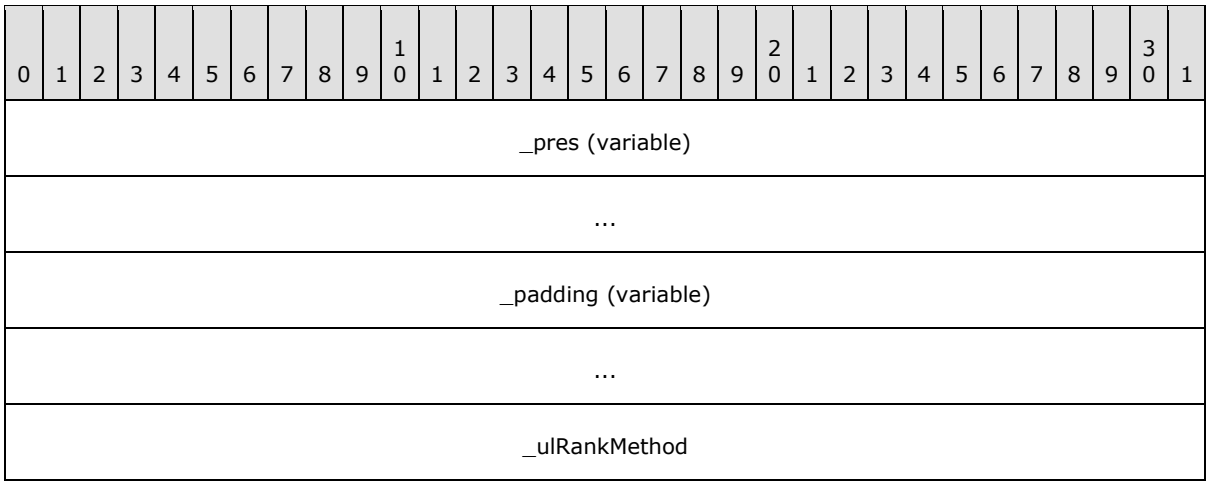
dwIndividual: A 32-bit unsigned integer. **dwIndividual** specifies how to treat properties of type VT_VECTOR with regard to sorting, and MUST be one of the following values.

Value	Meaning
QUERY_SORTALL 0x00000000	The complete property is used for sorting, resulting in a single row for each result.
QUERY_SORTINDIVIDUAL 0x00000001	Each element of the VT_VECTOR is used for sorting independently, possibly resulting in multiple rows for a single result.

locale: A 32-bit unsigned integer indicating the locale (as specified in [MS-LCID]) of the column. The **locale** determines the sorting rules to use when sorting textual values. A Windows Search service SHOULD use the appropriate operating system facilities to do this.

2.2.1.11 CVectorRestriction

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



_pres: A CNodeRestriction command tree upon which a ranked OR operation is to be performed.

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

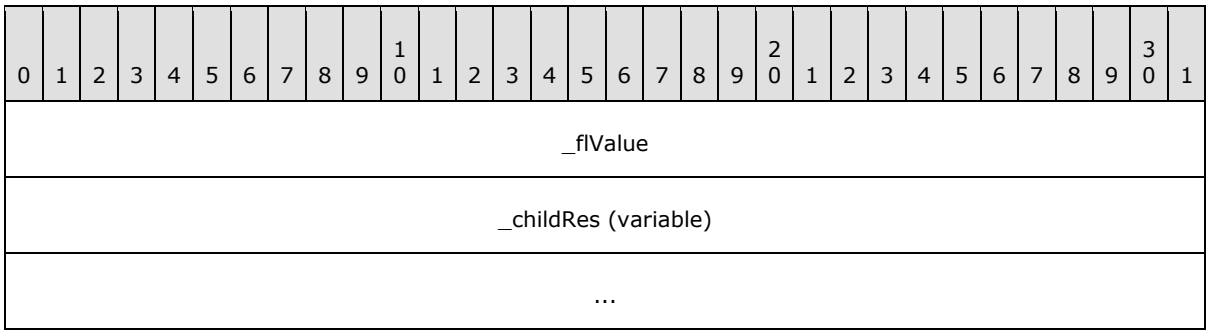
_ulRankMethod: A 32-bit unsigned integer specifying a ranking algorithm. MUST be set to one of the following values.

Value	Meaning
VECTOR_RANK_MIN	Use the minimum algorithm, as specified in [SALTON].

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

2.2.1.12 CCoercionRestriction

The CCoercionRestriction structure contains the modifier and rank coercion operation.



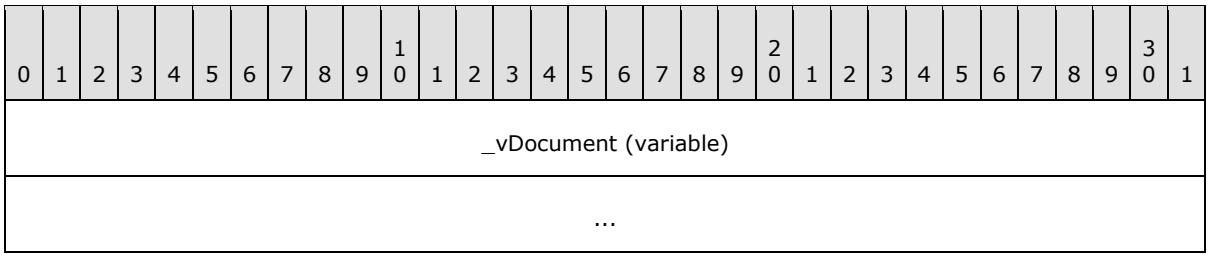
_flValue: An IEEE 32-bit floating point number [IEEE754] representing the coercion value upon which the rank coercion operation happens. Note that the coercion operation is determined by the containing CRestriction structure. The following coercion operations are supported.

Operation	Meaning
ADD	The rank value returned is the sum of the original rank value and the coercion value.
MULTIPLY	The rank value returned is the product of the original rank value and the coercion value and MUST be in the range 0.001 to 1.0.
ABSOLUTE	The rank value returned is the value specified in the coercion value and MUST be in the range 0 to 1000.

_childRes: CRestriction structure that specifies a command tree. The returned rank value for results of a child restriction will be coerced as specified by the containing CRestriction structure.

2.2.1.13 CRelDocRestriction

A CRelDocRestriction structure contains a relevant document ID.



_vDocument: A CBaseStorageVariant structure that specifies a relevant document for a relevance feedback query. The **vType** field of the **_vDocument** structure MUST be set to either VT_I4 or to VT_BSTR, specifying a URL string.

If **vType** is set to VT_I4, then **vValue** MUST be set to the document ID.

If **vType** is set to VT_BSTR, the URL string MUST be formatted by concatenating a constant string containing the docId protocol, the application and the catalog name with the document ID in decimal representation, as in the following example:

```
docid:Windows.SystemIndex.153
```

In the example, "Windows" is the application name, "SystemIndex" is the catalog name and "153" is the document ID for the document in decimal notation.

2.2.1.14 CProbRestriction

A CProbRestriction structure contains parameters for probabilistic ranking.

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)																															
...																															
_flK1																															
_flK2																															
_flK3																															
_flB																															
_cFeedbackDoc																															
_ProbQueryPid																															

_Property: A CFullPropSpec structure, indicating which property to use for probabilistic ranking or the columns' group full property specification (which corresponds to **_groupPid** field in the CColumnGroup structure).

_flK1: An IEEE 32-bit floating point number [IEEE754] that indicates parameter k1 in formula [1], specified below.

_flK2: An IEEE 32-bit floating point number.

Note MUST be set to 0.0.

_flK3: An IEEE 32-bit floating point number that indicates parameter k3 in formula [1].

_flB: An IEEE 32-bit floating point number that indicates parameter b in formula [1] below.

_cFeedbackDoc: A 32-bit unsigned integer specifying the count of relevant documents.

_ProbQueryPid: A 32-bit unsigned integer.

Note Reserved. MUST be set to 0x00000000.

Formula [1] for probabilistic ranking is the following sum for each query term:

$$\sum \frac{wtf(k_1+1)}{k_1\left((1-b)+b\frac{dl}{avdl}\right)+wtf} \times \log\left(\frac{N-n+0.5}{n+0.5}\right) \times \frac{qtf}{k_3+qtf}$$

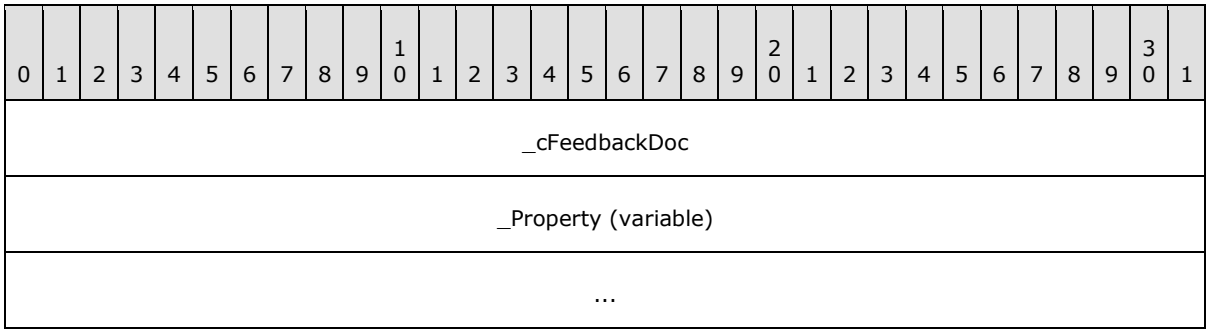
Figure 1: Linear equation for probabilistic ranking restrictions in search request

Where:

- wtf (weighted term frequency) is the sum of term frequencies of a given term multiplied by weights across all properties.
- dl is the document length, in terms of number of words.
- avdl is the average document length in the corpus, in terms of number of words.
- N is the total number of documents in the corpus.
- n is the number of documents in the corpus that have the given query term.
- qtf is the number of documents containing the given query term; the sum is across all query terms.
- k1, k3, and b are parameters, specified in the CProbRestriction structure.

2.2.1.15 CFeedbackRestriction

The CFeedbackRestriction structure contains the number of relevant documents and a property specification for a relevance feedback query.

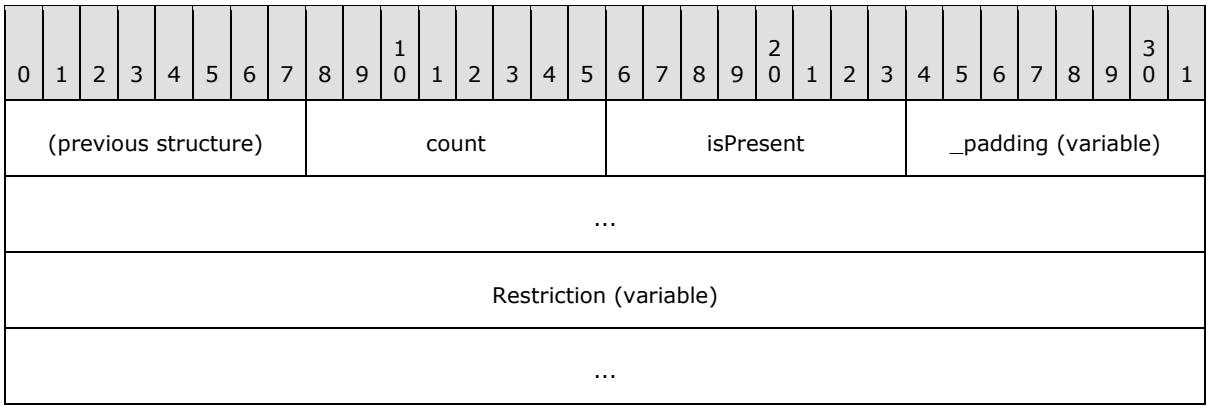


_cFeedbackDoc: A 32-bit unsigned integer specifying the count of relevant documents.

_Property: A CFullPropSpec structure, specifying a property.

2.2.1.16 CRestrictionArray

The CRestrictionArray structure contains an array of restriction nodes. The first two fields (**count** and **isPresent**) are not padded and will start where the previous structure in the message ended (as indicated by the "previous structure" entry in the diagram below). The 1-byte length of "previous structure" is arbitrary and is not meant to suggest **count** will begin on any particular boundary. However, the **Restriction** field MUST be aligned to begin at a multiple of 4 bytes from the beginning of the message, and hence the format is depicted as follows.



count: An 8-bit unsigned integer specifying the number of CRestrictionWithFlag records contained in the **RestrictionWithFlag** field.

Note This field MUST be set to 0x01.

isPresent: An 8-bit unsigned integer. MUST be set to one of the following values:

Value	Meaning
0x00	The **_padding** and **Restriction** fields are omitted.
0x01	The **_padding** and **Restriction** fields are present.

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

Note This field MUST be omitted if the value of **isPresent** is set to 0x00.

Restriction: A CRestriction structure, specifying a node of a query command tree.

Note Restriction MUST be omitted if the value of **isPresent** is set to 0x00.

2.2.1.17 CRestriction

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

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
_ulType																															
Weight																															
Restriction (variable)																															
...																															

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

Value	Meaning
RTNone 0x00000000	The node represents a **noise word** in a vector query.
RTAnd 0x00000001	The node contains a CNodeRestriction upon which a logical AND operation is to be performed.
RTOr 0x00000002	The node contains a CNodeRestriction upon which a logical OR operation is to be performed.
RTNot 0x00000003	The node contains a CRestriction upon which a NOT 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 is undefined. The restriction requires the words or phrases found in the CContentRestriction structures to be within a Windows Search service defined range in order to be a match. The Windows Search service can also compute a rank based on how far apart the words or phrases are.
RTVector 0x00000007	The node contains a CVectorRestriction.
RTNatLanguage 0x00000008	The node contains a CNatLanguageRestriction.
RTScope 0x00000009	The node contains a CScopeRestriction.
RTInternalProp 0x00FFFFFA	The node contains a CInternalPropertyRestriction.
RTPhrase	The node contains a CNodeRestriction upon which a phrase match is to be

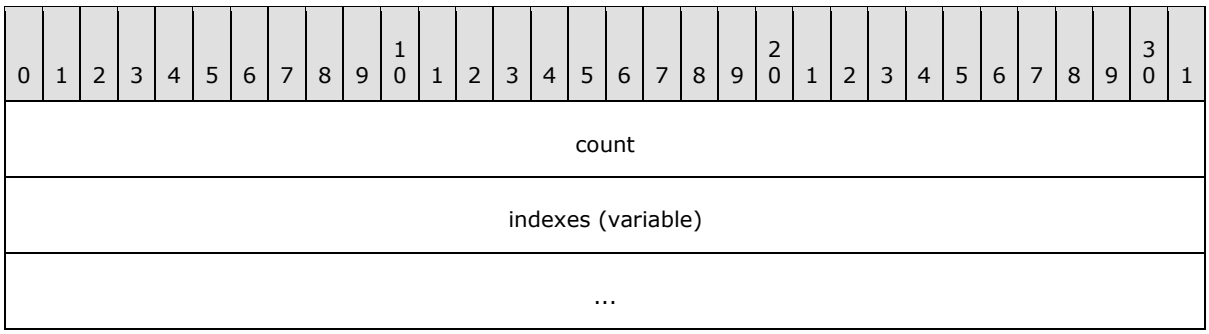
Value	Meaning
0x00FFFFFD	performed.
RTCoerce_Add 0x0000000A	The node contains a CCoercionRestriction structure with operation ADD, as specified in section 2.2.1.12.
RTCoerce_Multiply 0x0000000B	The node contains a CCoercionRestriction with structure operation MULTIPLY, as specified in section 2.2.1.12.
RTCoerce_Absolute 0x0000000C	The node contains a CCoercionRestriction structure with operation ABSOLUTE, as specified in section 2.2.1.12.
RTProb 0x0000000D	The node contains a CProbRestriction structure.
RTFeedback 0x0000000E	The node contains a CFeedbackRestriction structure.
RTReldoc 0x0000000F	The node contains a CRelDocRestriction structure.

Weight: 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: The restriction type for the command tree node. The syntax MUST be as indicated by the **_ulType** field.

2.2.1.18 CColumnSet

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

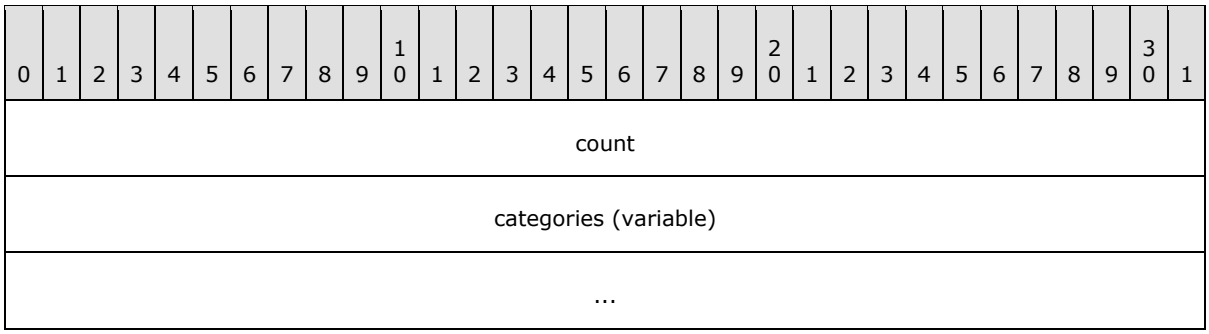


count: A 32-bit unsigned integer specifying the number of elements in the **indexes** array.

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

2.2.1.19 CCategorizationSet

The CCategorizationSet structure contains information on how the grouping is done at each level in a hierarchical result set.

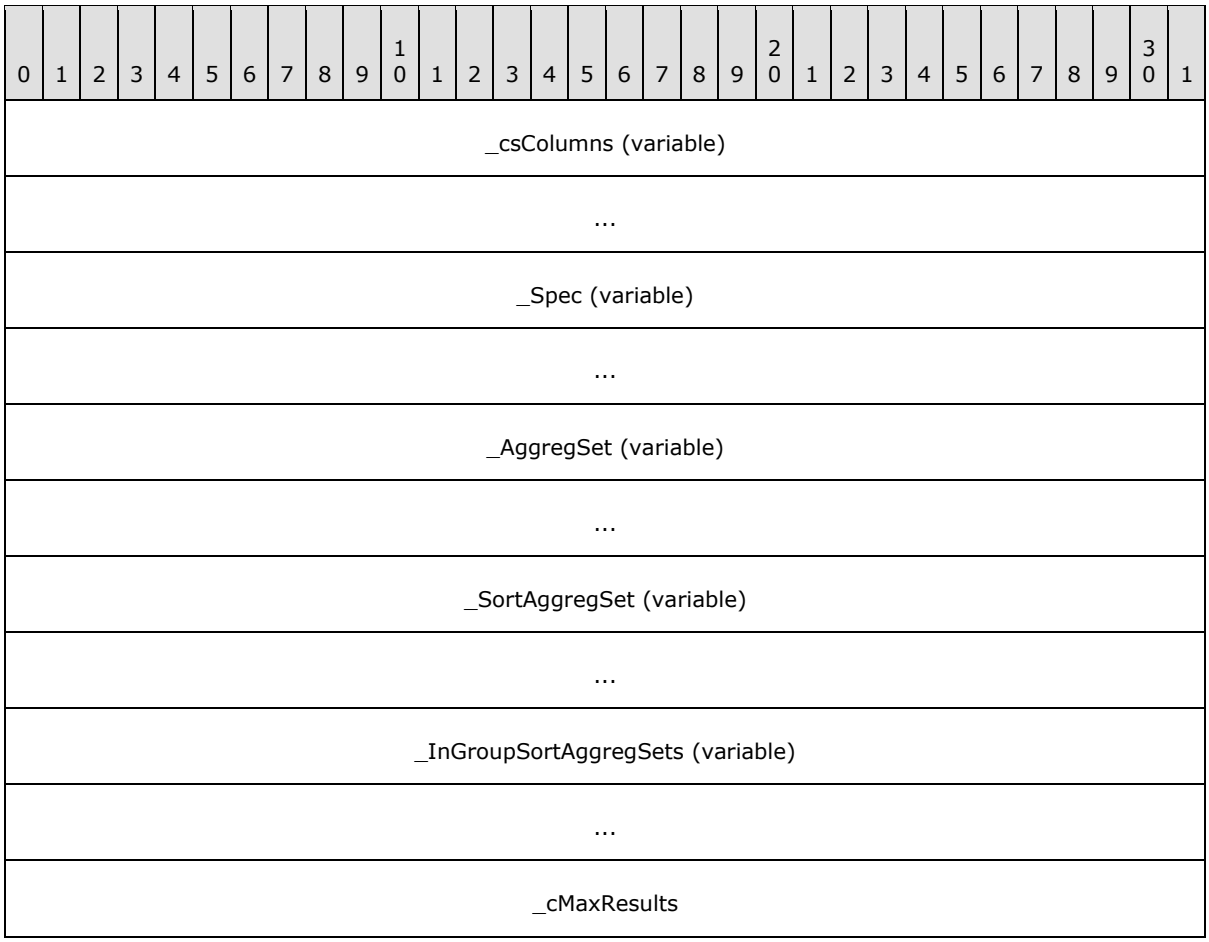


count: A 32-bit unsigned integer containing the number of elements in the **categories** array.

categories: Array of CCategorizationSpec structures specifying the grouping for each level in a hierarchical query. The first structure specifies the top-level.

2.2.1.20 CCategorizationSpec

The CCategorizationSpec structure specifies how grouping is done at one level in a hierarchical query.



_csColumns: A CColumnSet structure indicating the columns to return at that level in a hierarchical result set.

_Spec: A CCategSpec structure specifying the type of categorization and the column for the group.

_AggregSet: A CAggregSet structure specifying aggregate information for the group.

_SortAggregSet: A CSortAggregSet structure specifying default sorting for the group.

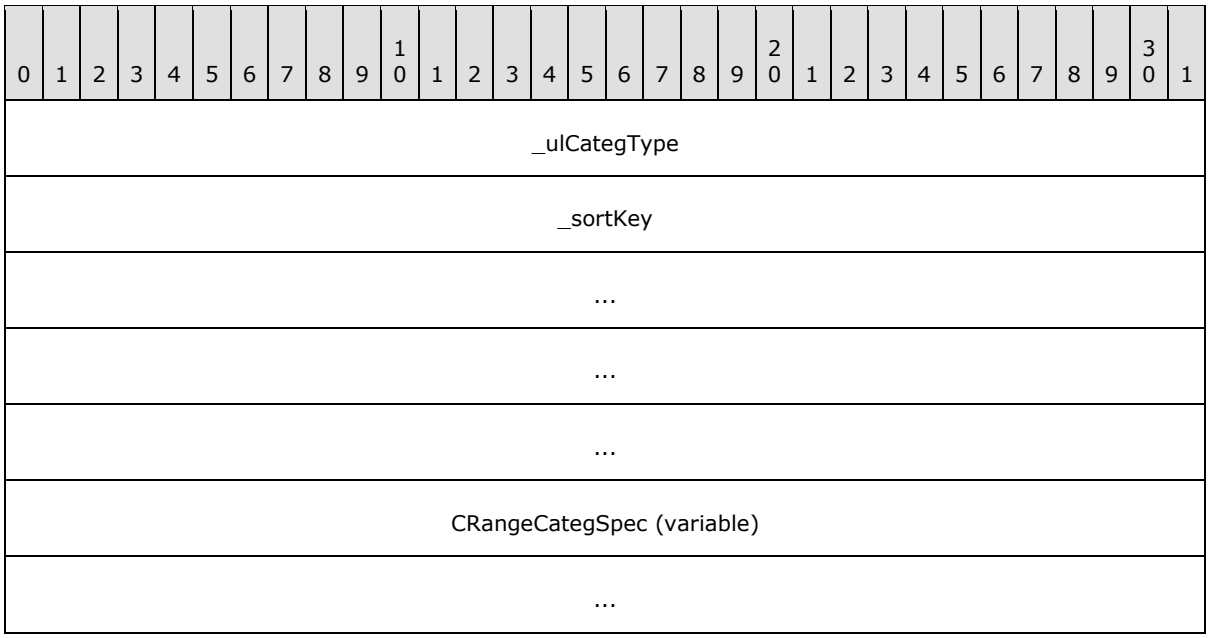
_InGroupSortAggregSets: A CInGroupSortAggregSets structure specifying sorting for the group with regard to the parent group's range boundaries.

_cMaxResults: A 32-bit unsigned integer. Reserved.

Note MUST be set to 0x00000000.

2.2.1.21 CCategSpec

The CCategSpec structure contains information about which grouping to perform over query results.



_ulCategType: MUST be set to one of the following values that indicates the type of grouping to perform.

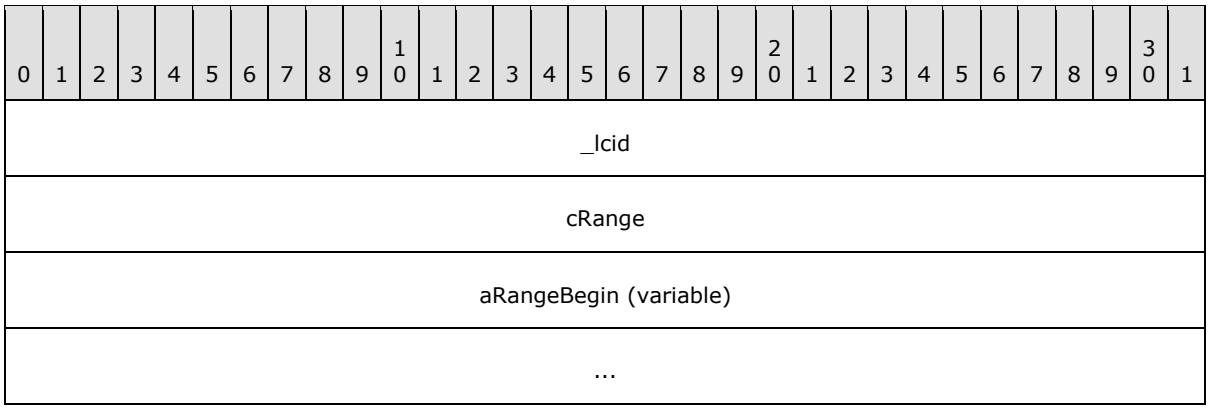
Value	Meaning
CATEGORIZE_UNIQUE 0x00000000	Unique categorization. Each unique value forms a category.
CATEGORIZE_RANGE 0x00000003	Range categorization. Ranges are explicitly specified in CRangeCategSpec.

_sortKey: A CSort structure, specifying the sort order for the group.

CRangeCategSpec: A CRangeCategSpec structure specifying the range values. This field MUST be omitted if **_ulCategType** is set to CATEGORIZE_UNIQUE; otherwise it MUST be present.

2.2.1.22 CRangeCategSpec

The CRangeCategSpec structure contains information about ranges for grouping into range-specified buckets.



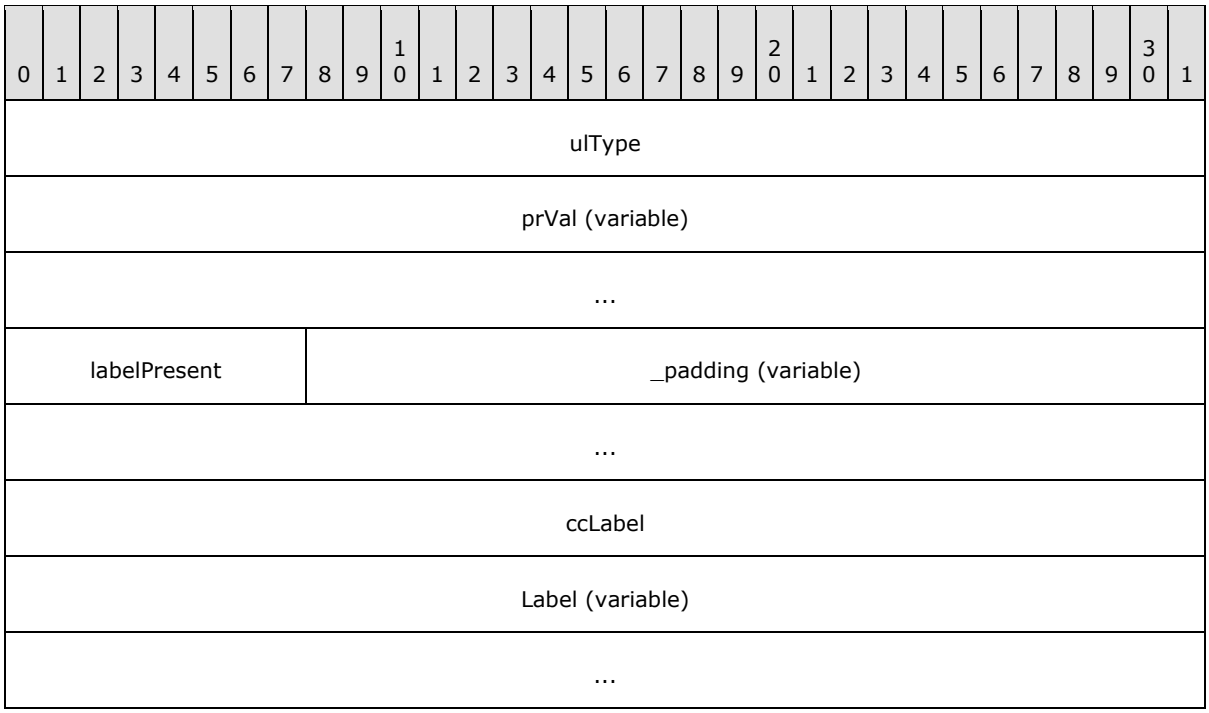
_lcid: A 32-bit unsigned integer. Reserved. The content of this field MAY be arbitrary and MUST be ignored by the receiver.

cRange: A 32-bit unsigned integer, indicating the number of RANGEBOUNDARY structures in **aRangeBegin**.

aRangeBegin: An array of RANGEBOUNDARY structures, specifying a set of ranges for which grouping is performed. Note that the first range is from minimum value to the boundary, represented by the first RANGEBOUNDARY structure. The next range is from where the first boundary cut off to the boundary represented by the second RANGEBOUNDARY structure, and so on. The last range includes all the items greater than the last RANGEBOUNDARY structure to the maximum value. There will be a total of **cRange** + 1 ranges. Values with **vType** set to VT_NULL and VT_EMPTY are always in the last group, regardless of sort order.

2.2.1.23 RANGEBOUNDARY

The RANGEBOUNDARY structure contains a single range.



ulType: A 32-bit unsigned integer that indicates which type of boundary is represented by this structure.

Note MUST be set to one of the following values.

Value	Meaning
DBRANGEBOUNDTTYPE_BEFORE 0x00000000	MUST only be used for Unicode string values in **prVal**. Items with a value less than the Unicode string immediately preceding **prVal** lexicographically are included in the range.
DBRANGEBOUNDTTYPE_EXACT 0x00000001	Items with a value less than **prVal** are included in the range.
DBRANGEBOUNDTTYPE_AFTER 0x00000002	MUST only be used for Unicode string values in **prVal**. Items with the value less than the Unicode string immediately after **prVal** lexicographically are included in the range.

For example, two RANGEBOUNDARY structures of DBRANGEBOUND_EXACT with a **prVal** of "a" and DBRANGEBOUND_AFTER with a **prVal** of "z" could be used to partition the full Unicode range into three buckets:

```
1. x < "a"
2. "a" <= x <= "z"
3. x > "z"
```

prVal: A CBaseStorageVariant structure.

Note Indicates the value for the range boundary. If **ulType** is set to DBRANGEBOUNDTTYPE_BEFORE or DBRANGEBOUNDTTYPE_AFTER, then the **vType** field of prVal MUST be set to a string type (VT_BSTR, VT_LPWSTR, or VT_COMPRESSED_LPWSTR).

labelPresent: An 8-bit unsigned integer. MUST be set to one of the following values:

Value	Meaning
0x00	The **_padding**, **ccLabel**, and **Label** fields are omitted.
0x01	The **_padding**, **ccLabel**, and **Label** fields are present.

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

Note This field MUST be omitted if **isPresent** is set to 0x00.

ccLabel: A 32-bit unsigned integer representing the number of characters in the **Label** field.

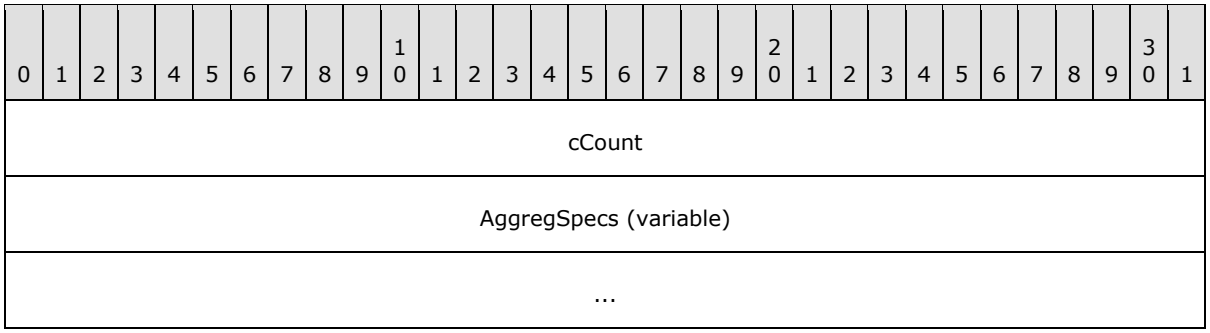
Note ccLabel MUST be omitted if **isPresent** is set to 0x00; otherwise, it MUST be greater than zero.

Label: A non null-terminated Unicode string representing the label for this range. The **ccLabel** field contains the length of the string.

Note Label MUST be omitted if **isPresent** is set to 0x00; otherwise, it MUST NOT be empty.

2.2.1.24 CAggregSet

The CAggregSet structure contains information about aggregates.

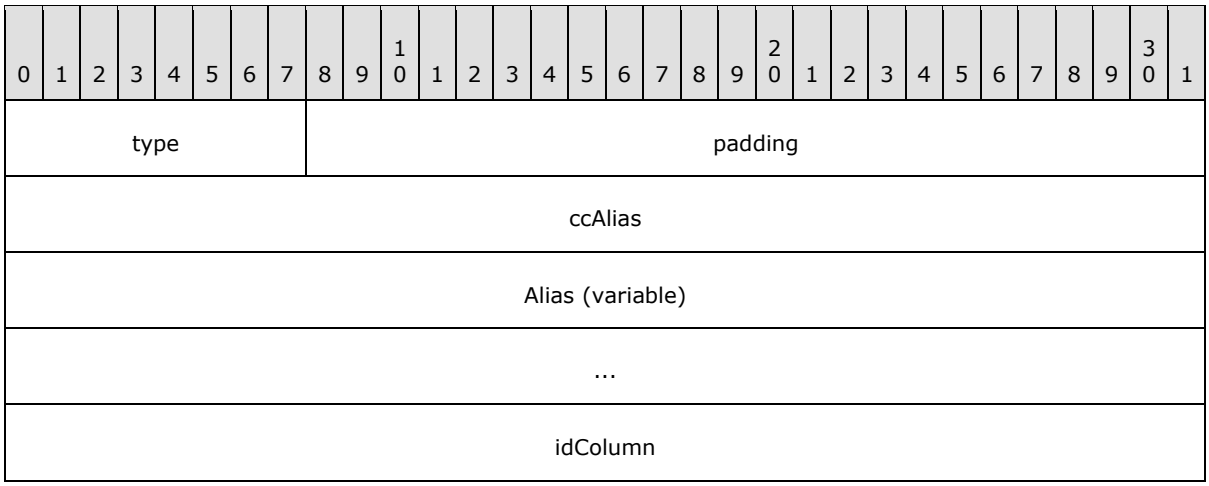


cCount: A 32-bit unsigned integer specifying the number of entries in **AggregSpecs**.

AggregSpecs: An array of CAggregSpec structures, each describing individual aggregation.

2.2.1.25 CAggregSpec

The CAggregSpec structure contains information about an individual aggregate.



type: An 8-bit unsigned integer specifying the type of aggregation used. The type MUST be one of the following values.

Value	Meaning
DBAGGTTYPE_BYNONE 0x00	No aggregation is used.
DBAGGTTYPE_SUM 0x01	Sum of results in group.
DBAGGTTYPE_MAX 0x02	Maximum value of results in group.
DBAGGTTYPE_MIN 0x03	Minimum value of results in group.
DBAGGTTYPE_AVG 0x04	Average value of results in group.
DBAGGTTYPE_COUNT 0x05	Count of rows in group.
DBAGGTTYPE_CHILDCOUNT 0x06	Count of top-level subgroups in group.

padding: This field MUST be 3 bytes in length, and the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

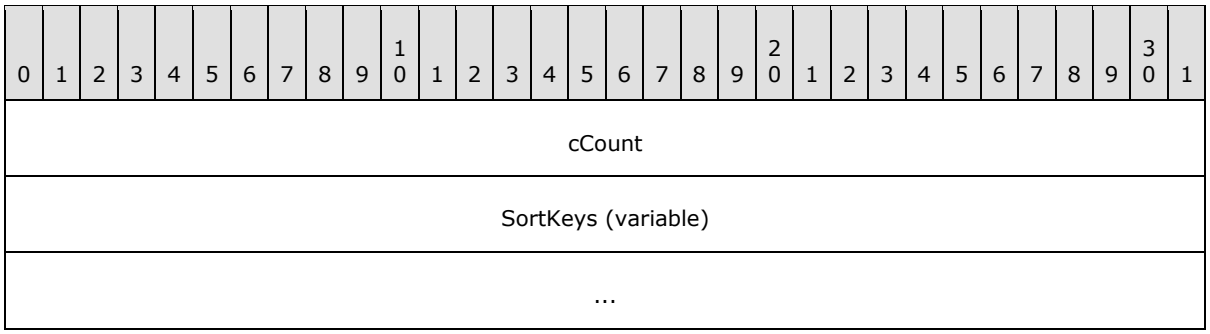
ccAlias: A 32-bit unsigned integer specifying the number of characters in the **Alias** field.

Alias: A non null-terminated Unicode string that represents the alias name for the aggregate. The **ccAlias** field contains the length of the string.

idColumn: Property ID for the column to be aggregated over.

2.2.1.26 CSortAggregSet

The CSortAggregSet structure contains information about group sorting.

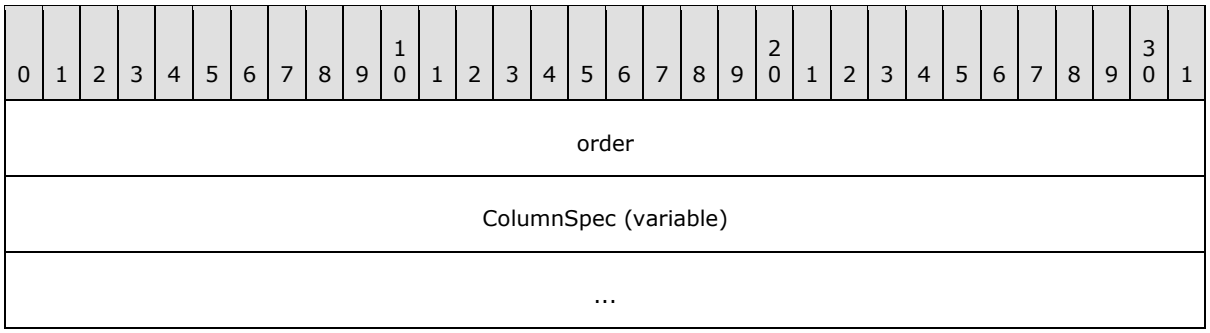


cCount: A 32-bit unsigned integer specifying the number of entries in **SortKeys**.

SortKeys: An array of CAggregSortKey structures, each describing a sort order.

2.2.1.27 CAggregSortKey

The CAggregSortKey structure contains information about sort order over single column.



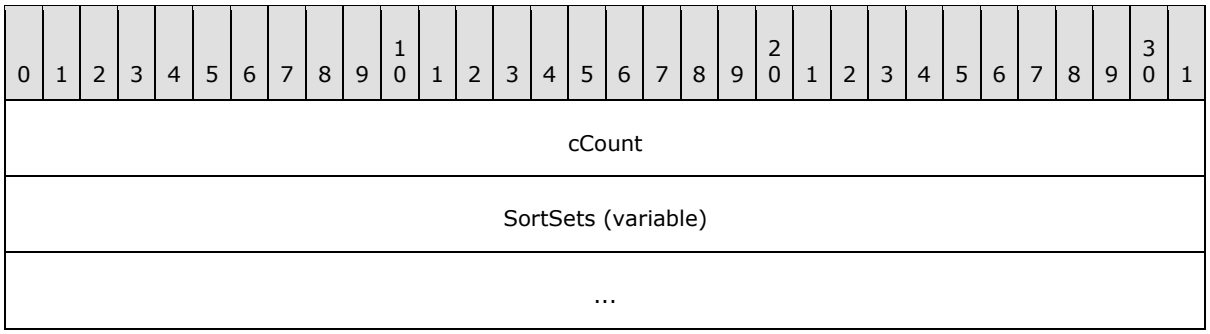
order: A 32-bit unsigned integer specifying sort order. MUST be set to one of the following values.

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

ColumnSpec: A CAggregSpec structure specifying which column to sort by.

2.2.1.28 CInGroupSortAggregSets

The CInGroupSortAggregSets structure contains information on how the group is sorted with regard to the parent's group ranges.

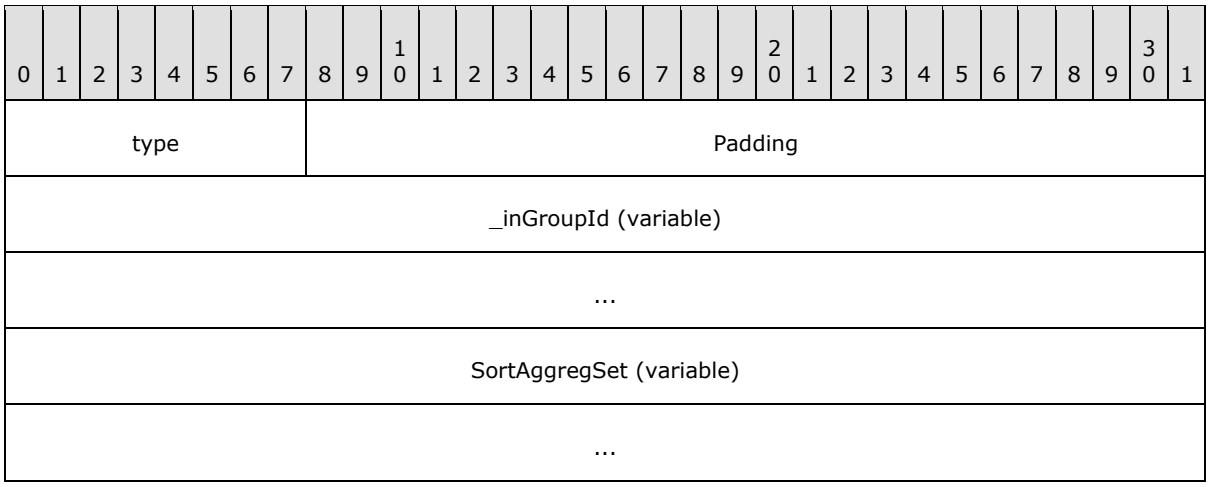


cCount: A 32-bit unsigned integer specifying the number of entries in **SortSets**.

SortSets: An array of CInGroupSortAggregSet structures.

2.2.1.29 CInGroupSortAggregSet

A CInGroupSortAggregSet structure specifies a sort order for a single range in the parent's group.



type: An 8-bit unsigned integer specifying the type of range in the parent's group. **type** MUST be set to one of the following values.

Value	Meaning
GroupIdDefault 0x00	Default for all ranges.
GroupIdMinValue 0x01	First range in the parent's group.
GroupIdNull 0x02	Last range in the parent's group.
GroupIdValue 0x03	Range with a particular range boundary value in parent's group.

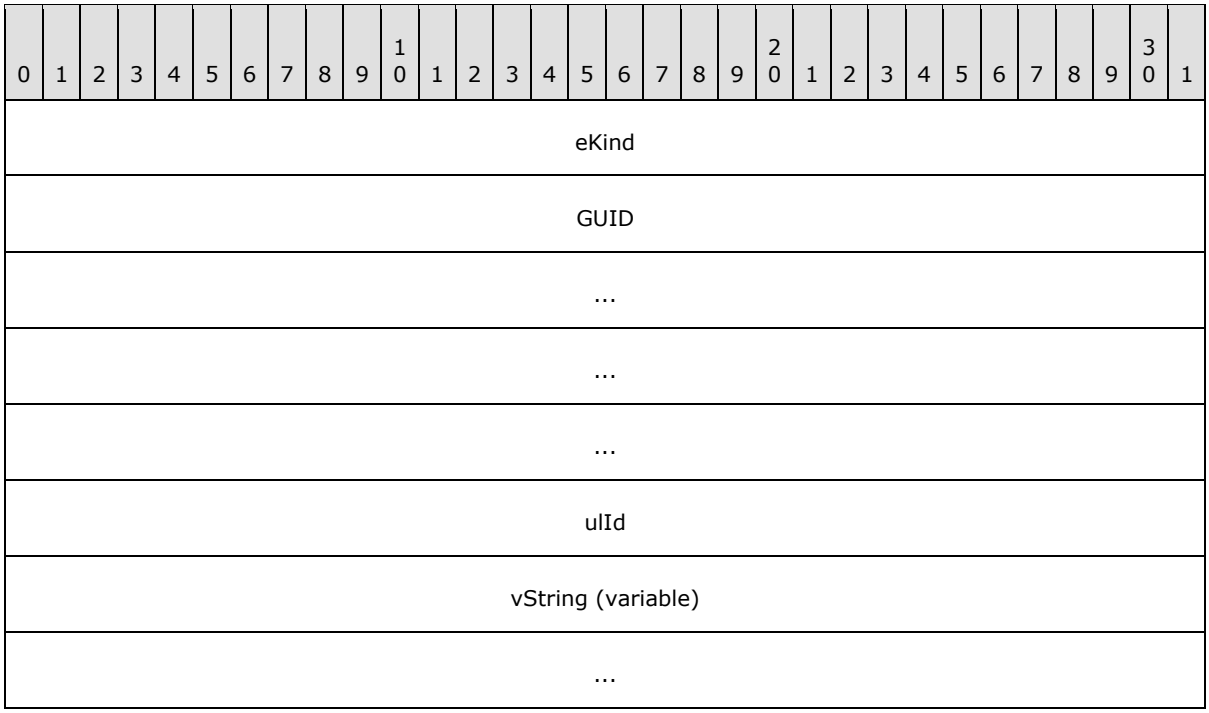
Padding: This field MUST be 3 bytes in length and the value it contains is arbitrary. The content of this field MUST be ignored by the receiver.

_inGroupId: A CBaseStorageVariant structure that contains a value representing a range in the parent's group. This field MUST not be present if type is not equal to GroupIdValue.

SortAggregSet: A CSortAggregSet structure, specifying the sort order for the range in the parent's group.

2.2.1.30 CDbColId

The CDbColId structure contains an OLE-DB Column ID.



eKind: MUST be set to one of the following values that indicates the contents of GUID and **vValue**.

Value	Meaning
DBKIND_GUID_NAME 0x00000000	**vString** contains a property name.
DBKIND_GUID_PROPID 0x00000001	**ulId** contains a 4-byte integer indicating the property ID.

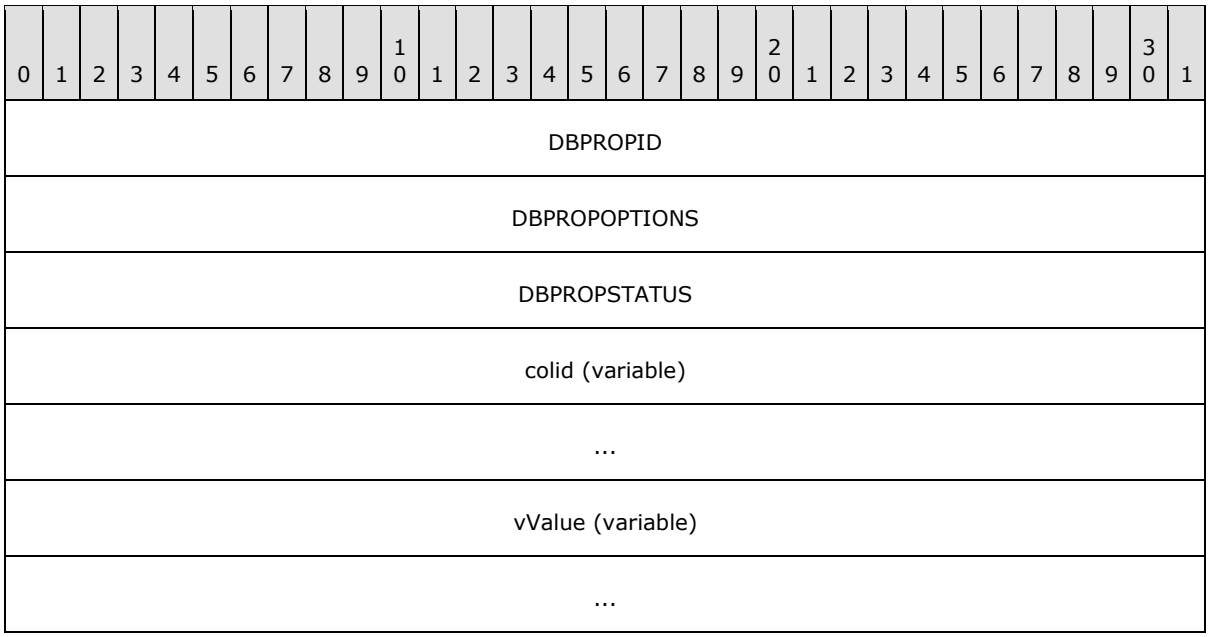
GUID: The property GUID.

ulId: If **eKind** is DBKIND_GUID_PROPID, this field contains an unsigned integer specifying the property ID. If **eKind** is DBKIND_GUID_NAME, this field contains an unsigned integer specifying the number of Unicode characters contained in the **vString** field.

vString: A non null-terminated Unicode string representing the property name. It MUST be omitted unless the **eKind** field is set to DBKIND_GUID_NAME.

2.2.1.31 CDbProp

The CDbProp structure contains an OLE-DB DBPROP database property. These properties control how queries are interpreted by the Windows Search service.



DBPROPID: A 32-bit unsigned integer indicating the property ID. This field uniquely identifies each property in a particular query, but has no other interpretation. In particular it is not a PROPID as found in the CDbColId structure.

DBPROPOPTIONS: Property options.

Note DBPROPOPTIONS MUST be set to 0x00000000.

DBPROPSTATUS: Property status.

Note DBPROPSTATUS MUST be set to 0x00000000.

colid: A CDbColId structure that defines the database property being passed.

vValue: A CBaseStorageVariant containing the property value.

2.2.1.31.1 Database Properties

This section details the properties that are used by the Windows Search Protocol to control the behavior of the Windows Search service. 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	Specifies the name of the catalog or catalogs to query. Value MUST be a

Value	Meaning
0x00000002	VT_LPWSTR or a VT_VECTOR | VT_LPWSTR.
DBPROP_CI_INCLUDE_SCOPES 0x00000003	Specifies one or more paths to be included in the query. Value MUST be a VT_LPWSTR or a VT_VECTOR | VT_LPWSTR.
DBPROP_CI_SCOPE_FLAGS 0x00000004	Specifies how the paths specified by the DBPROP_CI_INCLUDE_SCOPES property are to be treated. Value MUST be a VT_I4 or a VT_VECTOR | VT_I4.
DBPROP_CI_QUERY_TYPE 0x00000007	Specifies the type of query. The CDbColId MUST be set to DB_NULLID.

The following table lists the flags for the DBPROP_CI_SCOPE_FLAGS property.

Value	Meaning
QUERY_DEEP 0x01	If set, indicates that files in the scope directory and all subdirectories are included in the results. If clear, only files in the scope directory are included in the results. MUST NOT be combined with QUERY_DEEP.
QUERY_VIRTUAL_Path 0x02	If set, indicates that the scope is a virtual path. If clear, indicates that the scope is a physical directory.

The following table lists the query types for the DBPROP_CI_QUERY_TYPE property.

Value	Meaning
CiNormal 0x00000000	A regular query.
CiVirtualRoots 0x00000001	The query is requesting a list of the virtual roots of the catalog. This value requires administrative privileges.
CiProperties 0x00000003	The query is requesting a list of all of the properties supported by the Windows Search service.
CiAdminOp 0x00000004	The query is an administrative operation. This value requires administrative privileges.

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

Value	Meaning
DBPROP_USECONTENTINDEX 0x00000002	Use the inverted index to optimize the speed of evaluating content restrictions at the cost of the index possibly being out of date. Value MUST be a VT_BOOL. If TRUE, the server is allowed to fail these queries.
DBPROP_DEFERNONINDEXEDTRIMMING 0x00000003	Some operations, such as filtering by scope or security, can be expensive. This flag indicates that it is acceptable to defer this filtering until the results are actually requested. Value MUST be a VT_BOOL.

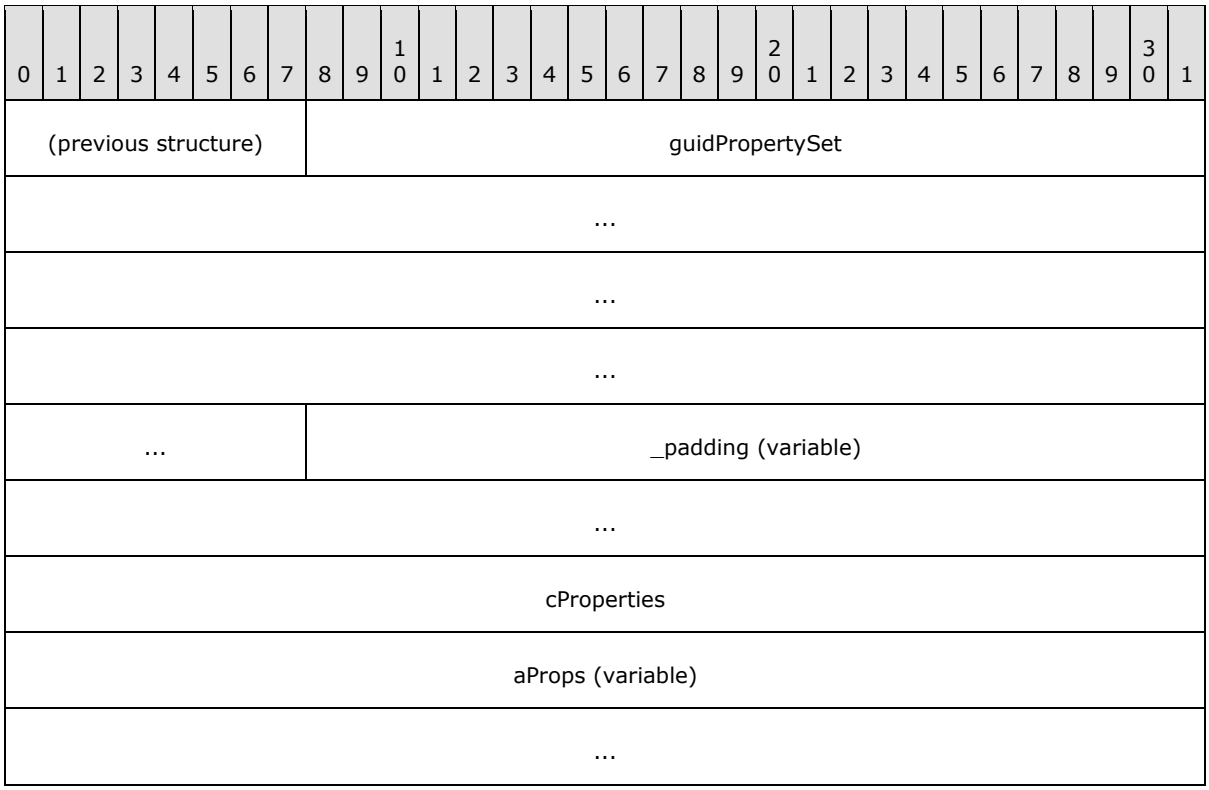
Value	Meaning
DBPROP_USEEXTENDEDDBTYPES 0x00000004	Indicates if the client supports VT_VECTOR data types. If TRUE, the client supports VT_VECTOR; if FALSE, the server is to convert VT_VECTOR data types to VT_ARRAY data types. The value MUST be a VT_BOOL.
DBPROP_FIRSTROWS 0x00000007	If TRUE, the Windows Search service should return the first rows that match. If FALSE, then rows should by default be returned in order of descending rank. Value MUST be a VT_BOOL.

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

Value	Meaning
DBPROP_MACHINE 0x00000002	Specifies the names of the computers on which a query is to be processed. The value MUST be either VT_BSTR or VT_ARRAY | VT_BSTR.
DBPROP_CLIENT_CLSID 0x00000003	Specifies a connection constant for the Windows Search service. The value MUST be a VT_CLSID containing 0x2A4880706FD911D0A80800A0C906241A.

2.2.1.32 CDbPropSet

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



guidPropertySet: A GUID identifying the property set. MUST be set to the binary form corresponding to one of the following values (shown in string representation form), identifying the property set of the properties contained in the **aProps** field.

Value/GUID	Meaning
DBPROPSET_FSCIFRMWRK_EXT A9BD1526-6A80-11D0-8C9D-0020AF1D740E	File system content index framework property set.
DBPROPSET_QUERYEXT A7AC77ED-F8D7-11CE-A798-0020F8008025	Query extension property set.
DBPROPSET_CIFRMWRKCORE_EXT AFAFACA5-B5D1-11D0-8C62-00C04FC2DB8D	Content index framework core property set.

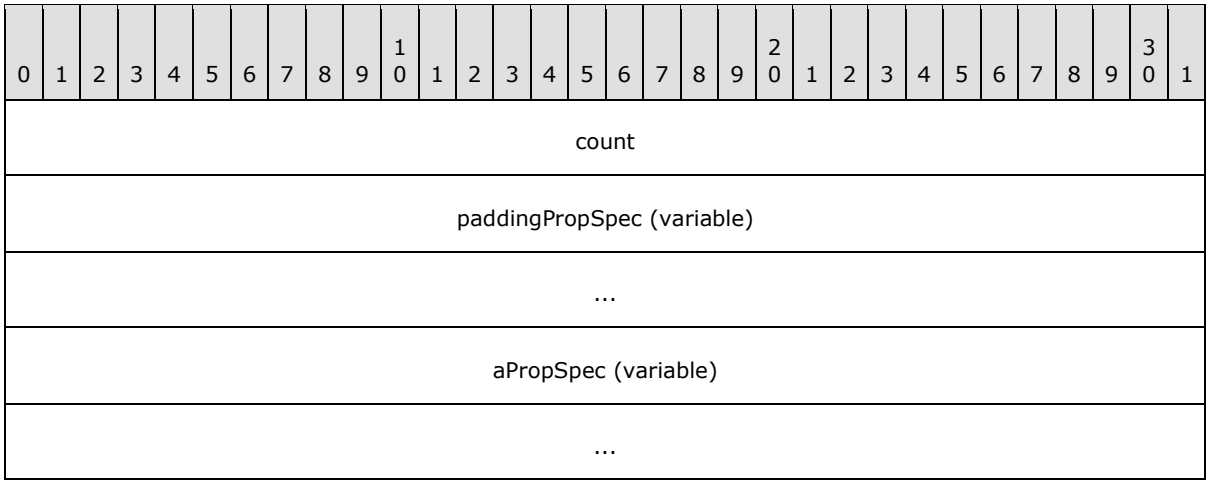
_padding: This field MUST be 0 to 3 bytes in length. The length of this field MUST be such that the following field begins at an offset that is a multiple of 4 bytes from the beginning of the message that contains this structure. If this field is present (that is, length nonzero), the value it contains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

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

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

2.2.1.33 CPidMapper

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



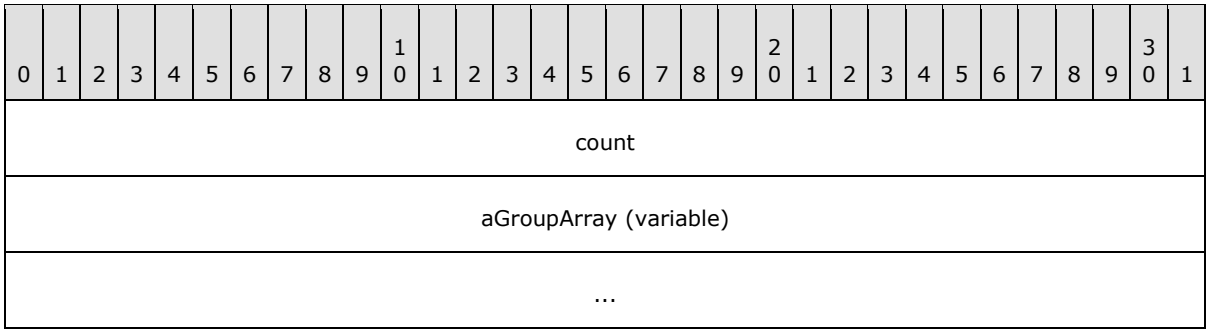
count: A 32-bit unsigned integer containing the number of elements in the **aPropSpec** array.

paddingPropSpec: This field MUST be 0 to 4 bytes in length. The length of this field MUST be such that the byte offset from the beginning of the message to the first structure contained in the **aPropSpec** field is a multiple of 8. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

aPropSpec: Array of CFullPropSpec structures indicating the properties to return. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure has a 4-byte alignment from the beginning of a message. Such padding bytes can be any arbitrary value and MUST be ignored on receipt.

2.2.1.34 CColumnGroupArray

The CColumnGroupArray structure contains a set of property groups with weights for each property.

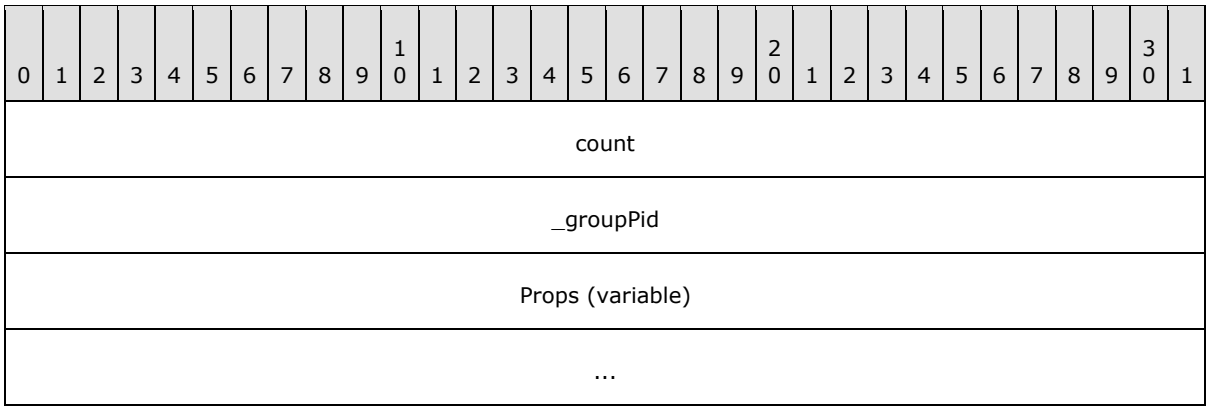


count: A 32-bit unsigned integer containing the number of elements in the **aGroupArray** array.

aGroupArray: An array of CColumnGroup structures indicating individual weights for each property, which are used in probabilistic ranking. Structures in the array MUST be separated by 0 to 3 padding bytes such that each structure has a 4-byte alignment from the beginning of a message. Such padding bytes can contain any arbitrary value and MUST be ignored on receipt.

2.2.1.35 CColumnGroup

The CColumnGroup structure contains information about a property's weight in a single group.



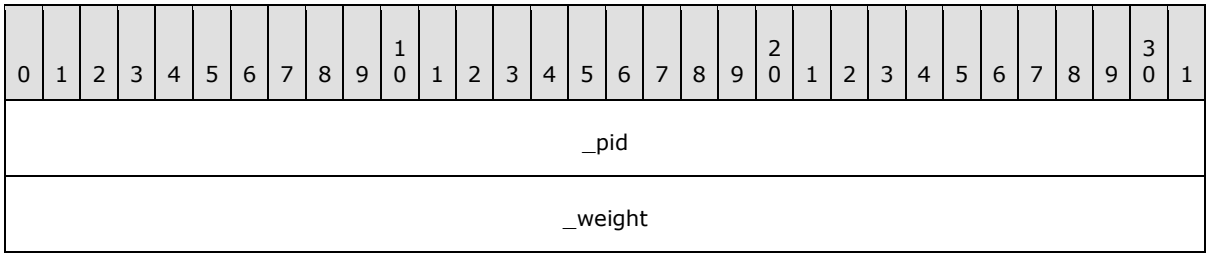
count: A 32-bit unsigned integer containing the number of elements in the **Props** array.

_groupPid: A 32-bit unsigned integer specifying group ID, full property specification for which can be used in the corresponding CProbRestriction.

Props: An array of SProperty structures each specifying a PID and a weight for a property.

2.2.1.36 SProperty

The SProperty structure contains information about single property weight.

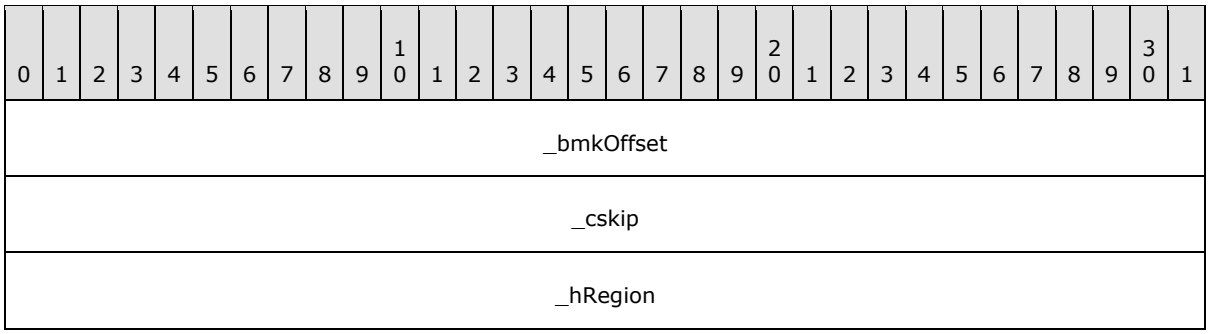


_pid: A 32-bit unsigned integer specifying a property identifier.

_weight: A 32-bit unsigned integer specifying the weight to be used in probabilistic ranking.

2.2.1.37 CRowSeekAt

The CRowSeekAt structure contains the offset at which to retrieve rows in a CPMGetRowsIn message.



_bmkOffset: A 32-bit value representing the handle of the bookmark indicating the starting position from which to skip the number of rows specified in **_cskip**, before beginning retrieval.

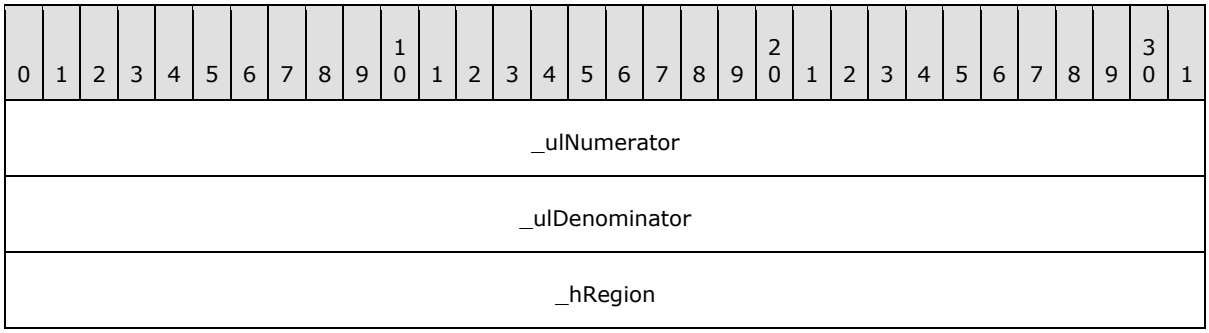
_cskip: A 32-bit unsigned integer containing the number of rows to skip in the rowset.

_hRegion: A 32-bit unsigned integer.

Note This field MUST be set to 0x00000000, and MUST be ignored.

2.2.1.38 CRowSeekAtRatio

The CRowSeekAtRatio structure identifies the point at which to begin retrieval for a CPMGetRowsIn message.



_ulNumerator: A 32-bit unsigned integer representing the numerator of the ratio of rows in the chapter at which to begin retrieval.

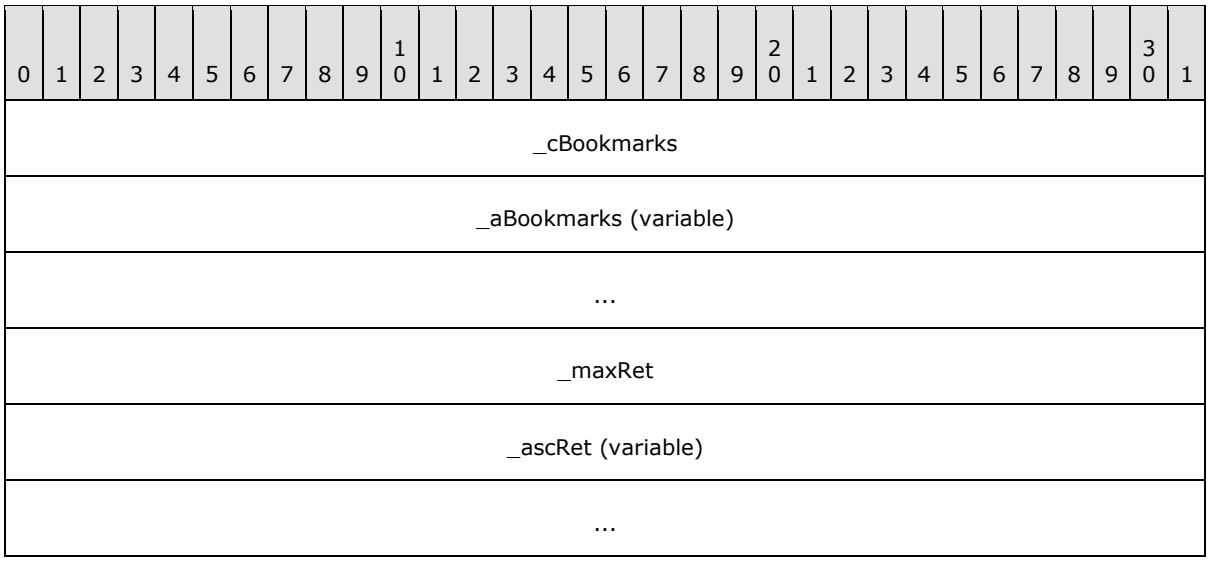
_ulDenominator: A 32-bit unsigned integer representing the denominator of the ratio of rows in the chapter at which to begin retrieval. MUST be greater than zero.

_hRegion: A 32-bit unsigned integer.

Note This field MUST be set to 0x00000000, and MUST be ignored.

2.2.1.39 CRowSeekByBookmark

The CRowSeekByBookmark structure identifies the bookmarks from which to begin retrieving rows for a CPMGetRowsIn message.



_cBookmarks: A 32-bit unsigned integer representing the number of elements in _aBookmarks array.

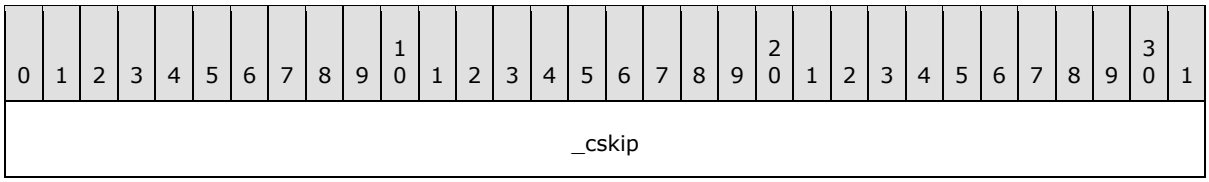
_aBookmarks: An array of bookmark handles (each represented by 4 bytes), as obtained from a CPMGetRowsOut message.

_maxRet: A 32-bit unsigned integer representing the number of elements in the **_ascRet** array.

_ascRet: An array of HRESULT values. When the CRowSeekByBookmark is sent as part of the CPMGetRowsIn request, the number of entries in the array MUST be equal to **_maxRet**. When sent by the client, the values MUST be set to zero, and the server MUST ignore the contents of the array. When sent by the server (as part of the CPMGetRowsOut message), the values in the array indicate the result status for each row retrieval.

2.2.1.40 CRowSeekNext

The CRowSeekNext structure contains the number of rows to skip in a CPMGetRowsIn message.



_cskip: A 32-bit unsigned integer representing the number of rows to skip in the rowset.

2.2.1.41 CRowsetProperties

The CRowsetProperties structure contains configuration information for a 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																															

_uBooleanOptions: The least significant 3 bits of this field MUST contain one of the following three values.

Value	Meaning
eSequential 0x00000001	The **cursor** can only be moved forward.
eLocatable 0x00000003	The cursor can be moved to any position.
eScrollable 0x00000007	The cursor can be moved to any position and fetch in any direction.

The remaining bits MAY either be clear or set to any combination of the following values logically ORed together.

Value	Meaning
eAsynchronous 0x00000008	The client will not wait for execution completion.
eFirstRows 0x00000080	Return the first rows encountered, not the best matches.
eHoldRows 0x00000200	The server MUST NOT discard rows until the client is done with a query.
eChaptered 0x00000800	The rowset supports chapters.
eUseCI 0x00001000	Use the inverted index to evaluate content restrictions even though it may be out of date. If not set, the Windows Search service can opt to execute the query by going directly against the file system.
eDeferTrimming 0x00002000	Non-indexed trimming operations like scoping or security checking can be expensive. This option gives the Windows Search service the option of deferring these operations until rows are actually requested.

_ulMaxOpenRows: A 32-bit unsigned integer.

Note This field MUST be set to 0x00000000. It is not used and MUST be ignored.

_ulMemoryUsage: A 32-bit unsigned integer.

Note This field MUST be set to 0x00000000. It is not used and MUST be ignored.

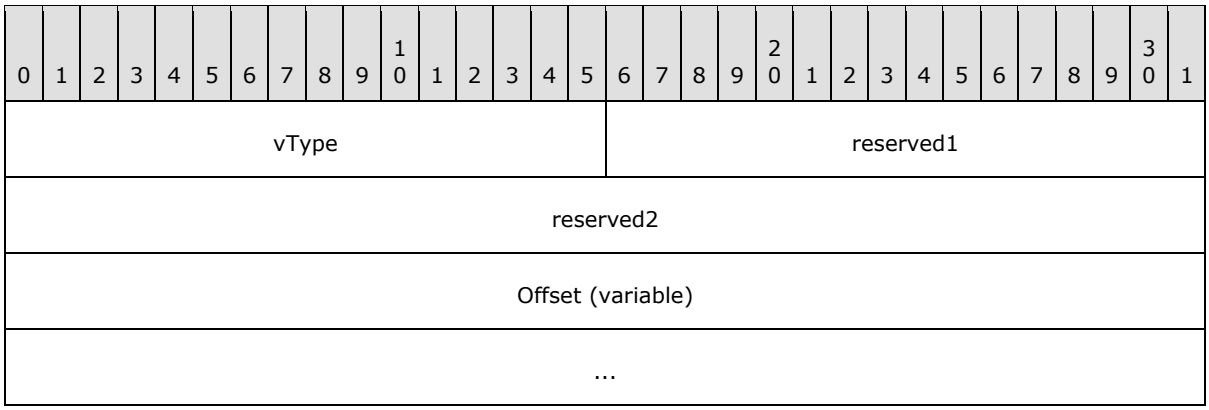
_cMaxResults: A 32-bit unsigned integer specifying the maximum number of rows that are to be returned for the query.

_cCmdTimeout: A 32-bit unsigned integer, specifying the number of seconds at which a query is to time out and automatically terminate, counting from the time the query starts executing on the server.

Note A value of 0x00000000 means that the query is not to time out.

2.2.1.42 CRowVariant

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



vType: A type indicator, indicating the type of **vValue**. It MUST be one of the values under the **vType** field, as specified in section 2.2.1.1.

reserved1: Not used. The value MAY be set to any arbitrary value, and it MUST be ignored on receipt.

reserved2: Not used. The value MAY be set to any arbitrary value, and it MUST be ignored on receipt.

Offset: An offset to variable length data (for example, a string). This MUST be a 32-bit value (4 bytes long) if 32-bit offsets are being used (per the rules in section 2.2.3.14), or a 64-byte value (8 bytes long) if 64-bit offsets are being used.

2.2.1.43 CSortSet

The CSortSet structure contains the sort order of the 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
count																															
sortArray (variable)																															
...																															

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

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

2.2.1.44 CTableColumn

The CTableColumn structure contains a column of a CPMSetBindingsIn message.

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
PropSpec (variable)																															
...																															
vType																															
AggregateUsed								AggregateType								ValueUsed								_padding1 (optional)							
ValueOffset (optional)																				ValueSize (optional)											
StatusUsed								_padding2 (optional)												StatusOffset (optional)											
LengthUsed								_padding3 (optional)												LengthOffset (optional)											

PropSpec: A CFullPropSpec structure.

vType: A 32-bit unsigned integer that specifies the type of data value contained in the column. See the **vType** field in section 2.2.1.1 for the list of values for this field.

AggregateUsed: MUST be set to one of the following values.

Value	Meaning
0x00	No aggregation is used, and the **AggregateType** field MUST NOT be present.

Value	Meaning
0x01	This column is used to aggregate the values for query results, as specified in **AggregateType**.

AggregateType:

Note This field MUST be set to one of the aggregation type values specified under the **type** field in section 2.2.1.25.

Value	Meaning
DBAGGTTYPE_BYNONE 0x00	No aggregation is used.
DBAGGTTYPE_SUM 0x01	An aggregated sum of results is used.
DBAGGTTYPE_MAX 0x02	A result's maximum value is used.
DBAGGTTYPE_MIN 0x03	A result's minimum value is used.
DBAGGTTYPE_AVG 0x04	An average value of results is used.
DBAGGTTYPE_COUNT 0x05	A count of results in the result group and all subgroups is used.
DBAGGTTYPE_CHILDCOUNT 0x06	Count only number of top level results in group, ignore subgroups count.

ValueUsed: A 1-byte field. MUST be set to one of the following values.

Value	Meaning
0x00	The value of the column is not transferred in the row.
0x01	The value of the column is transferred within the row.

_padding1: A padding field.

Note This field MUST be inserted before **ValueOffset** if, without it, **ValueOffset** would not begin at an even offset from the beginning of the message. The value of this byte is arbitrary, and MUST be ignored. If **ValueUsed** is set to 0x00, this field MUST NOT be present.

ValueOffset: An unsigned 2-byte integer specifying the offset of the column value in the row. If **ValueUsed** is set to 0x00, this field MUST NOT be present.

ValueSize: An unsigned 2-byte integer specifying the size of the column value in bytes. If **ValueUsed** is set to 0x00, this field MUST NOT be present.

StatusUsed: MUST be set to one of the following values.

Value	Meaning
0x00	The status of the column is not transferred within the row.
0x01	The status of the column is transferred within the row.

_padding2: A padding field.

Note This field MUST be inserted before **StatusOffset** if, without it, the **StatusOffset** field would not begin at an even offset from the beginning of the message. The value of this byte is arbitrary and MUST be ignored. If **StatusUsed** is set to 0x00, this field MUST NOT be present.

StatusOffset: An unsigned 2-byte integer.

Note Specifies the offset of the column status in the row. If **StatusUsed** is set to 0x00, this field MUST NOT be present.

LengthUsed: A 1-byte field that MUST be set to one of the following values.

Value	Meaning
0x00	The length of the column MUST NOT be transferred within the row.
0x01	The length of the column is transferred within the row.

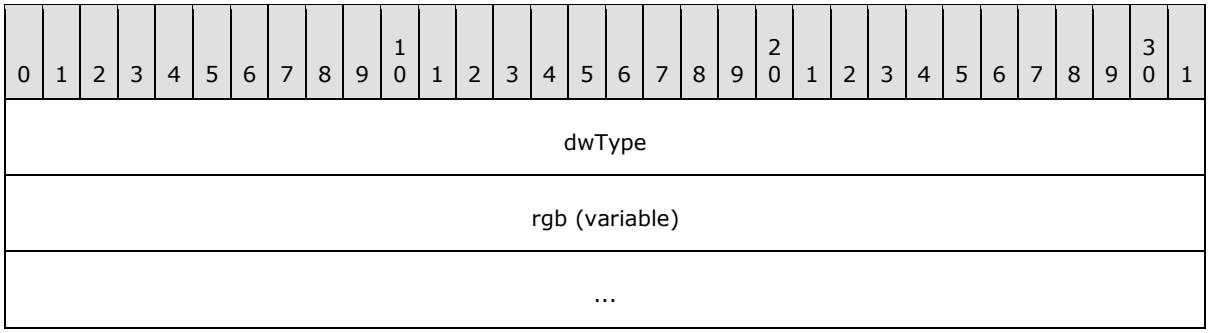
_padding3: A padding field.

Note This field MUST be inserted before **LengthOffset** if, without it, **LengthOffset** would not begin at an even offset from the beginning of a message. The value of this byte is arbitrary and MUST be ignored. If **LengthUsed** is set to 0x00, this field MUST NOT be present.

LengthOffset: An unsigned 2-byte integer specifying the offset of the column length in the row. In CPMGetRowsOut, length is represented by a 32-bit unsigned integer by the offset specified in **LengthOffset**. If **LengthUsed** is set to 0x00, this field MUST NOT be present.

2.2.1.45 SERIALIZEDPROPERTYVALUE

The SERIALIZEDPROPERTYVALUE structure contains a serialized value.



dwType: One of the variant types, as defined in section 2.2.1.1, that can be combined with variant type modifiers. For all variant types, except those combined with VT_ARRAY, SERIALIZEDPROPERTYVALUE has the same layout as CBaseStorageVariant. If the variant type

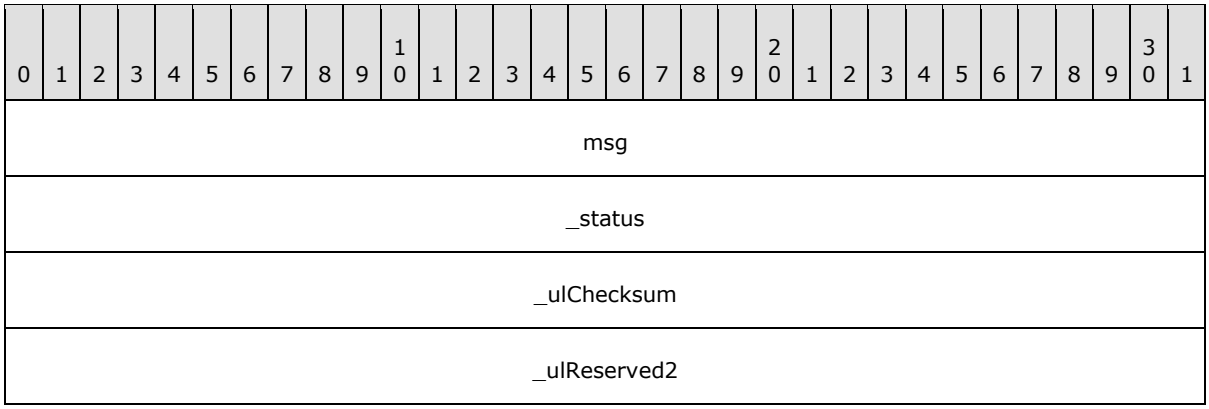
is combined with the VT_ARRAY type modifier, SAFEARRAY2 is used instead of SAFEARRAY in the **vValue** field of CBaseStorageVariant.

rgb: Serialized value. The serializiation depends on the value of **dwType** which is used identically to **vValue** in section 2.2.1.1.

2.2.2 Message Headers

All Windows Search Protocol messages have a 16-byte header.

The following diagram shows the Windows Search Protocol message header format.



msg: A 32-bit integer that identifies the type of message following the header. The following table lists the Windows Search Protocol messages and the integer values specified for each message. As shown in the table, some values identify two messages in the table. In those instances the message following the header can be identified by the direction of the message flow. If the direction is client to server, the message with "In" appended to the message name is indicated. If the direction is server to client, the message with "Out" appended to the message name is indicated.

Value	Meaning
0x000000C8	CPMConnectIn or CPMConnectOut
0x000000C9	CPMDisconnect
0x000000CA	CPMCreateQueryIn or CPMCreateQueryOut
0x000000CB	CPMFreeCursorIn or CPMFreeCursorOut
0x000000CC	CPMGetRowsIn or CPMGetRowsOut
0x000000CD	CPMRatioFinishedIn or CPMRatioFinishedOut
0x000000CE	CPMCompareBmkIn or CPMCompareBmkOut
0x000000CF	CPMGetApproximatePositionIn or CPMGetApproximatePositionOut
0x000000D0	CPMSetBindingsIn
0x000000D1	CPMGetNotify

Value	Meaning
0x000000D2	CPMSendNotifyOut
0x000000D7	CPMGetQueryStatusIn or CPMGetQueryStatusOut
0x000000D9	CPMCiStateInOut
0x000000E1	CPMForceMergeIn
0x000000E4	CPMFetchValueIn or CPMFetchValueOut
0x000000E6	CPMUpdateDocumentsIn
0x000000E7	CPMGetQueryStatusExIn or CPMGetQueryStatusExOut
0x000000E8	CPMRestartPositionIn
0x000000E9	CPMStopAsynchIn
0x000000EC	CPMSetCatStateIn (not supported)

_status: An HRESULT, indicating the status of the requested operation.<5>

_ulChecksum: The _ulChecksum MUST be calculated as specified in section 3.1.4 for the following messages:

- CPMConnectIn
- CPMCreateQueryIn
- CPMSetBindingsIn
- CPMGetRowsIn
- CPMFetchValueIn

Note For all other messages, _ulChecksum MUST be set to 0x00000000. A client MUST ignore the _ulChecksum field.

_ulReserved2: MUST be ignored by the receiver.

Note This field MUST be set to 0x00000000.

2.2.3 Messages

2.2.3.1 CPMCiStateInOut

The CPMCiStateInOut message contains information about the state of the Windows Search service.

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
cbStruct																															
cWordList																															
cPersistentIndex																															
cQueries																															
cDocuments																															
cFreshTest																															
dwMergeProgress																															
eState																															
cFilteredDocuments																															
cTotalDocuments																															
cPendingScans																															
dwIndexSize																															
cUniqueKeys																															
cSecQDocuments																															
dwPropCacheSize																															

cbStruct: A 32-bit unsigned integer indicating the size, in bytes, of this message (excluding the common header). MUST be set to 0x0000003C.

cWordList: A 32-bit unsigned integer containing the number of in-memory indexes created for recently indexed documents.

cPersistentIndex: A 32-bit unsigned integer containing the number of persisted indexes.

cQueries: A 32-bit unsigned integer indicating a number of actively running queries.

cDocuments: A 32-bit unsigned integer indicating the total number of documents waiting to be indexed.

cFreshTest: A 32-bit unsigned integer indicating the number of unique documents with information in indexes that are not fully optimized for performance.

dwMergeProgress: A 32-bit unsigned integer specifying the completion percentage of current full optimization of indexes while optimization is in progress. MUST be less than or equal to 100.

eState: A 32-bit unsigned integer indicating the state of content indexing. MUST be one or more of the CI_STATE_* constants defined in the following table.

Value	Meaning
CI_STATE_SHADOW_MERGE 0x00000001	The Windows Search service is in the process of optimizing some of the indexes to reduce memory usage and improve query performance.
CI_STATE_MASTER_MERGE 0x00000002	The Windows Search service is in the process of full optimization for all indexes.
CI_STATE_CONTENT_SCAN_REQUIRED 0x00000004	Some documents in the inverted index have changed and the Windows Search service needs to determine which have been added, changed, or deleted.
CI_STATE_ANNEALING_MERGE 0x00000008	The Windows Search service is in the process of optimizing indexes to reduce memory usage and improve query performance. This process is more comprehensive than the one identified by the CI_STATE_SHADOW_MERGE value, but it is not as comprehensive as specified by the CI_STATE_MASTER_MERGE value. Such optimizations are implementation-specific as they depend on the way data is stored internally; the optimizations do not affect the protocol in any way other than response time.
CI_STATE_SCANNING 0x00000010	The Windows Search service is examining a directory or a set of directories to see if any files have been added, deleted, or updated since the last time the directory was indexed.
CI_STATE_LOW_MEMORY 0x00000080	Most of the virtual memory of the server is in use.
CI_STATE_HIGH_IO 0x00000100	The level of input/output (I/O) activity on the server is relatively high.
CI_STATE_MASTER_MERGE_PAUSED 0x00000200	The process of full optimization for all indexes that was in progress has been paused. This is given for informative purposes only and does not affect the Windows Search Protocol.
CI_STATE_READ_ONLY 0x00000400	The portion of the Windows Search service that picks up new documents to index has been paused. This is given for informative purposes only and does not affect the Windows Search Protocol.
CI_STATE_BATTERY_POWER 0x00000800	The portion of the Windows Search service that picks up new documents to index has been paused to conserve battery lifetime but still replies to the queries. This is given for informative purposes only and does not affect

Value	Meaning
	the Windows Search Protocol.
CI_STATE_USER_ACTIVE 0x00001000	The portion of the Windows Search service that picks up new documents to index has been paused due to high activity by the user (keyboard or mouse) but still replies to the queries. This is given for informative purposes only and does not affect the Windows Search Protocol.
CI_STATE_LOW_DISK 0x00010000	The service is paused due to low disk availability.
CI_STATE_HIGH_CPU 0x00020000	The service is paused due to high CPU usage.

cFilteredDocuments: A 32-bit unsigned integer indicating the number of documents indexed since content indexing was begun.

cTotalDocuments: A 32-bit unsigned integer indicating the total number of documents in the system.

cPendingScans: A 32-bit unsigned integer indicating the number of pending high-level indexing operations. The meaning of this value is provider-specific, but larger numbers are expected to indicate that more indexing remains.<6>

dwIndexSize: A 32-bit unsigned integer indicating the size, in megabytes, of the index (excluding the property cache).

cUniqueKeys: A 32-bit unsigned integer indicating the approximate number of unique keys in the catalog.

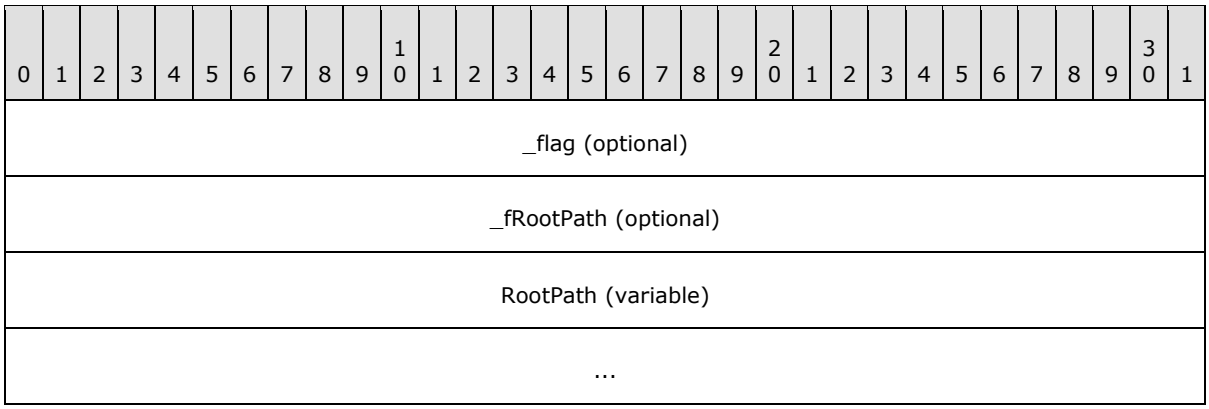
cSecQDocuments: A 32-bit unsigned integer indicating the number of documents that the Windows Search service will attempt to index again because of a failure during the initial indexing attempt.

dwPropCacheSize: A 32-bit unsigned integer indicating the size, in megabytes, of the property cache.

2.2.3.2 CPMUpdateDocumentsIn

The CPMUpdateDocumentsIn message directs the server to index the specified path.

The server will reply with the message header of the CPMUpdateDocumentsOut message with the results of the request contained in the **_status** field of the message header.



_flag: A 32-bit unsigned integer indicating the type of update to be performed. This field MUST be present when the message is sent by the client and MUST be absent when the message is sent by the server. This field MUST be set to one of the following values.

Value	Meaning
UPD_INCREM 0x00000000	An incremental update is to be performed.
UPD_FULL 0x00000001	A full update is to be performed.
UPD_INIT 0x00000002	A new initialization is to be performed.

_fRootPath: A Boolean value indicating if the **RootPath** field specifies a path on which to perform the update.

Note This field MUST be present when the message is sent by the client and MUST be absent when the message is sent by the server. This field MUST be set to 0x00000001 or 0x00000000. If set to 0x00000001, a path on which to perform the update is included in **RootPath**. If set to 0x00000000, the update is to be performed on all indexed paths.

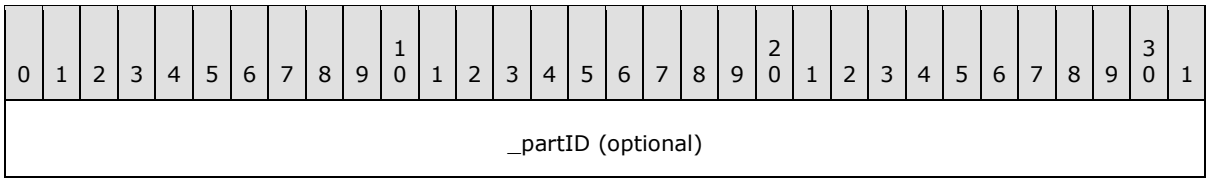
RootPath: The name of the path to be updated.

Note This field MUST be present when the message is sent by the client and MUST be absent when the message is sent by the server. The name MUST be a null-terminated Unicode string. This field MUST be omitted if **_fRootPath** is set to 0x00000000.

2.2.3.3 CPMForceMergeIn

The CPMForceMergeIn message requests a server to perform any maintenance necessary to improve query performance. The server will reply with the message header of the CPMForceMergeIn message with the results of the request contained in the **_status** field.

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



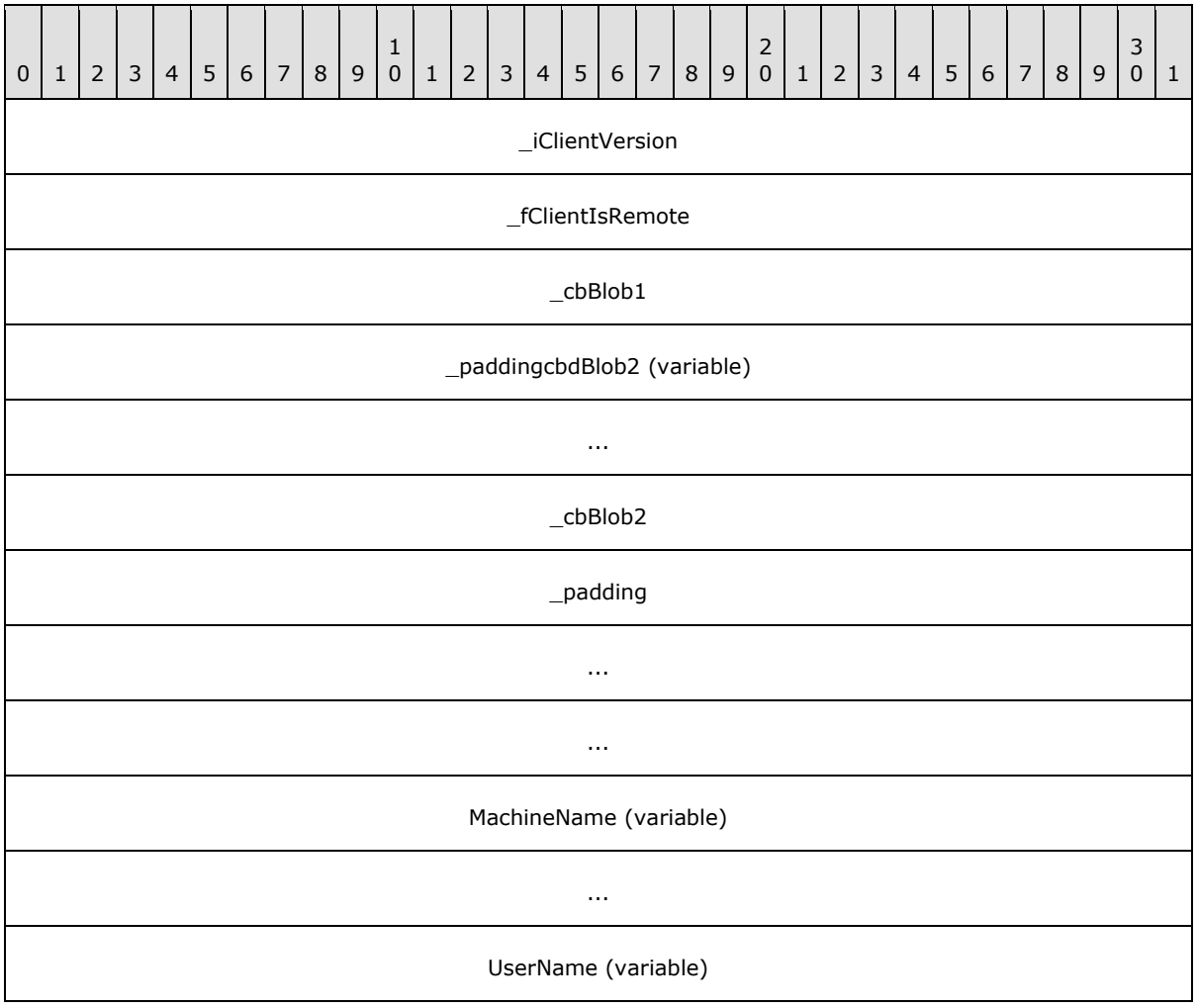
_partID: A 32-bit unsigned integer.

Note This field MUST be present when the message is sent by the client and MUST be absent when the message is sent by the server. When this field is present, it MUST be set to 0x00000001.

2.2.3.4 CPMConnectIn

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

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



...
_paddingcPropSets (variable)
...
cPropSets
PropertySet1 (variable)
...
PropertySet2 (variable)
...
PaddingExtPropset (variable)
...
cExtPropSet
aPropertySets (variable)
...

_iClientVersion: A 32-bit integer indicating whether the server is to validate the checksum value specified in the **_ulChecksum** field of the message headers for messages sent by the client.

Note If the **_iClientVersion** field is set to 0x00000102 or greater, the server MUST validate the **_ulChecksum** field value for the following messages:

- CPMConnectIn
- CPMCreateQueryIn
- CPMFetchValueIn
- CPMGetRowsIn
- CPMSetBindingsIn

For how the server validates the value specified by the client in the **_ulChecksum** field for the messages previously listed, see section 3.2.5.

If the value is greater than 0x00000102, the client is assumed capable of handling 64-bit offsets in CPMGetRowsOut messages.<7>

_fClientIsRemote: A Boolean value indicating if the client is running on a different machine than the server.

Note This field MUST be set to 0x00000001.

_cbBlob1: A 32-bit unsigned integer indicating the size in bytes of the **cPropSet**, **PropertySet1**, and **PropertySet2** fields, combined.

_paddingcbdBlob2: This field MUST be 0 to 4 bytes in length. The length of this field MUST be such that the byte offset from the beginning of the message to the first structure contained in the **_cbBlob2** field is a multiple of 8. The value of the bytes can be any arbitrary value, and MUST be ignored by the receiver.

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

_padding: 12 bytes of padding that can contain arbitrary values and MUST be ignored.

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

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

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

cPropSets: A 32-bit unsigned integer indicating the number of CDbPropSet structures following this field.

Note This field MUST be set to 0x0000002.

PropertySet1: A CDbPropSet structure with **guidPropertySet** containing DBPROPSET_FSCIFRMWRK_EXT.

PropertySet2: A CDbPropSet structure with **guidPropertySet** containing DBPROPSET_CIFRMWRKCORE_EXT.

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

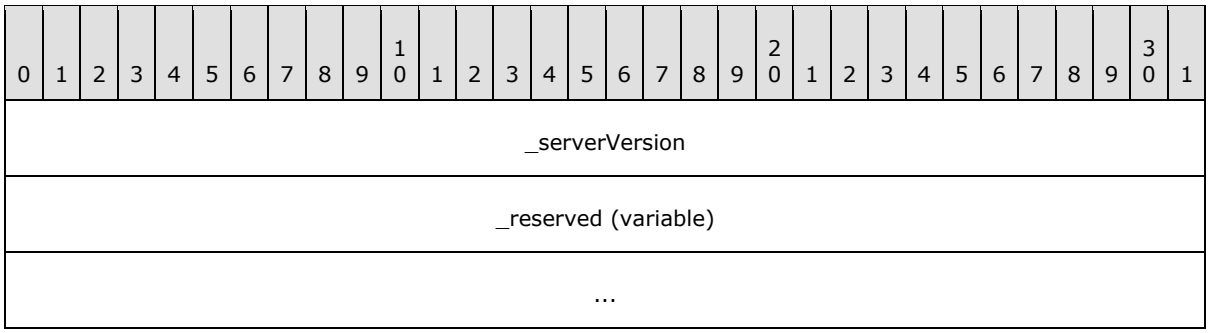
cExtPropSet: A 32-bit unsigned integer indicating the number of CDbPropSet structures following this field.

aPropertySets: An array of CDbPropSet structures specifying other properties. The number of elements in this array MUST be equal to **cExtPropSet**.

2.2.3.5 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.



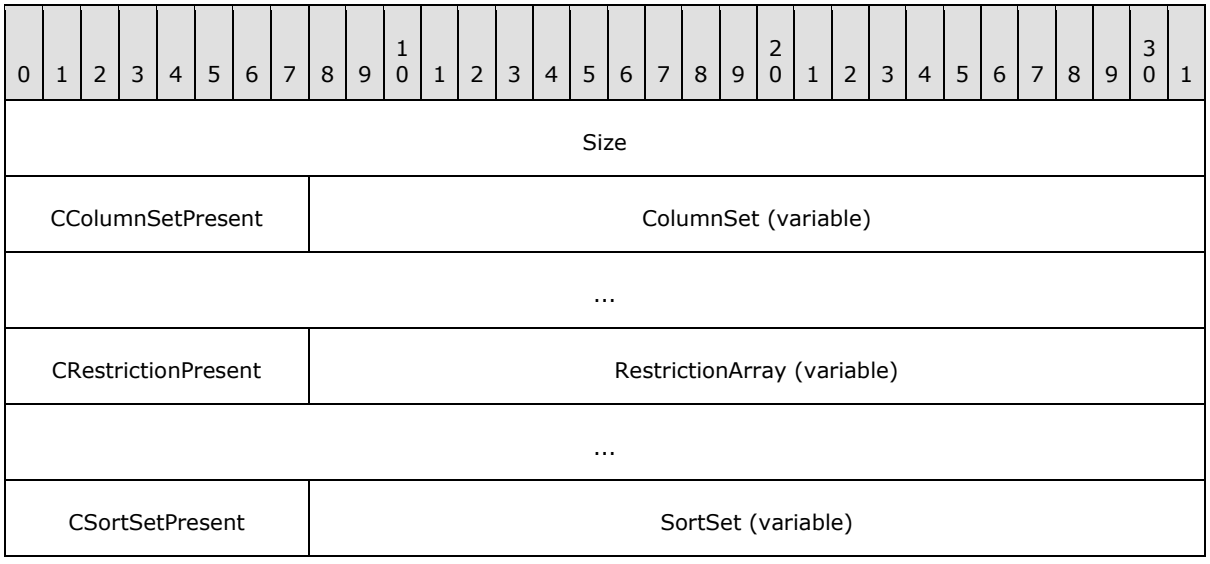
_serverVersion: A 32-bit integer that indicates whether the server can support 64-bit offsets, as specified in section 2.2.3.14.

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

_reserved: Reserved. The server MAY send an arbitrary number of arbitrary values, and the client MUST ignore these values if present.

2.2.3.6 CPMCreateQueryIn

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



...	
CCategorizationSetPresent	CCategorizationSet (variable)
...	
RowSetProperties	
...	
...	
...	
...	
PidMapper (variable)	
...	
GroupArray (variable)	
...	
Lcid	

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

CColumnSetPresent: A byte field indicating if the **ColumnSet** field is present. 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.

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

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

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

RestrictionArray: A CRestrictionArray structure containing the command tree of the query.

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

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

SortSet: A CSortSet structure indicating the sort order of the query.

CCategorizationSetPresent: A byte field indicating if the **CCategorizationSet** field is present.

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

CCategorizationSet: A CCategorizationSet structure that contains the groups for the query.

RowSetProperties: A CRowsetProperties structure providing configuration information for the query.

PidMapper: A CPidMapper structure that maps from property offsets to full property descriptions.

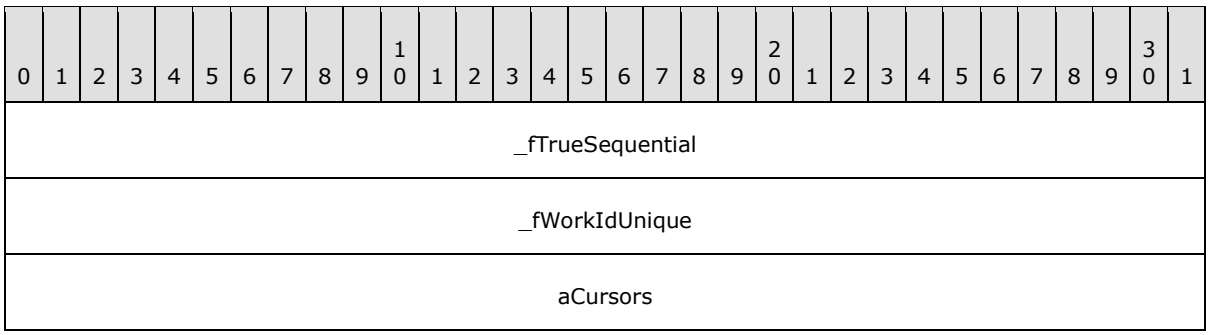
GroupArray: A CColumnGroupArray structure, describing property weights for probabilistic ranking.

Lcid: A 32-bit unsigned integer, indicating user's locale for this query, as specified in [MS-LCID].

2.2.3.7 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: A 32-bit unsigned integer. MUST be set to one of the following values:

Note An informative value indicating if the query can be expected to provide results faster.

Value	Meaning
0x00000000	For the query provided in CPMCreateQueryIn, there would be a greater latency in delivering query results.
0x00000001	For the query provided in CPMCreateQueryIn, the server can use the inverted index in such a way that query results will likely be delivered faster.

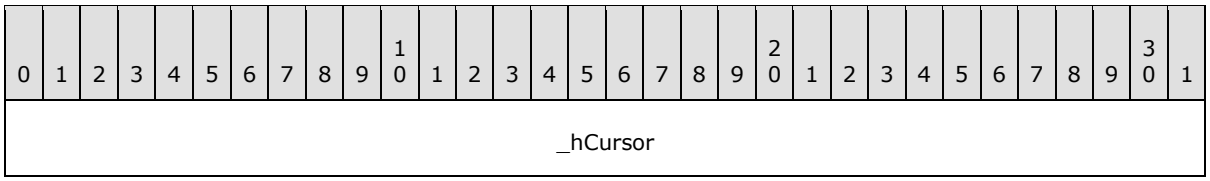
_fWorkIdUnique: A Boolean value indicating if the document identifiers pointed by the cursors are unique throughout query results. MUST be set to one of the following values:

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

aCursors: An array of 32-bit unsigned integers representing the handles to cursors with the number of elements equal to the number of categories in the **CategorizationSet** field of CPMCreateQueryIn message plus one element, representing an uncategorized cursor.

2.2.3.8 CPMGetQueryStatusIn

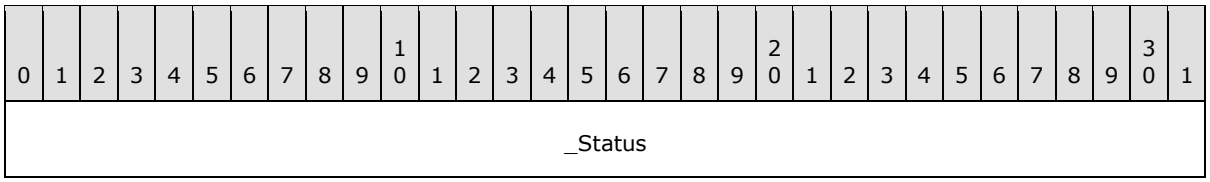
The CPMGetQueryStatusIn message requests the status of a query. The format of the CPMGetQueryStatusIn message that follows the header is shown in the following diagram.



_hCursor: A 32-bit unsigned integer representing the handle from the CPMCreateQueryOut message identifying the query for which to retrieve status information.

2.2.3.9 CPMGetQueryStatusOut

The CPMGetQueryStatusOut message replies to a CPMGetQueryStatusIn message with the status of the query. The format of the CPMGetQueryStatusOut message that follows the header is shown in the following diagram.



_Status: A 32-bit unsigned integer. A bitmask of values defined in the following tables that describe the query.

The following table lists STAT_* values obtained by performing a bitwise AND operation on **_Status** with 0x00000007. The result MUST be one of the following.

Constant	Meaning
STAT_NOISE_WORDS 0x00000000	The asynchronous query is still running.
STAT_ERROR 0x00000001	The query is in an error state.
STAT_DONE	The query is complete.

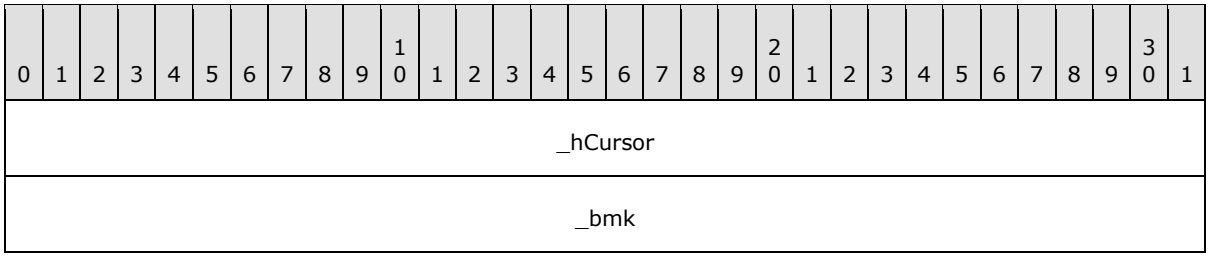
Constant	Meaning
0x00000002	
STAT_REFRESH 0x00000003	The query is complete, but updates are resulting in additional query computation.

The following table lists additional STAT_* bits that can be set independently.

Constant	Meaning
STAT_NOISE_WORDS 0x00000010	Noise words were replaced by wildcard characters in the content query.
STAT_CONTENT_OUT_OF_DATE 0x00000020	The results of the query might be incorrect because the query involved modified but un-indexed files.
STAT_CONTENT_QUERY_INCOMPLETE 0x00000080	The content query was too complex to complete or required enumeration instead of use of the content index.
STAT_TIME_LIMIT_EXCEEDED 0x00000100	The results of the query might be incorrect because the query execution reached the maximum allowable time.

2.2.3.10 CPMGetQueryStatusExIn

The CPMGetQueryStatusExIn message requests the status of a query and additional information, such as the number of documents that have been indexed or the number of documents remaining to be indexed. The format of the CPMGetQueryStatusExIn message that follows the header is shown in the following diagram.



_hCursor: A 32-bit value representing the handle from the CPMCreateQueryOut message identifying the query for which to retrieve status information.

_bmk: A 32-bit value indicating the handle of a bookmark whose position should be retrieved.

2.2.3.11 CPMGetQueryStatusExOut

The CPMGetQueryStatusExOut message replies to a CPMGetQueryStatusExIn message with both the status of the query and other status information, as outlined in the following diagram. The format of the CPMGetQueryStatusExOut 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
_Status																															
_cFilteredDocuments																															
_cDocumentsToFilter																															
_dwRatioFinishedDenominator																															
_dwRatioFinishedNumerator																															
_iRowBmk																															
_cRowsTotal																															
_maxRank																															
_cResultsFound																															

_Status: One of the STAT_* values specified in section 2.2.3.8.

_cFilteredDocuments: A 32-bit unsigned integer indicating the number of documents that have been indexed.

_cDocumentsToFilter: A 32-bit unsigned integer indicating the number of documents that remain to be indexed.

_dwRatioFinishedDenominator: A 32-bit unsigned integer indicating the denominator of the ratio of documents the query has finished processing.

_dwRatioFinishedNumerator: A 32-bit unsigned integer indicating the numerator of the ratio of documents the query has finished processing.

_iRowBmk: A 32-bit unsigned integer indicating the approximate position of the bookmark in the rowset in terms of rows.

_cRowsTotal: A 32-bit unsigned integer specifying the total number of rows in the rowset.

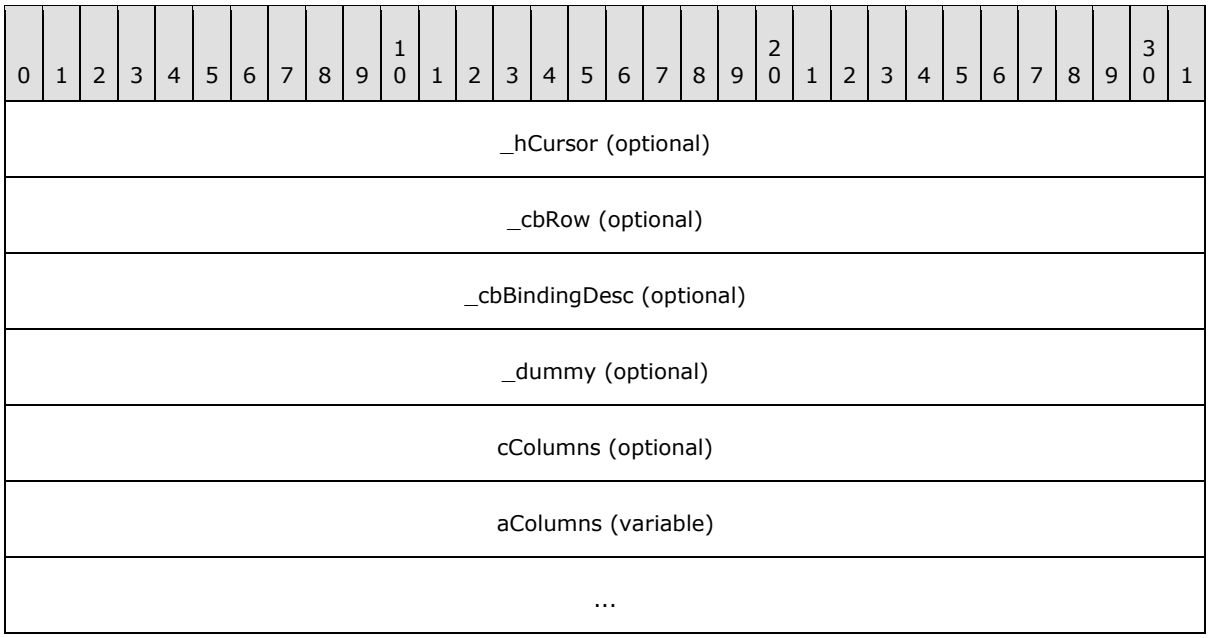
_maxRank: A 32-bit unsigned integer specifying the maximum rank found in the rowset.

_cResultsFound: A 32-bit unsigned integer specifying the number of unique results returned in the rowset.

2.2.3.12 CPMSetBindingsIn

The CPMSetBindingsIn message requests the **binding** of columns to a rowset. The server will reply to the CPMSetBindingsIn request message using the header section of the CPMSetBindingsIn

message with the results of the request contained in the **_status** field. The format of the CPMSetBindingsIn message that follows the header is shown in the following diagram.



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

_cbRow: 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 client and MUST be absent when the message is sent by the server.

_cbBindingDesc: 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 client and MUST be absent when the message is sent by the server.

_dummy: 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 client and MUST be absent when the message is sent by the server.

cColumns: 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 client and MUST be absent when the message is sent by the server.

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

2.2.3.13 CPMGetRowsIn

The CPMGetRowsIn message requests rows from a query. The format of the CPMGetRowsIn 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
_hCursor																															
_cRowsToTransfer																															
_cbRowWidth																															
_cbSeek																															
_cbReserved																															
_cbReadBuffer																															
_ulClientBase																															
_fBwdFetch																															
eType																															
_chapt																															
SeekDescription (variable)																															
...																															

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

_cRowsToTransfer: A 32-bit unsigned integer indicating the maximum number of rows the client wants to receive in response to this message.

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

_cbSeek: A 32-bit unsigned integer indicating the size of the message beginning with **eType**.

_cbReserved: A 32-bit unsigned integer indicating the size, in bytes, of a CPMGetRowsOut message (without the **Rows** and **SeekDescriptions** fields). This value in this field is added to the value of the **_cbSeek** field, and then is to be used to calculate the offset of **Rows** field in the CPMGetRowsOut message.

_cbReadBuffer: A 32-bit unsigned integer.

Note This field MUST be set to the maximum of the value of **_cbRowWidth** or 1000 times the value of **_cRowsToTransfer**, rounded up to the nearest 512 byte multiple. The value MUST NOT exceed 0x00004000.

_ulClientBase: 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 **reserved2** field of the message header is used as the upper 32-bits and _ulClientBase as the lower 32-bits of a 64-bit value. See section 2.2.3.14.

_fBwdFetch: A 32-bit unsigned integer indicating the order in which to fetch the rows that MUST be set to one of the following values:

Value	Meaning
0x00000000	The rows are to be fetched in forward order.
0x00000001	The rows are to be fetched in reverse order.

eType: A 32-bit unsigned integer that MUST contain one of the following values indicating the type of operation to perform.

Value	Meaning
eRowSeekNext 0x00000001	**SeekDescription** contains a CRowSeekNext structure.
eRowSeekAt 0x00000002	**SeekDescription** contains a CRowSeekAt structure.
eRowSeekAtRatio 0x00000003	**SeekDescription** contains a CRowSeekAtRatio structure.
eRowSeekByBookmark 0x00000004	**SeekDescription** contains a CRowSeekByBookmark structure.

_chapt: A 32-bit value representing the handle of the rowset chapter.

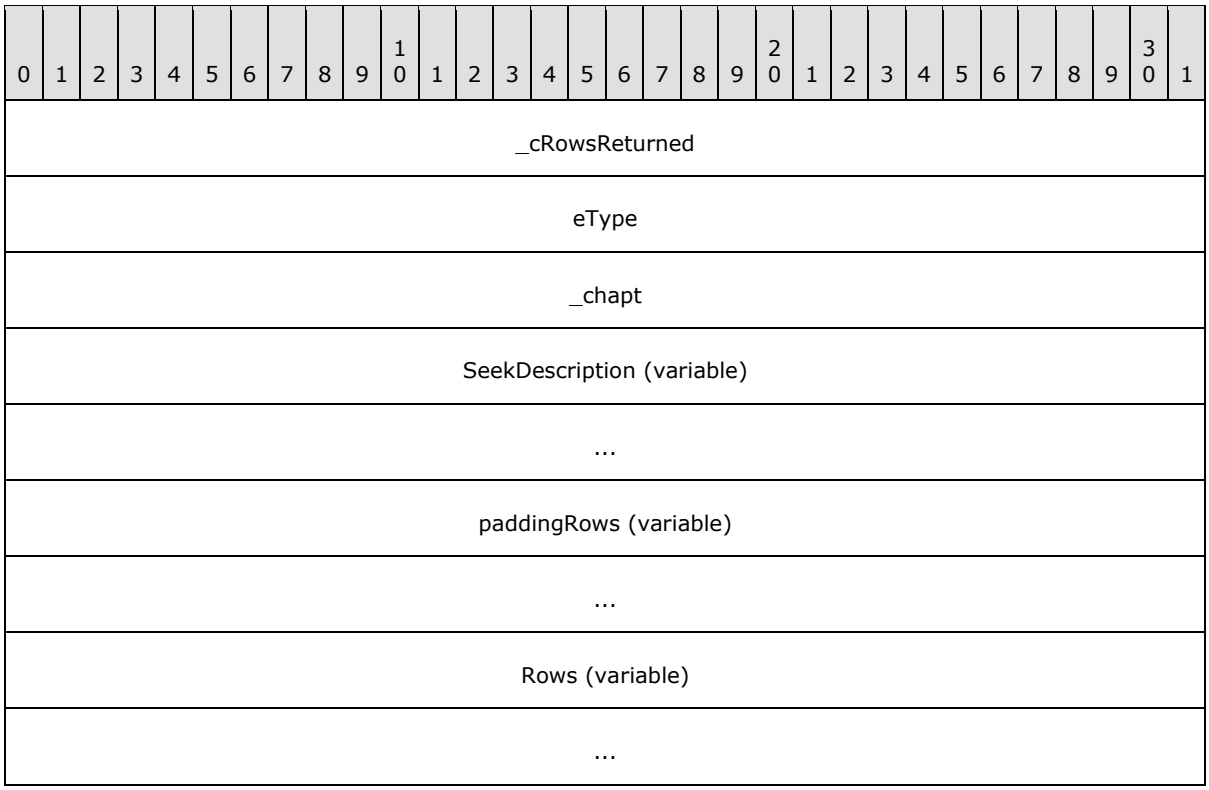
SeekDescription: This field MUST contain a structure of the type indicated by the **eType** value.

2.2.3.14 CPMGetRowsOut

The CPMGetRowsOut message replies to a CPMGetRowsIn message with the rows of a query. Servers MUST format offsets to variable length data types in the row field as follows:

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

The format of the CPMGetRowsOut message that follows the header is diagrammed below.



_cRowsReturned: A 32-bit unsigned integer indicating the number of rows returned in **Rows**.

eType: A 32-bit unsigned integer. MUST contain one of the following values indicating the point at which to begin retrieving rows.

Value	Meaning
eRowsSeekNone 0x00000000	No **SeekDescription**; ignore the **SeekDescription** field.
eRowSeekNext 0x00000001	**SeekDescription** contains a CRowSeekNext structure.
eRowSeekAt 0x00000002	**SeekDescription** contains a CRowSeekAt structure.
eRowSeekAtRatio 0x00000003	**SeekDescription** contains a CRowSeekAtRatio structure.
eRowSeekByBookmark 0x00000004	**SeekDescription** contains a CRowSeekByBookmark structure.

_chapt: A 32-bit value representing the handle of the rowset chapter.

SeekDescription: This field MUST contain a structure of the type indicated by the **eType** field.

paddingRows: This field MUST be of sufficient length (0 to _cbReserved-1 bytes) to pad the **Rows** field to _cbReserved offset from the beginning of a message, where _cbReserved is the

value in the CPMGetRowsIn message. Padding bytes used in this field can be any arbitrary value. This field MUST be ignored by the receiver.

Rows: Row data is formatted as prescribed by column information in the most recent CPMSetBindingsIn message. Rows MUST be stored in forward order (for example, row 1 before row 2).

Fixed-sized columns MUST be stored at the offsets specified by the most recent CPMSetBindingsIn message.

Variable-sized columns (for example, strings) MUST be stored as follows:

- The variable data itself (for example, the string) is stored near the end of the buffer in descending order (for example, the collection of all variable data for row 1 is at the end, row 2 next closest, and so on).
- The fixed sized area (at the beginning of the row buffer) MUST contain a CRowVariant for each column, stored at the offset specified in the most recent CPMSetBindingsIn message. **vType** MUST contain the data type (ex: VT_LPWSTR). If, as determined by the rules at the beginning of this section, 32-bit offsets are being used, the **Offset** field in CRowVariant MUST contain a 32-bit value that is the offset of the variable data from the beginning of the CPMGetRowsOut message, plus the value of **_ulClientBase** specified in the most recent CPMGetRowsIn message. If 64-bit offsets are being used, the **Offset** field in CRowVariant MUST contain a 64-bit value that is the offset from the beginning of the CPMGetRowsOut message, added to a 64-bit value composed by using **_ulClientBase** as the low 32-bits and **_ulReserved2** as the high 32-bits.

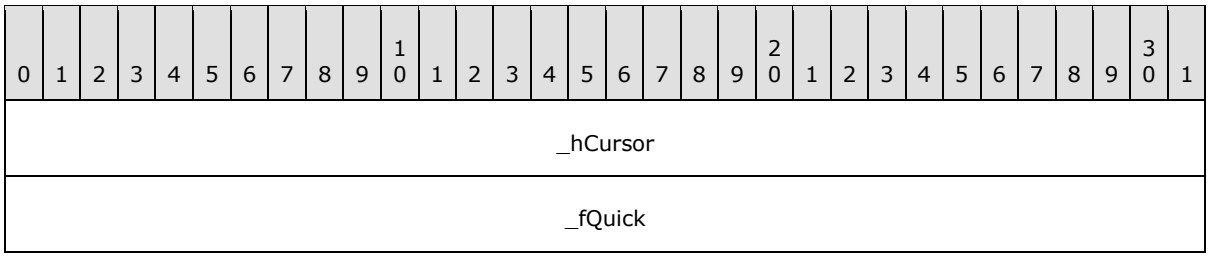
For example, if the CPMSetBindingsIn message specified two columns—Size (VT_I4) and Title (VT_LPWSTR)—and **_ulClientBase** from CPMGetRowsIn was 0x10000, two rows would appear as follows. The section marked in grey is the fixed-length part of the buffer.

<table>
<tr><td colspan="8">Header</td></tr>
<tr><td colspan="4">← Rows starts here</td><td colspan="4">Row 1 Size (8 bytes)</td></tr>
<tr><td>0x1F</td><td colspan="3">(reserved)</td><td colspan="4">0x10088</td></tr>
<tr><td colspan="4"></td><td colspan="4">Row 2 Size (8 bytes)</td></tr>
<tr><td>0x1F</td><td colspan="3">(reserved)</td><td colspan="4">0x1006E</td></tr>
<tr><td>0x00</td><td>0x00</td><td>0x00</td><td>S</td><td>E</td><td>C</td><td>O</td><td>N</td></tr>
<tr><td>D</td><td>S</td><td>T</td><td>R</td><td>I</td><td>N</td><td>G</td><td>0x00</td></tr>
<tr><td>S</td><td>T</td><td>R</td><td>I</td><td>N</td><td>G</td><td>1</td><td>0x00</td></tr>
</table>

Figure 2: Windows Search Protocol message header field offsets

2.2.3.15 CPMRatioFinishedIn

The CPMRatioFinishedIn message requests the completion percentage of a query. The format of the CPMRatioFinishedIn message that follows the header is shown in the following diagram.



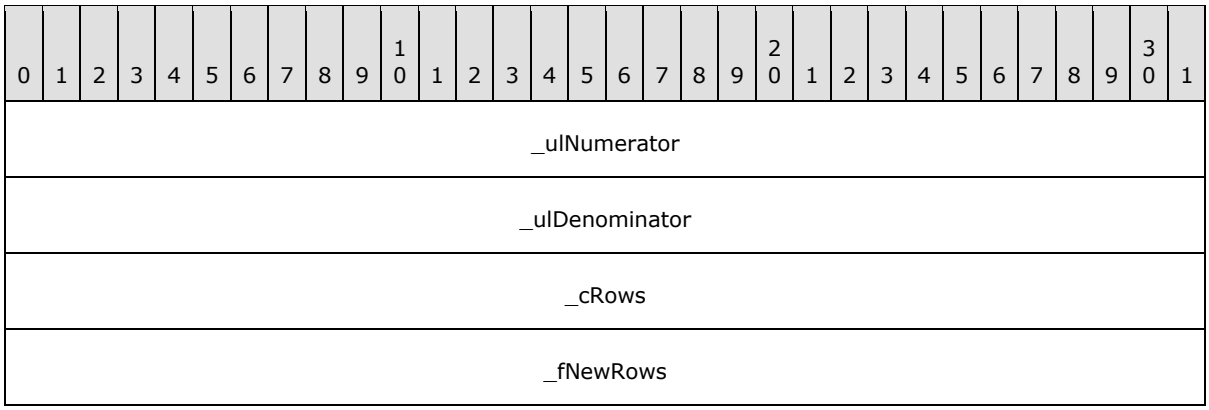
_hCursor: The handle from the CPMCreateQueryOut message identifying the query for which to request completion information.

_fQuick: This is unused and MUST be ignored by the server.

Note This field MUST be set to 0x00000001.

2.2.3.16 CPMRatioFinishedOut

The CPMRatioFinishedOut message replies to a CPMRatioFinishedIn message with the completion ratio of a query. The format of the CPMRatioFinishedOut message that follows the header is shown in the following diagram.



_ulNumerator: A 32-bit unsigned integer indicating the numerator of the completion ratio in terms of rows.

_ulDenominator: A 32-bit unsigned integer indicating the denominator of the completion ratio in terms of row. MUST be greater than zero.

_cRows: A 32-bit unsigned integer indicating the total number of rows for the query.

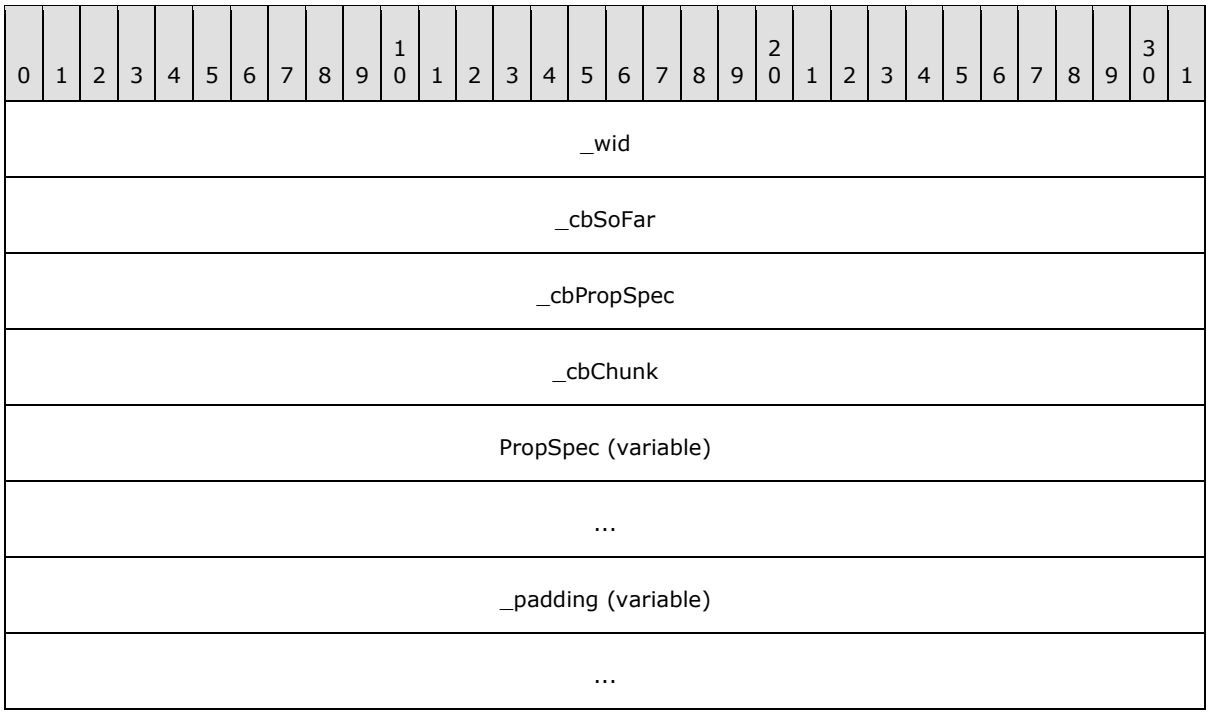
_fNewRows: A Boolean value indicating if there are new rows available. This field MUST NOT be set to any values other than the following.

Value	Meaning
0x00000000	There are no new rows in the rowset.
0x00000001	There are new rows available in the rowset.

2.2.3.17 CPMFetchValueIn

The CPMFetchValueIn message requests a property value that was too large to return in a rowset. As specified in section 3.2.4.2.5, this message is sent repeatedly to retrieve all bytes of the property, updating **_cbSoFar** for each, until the **_fMoreExists** field of the CPMFetchValueOut message is set to FALSE.

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



_wid: A 32-bit unsigned integer representing the document ID identifying the document for which a property should be fetched.

_cbSoFar: A 32-bit unsigned integer containing the number of bytes previously transferred for this property.

Note This field MUST be set to 0x00000000 in the first message.

_cbPropSpec: A 32-bit unsigned integer containing the size, in bytes, of the **PropSpec** field.

_cbChunk: A 32-bit unsigned integer containing the maximum number of bytes that the sender can accept in a CPMFetchValueOut message.<8>

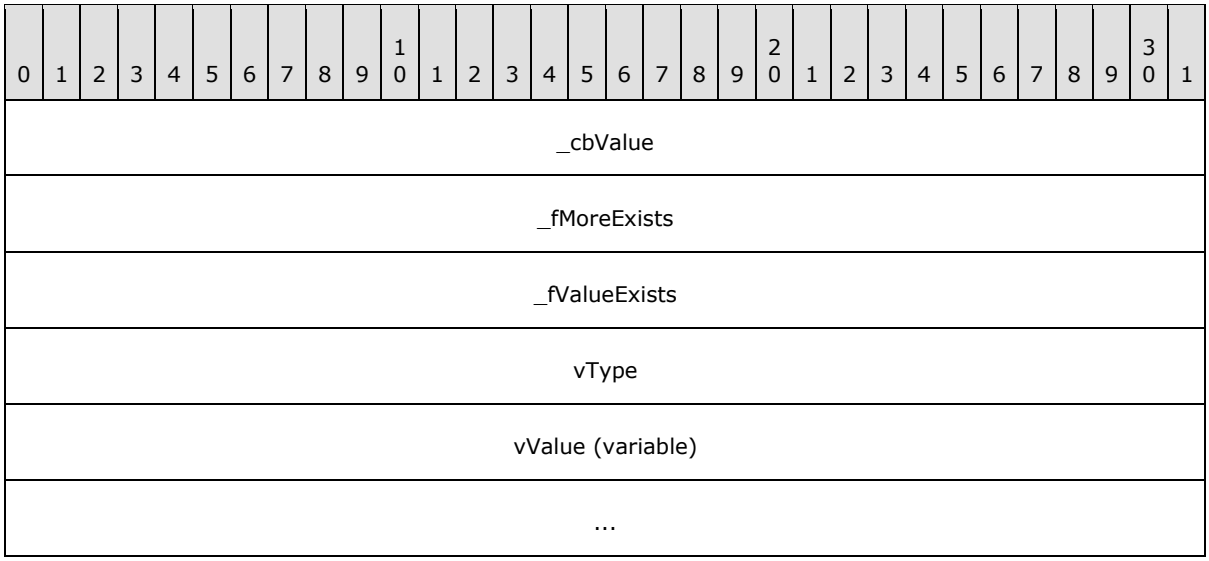
PropSpec: A CFullPropSpec structure specifying the property to retrieve.

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

2.2.3.18 CPMFetchValueOut

The CPMFetchValueOut message replies to a CPMFetchValueIn message with a property value from a previous query. As specified in section 3.2.4.2.5, this message is sent after each CPMFetchValueIn message until all bytes of the property are transferred.

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



_cbValue: A 32-bit unsigned integer containing the total size, in bytes in **vValue**.

_fMoreExists: A Boolean value indicating whether there are additional CPMFetchValueOut messages available.

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

_fValueExists: A Boolean value indicating whether there is a value for the property.

Value	Meaning
0x00000000	A value for the property does not exist.
0x00000001	A value for the property exists.

vType: One of the type values specified in section 2.2.1.1, describing the type of the property.

vValue: A portion of a byte array containing a SERIALIZEDPROPERTYVALUE, where the offset of the beginning of the portion is the value of **_cbSoFar** in CPMFetchValueIn. The length of the portion, indicated by the **_cbValue** field, MUST be equal to the value of **_cbChunk** in CPMFetchValueIn if **_fMoreExists** is set to 0x00000001. Otherwise the length of the portion MUST be less than or equal to the value of **_cbChunk** in CPMFetchValueIn.

2.2.3.19 CPMGetNotify

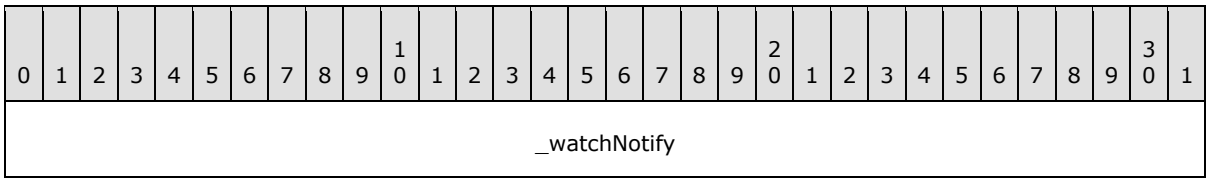
The CPMGetNotify message requests that the client wants to be notified of rowset changes.

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

2.2.3.20 CPMSendNotifyOut

The CPMSendNotifyOut message notifies the client of a change to the results of a query.

This message is only sent when a change occurs. The format of the CPMSendNotifyOut message that follows the header is shown in the following diagram.

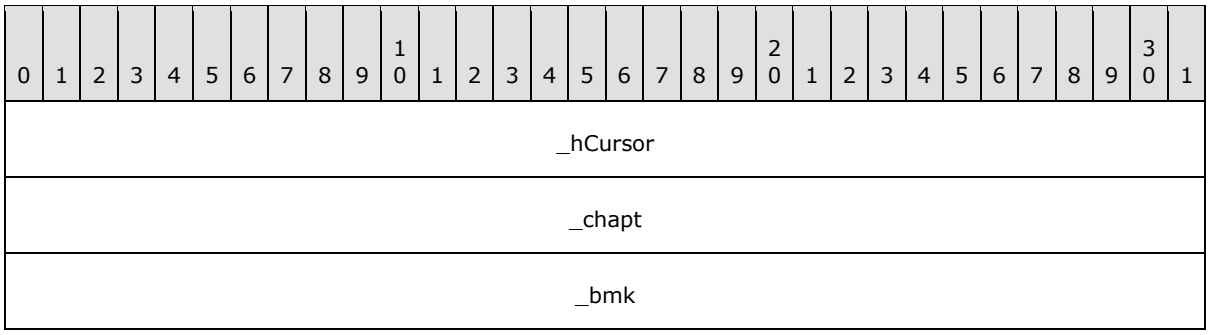


_watchNotify: A 32-bit unsigned integer representing the change to the query. It MUST be one of the following values.

Value	Meaning
DBWATCHNOTIFY_ROWSCHANGED 0x00000001	The number of rows in the query rowset has changed.
DBWATCHNOTIFY_QUERYDONE 0x00000002	The query has completed.
DBWATCHNOTIFY_QUERYREEXECUTED 0x00000003	The query has been executed again.

2.2.3.21 CPMGetApproximatePositionIn

The CPMGetApproximatePositionIn message requests the approximate position of a bookmark in a chapter. The format of the CPMGetApproximatePositionIn message that follows the header is shown in the following diagram.



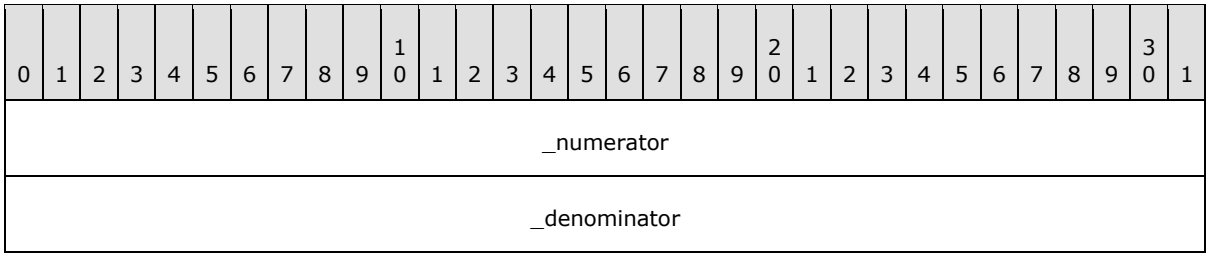
_hCursor: A 32-bit value representing the query cursor obtained from CPMCreateQueryOut for the rowset containing the bookmark.

_chapt: A 32-bit value representing the handle to the chapter containing the bookmark.

_bmk: A 32-bit value representing the handle to the bookmark for which to retrieve the approximate position.

2.2.3.22 CPMGetApproximatePositionOut

The CPMGetApproximatePositionOut message replies to a CPMGetApproximatePositionIn message describing the approximate position of the bookmark in the chapter. The format of the CPMGetApproximatePositionOut message that follows the header is shown in the following diagram.



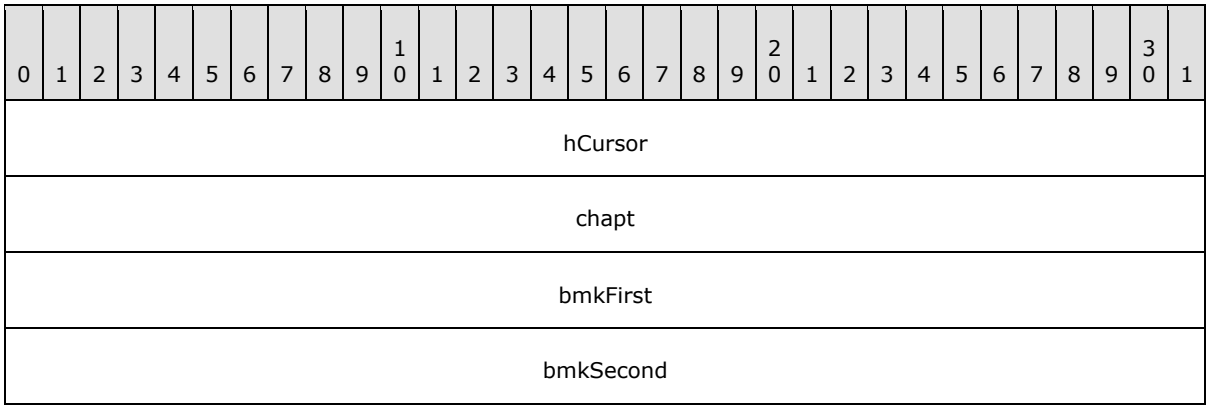
_numerator: A 32-bit unsigned integer containing the row number of the bookmark in the rowset. If there are no rows, this field MUST be set to 0x00000000.

_denominator: A 32-bit unsigned integer containing the number of rows in the rowset.

2.2.3.23 CPMCompareBmkIn

The CPMCompareBmkIn message requests a comparison of two bookmarks in a chapter.

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



hCursor: A 32-bit unsigned integer representing the handle from the CPMCreateQueryOut message for the rowset containing the bookmarks.

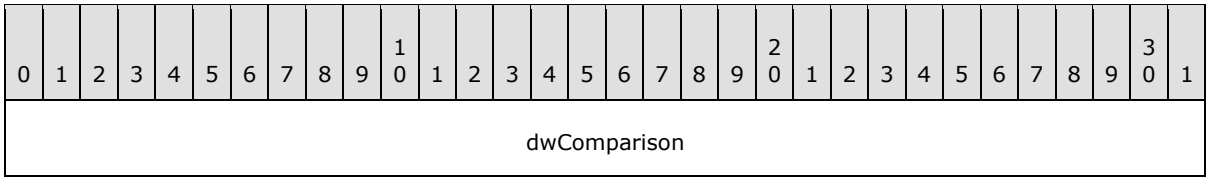
chapt: A 32-bit unsigned integer representing the handle of the chapter containing the bookmarks to compare.

bmkFirst: A 32-bit unsigned integer representing the handle to the first bookmark to compare.

bmkSecond: A 32-bit unsigned integer representing the handle to the second bookmark to compare.

2.2.3.24 CPMCompareBmkOut

The CPMCompareBmkOut message replies to a CPMCompareBmkIn message with the comparison of the two bookmarks in the chapter. The format of the CPMCompareBmkOut message that follows the header is shown in the following diagram.



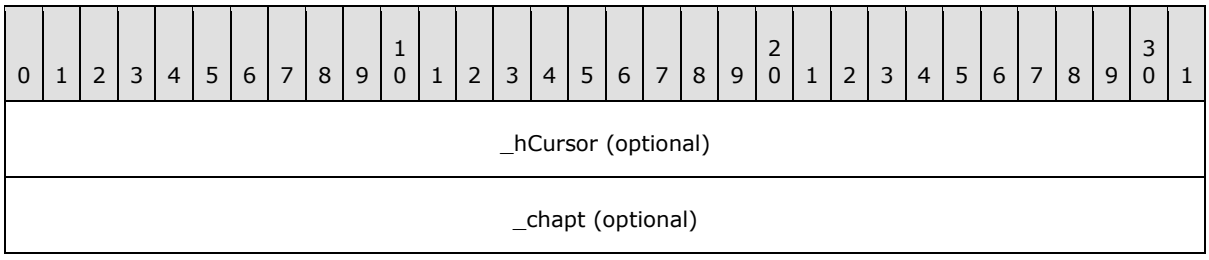
dwComparison: A 32-bit unsigned integer. MUST be one of the following values, indicating the relative positions of the two bookmarks in the chapter.

Value	Meaning
DBCOMPARE_LT 0x00000000	The first bookmark is positioned before the second.
DBCOMPARE_EQ 0x00000001	The first bookmark has the same position as the second.
DBCOMPARE_GT 0x00000002	The first bookmark is positioned after the second.
DBCOMPARE_NE 0x00000003	The first bookmark does not have the same position as the second.
DBCOMPARE_NOTCOMPARABLE 0x00000004	The first bookmark is not comparable to the second.

2.2.3.25 CPMRestartPositionIn

The CPMRestartPositionIn message moves the fetch position for a cursor to the beginning of the chapter. As specified in section 3.1.5.2.12, the server will reply using the same message, with the results of the request contained in the **_status** field of the Windows Search Protocol header.

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



_hCursor: A 32-bit value that represents the handle obtained from a CPMCreateQueryOut message and that identifies the query for which to restart the position. This field MUST be

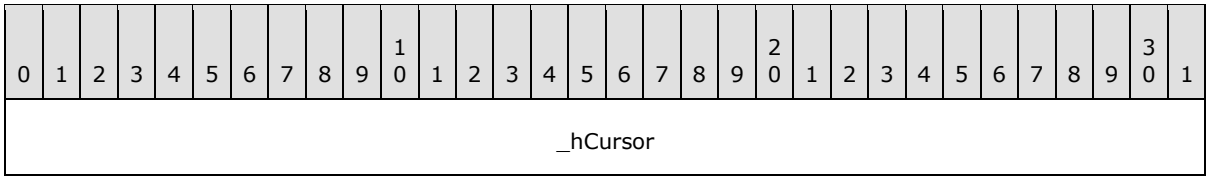
present when the message is sent by the client and MUST be absent when the message is sent by the server.

_chapt: A 32-bit value representing the handle of a chapter from which to retrieve rows. This field MUST be present when the message is sent by the client and MUST be absent when the message is sent by the server.

2.2.3.26 CPMStopAsynchIn

The CPMStopAsynchIn message contains a cursor handle for which an asynchronous query should be stopped.

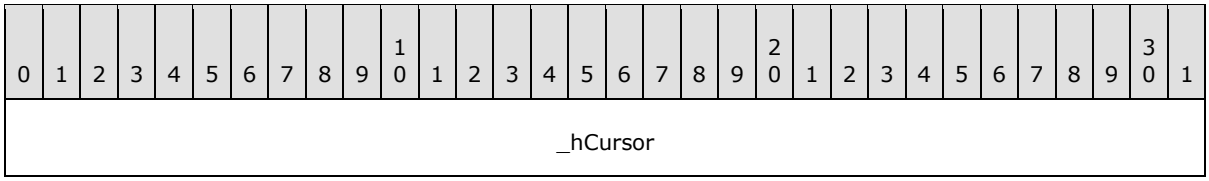
The format of the CPMStopAsynchIn message that follows the header is shown in the following diagram.



_hCursor: The handle from the CPMCreateQueryOut message identifying the query for which to stop asynchronous processing.

2.2.3.27 CPMFreeCursorIn

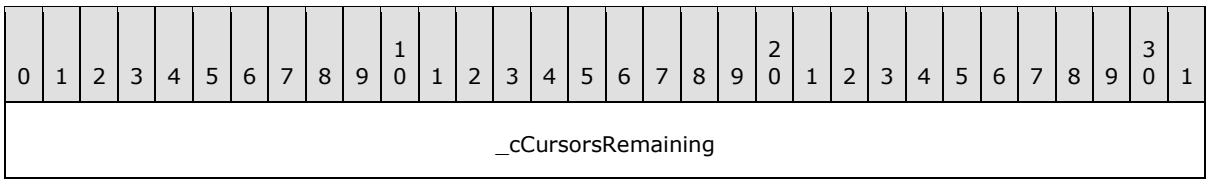
The CPMFreeCursorIn message requests the release of a cursor. The format of the CPMFreeCursorIn message that follows the header is shown in the following diagram.



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

2.2.3.28 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: A 32-bit unsigned integer indicating the number of cursors still in use for the query.

2.2.3.29 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 Windows Search Protocol messages MUST return 0x00000000 to indicate success; otherwise, they return a 32-bit nonzero error code that can either be an HRESULT or an NTSTATUS value (see section 1.8). If a buffer is too small to fit a result, a status code of STATUS_INSUFFICIENT_RESOURCES (0xC0000009A) MUST be returned, and the failing operation should be retried with a larger buffer.

All other error values MUST be treated the same; the error MUST be considered fatal and reported to the higher-level caller. Future messages MAY be sent over the same pipe as if no error had occurred.<9>

Note The HRESULT and NTSTATUS numbering spaces do not currently overlap—except with values of identical meaning. However, even if there were to be conflicts in the future, it would not cause any protocol issues as long as the value for STATUS_INSUFFICIENT_RESOURCES remains unique, because all other error values are treated the same.

2.2.5 Standard Properties

Properties in the Windows Search service are represented by the combination of a property set **GUID** and either a string property name or an integer property ID. For details, see CFullPropSpec.

There are three classes of properties: database properties, query properties, and open properties. Database properties help control the indexing service behavior and are as specified in section 2.2.1.15.1. Query properties can be used in a restriction and in some cases returned with every result. They are special because they are built into the Windows Search service. Open properties are defined by individual applications. There are a typical set of common properties in use, but there is no requirement to use them.

2.2.5.1 Query Properties

Query Property Set

```
#define  QueryGuid     \
{0x49691c90,0x7e17,0x101a,0xa9,0x1c,0x08,0x00,0x2b,0x2e,0xcd,0xa9}
```

Friendly name/PropId	Datatype	Description
RankVector 0x00000002	VT_UI4|VT_VECTOR	The 0-1000 rank computed for each element when performing vector ranking.
Rank 0x00000003	VT_I4	The rank 0-1000 computed for this item. How rank is computed is defined by the indexing service. Typically, content and proximity restrictions influence the rank, while other comparison operators do not.

Friendly name/PropId	Datatype	Description
HitCount 0x00000004	VT_I4	The number of words from the query found.
WorkId 0x00000005	VT_I4	A unique identifier for each result found.
All 0x00000006	VT_LPWSTR	Allows a content restriction over all textual properties. Cannot be retrieved.
VPath 0x00000009	VT_LPWSTR	Full virtual path to file, including file name. If there is more than one possible path, then the best match for the specific query is chosen.

Storage Property Set

```
#define  StorageGuid     \
{0xb725f130,0x47ef,0x101a,0xa5,0xf1,0x02,0x60,0x8c,0x9e,0xeb,0xac}
```

Friendly name/PropId	Datatype	Description
Contents 0x00000013	VT_LPWSTR	Main contents of a file. Usually this property cannot be retrieved.

2.2.5.2 Common Open Properties

A Windows Search service can allow querying and retrieval over any property. The tables below outline some properties typically used.

Storage Property Set

```
#define  StorageGuid     \
{0xb725f130,0x47ef,0x101a,0xa5,0xf1,0x02,0x60,0x8c,0x9e,0xeb,0xac}
```

Friendly name/PropId	Datatype	Description
Directory 0x00000002	VT_LPWSTR	The physical path to the file, not including the file name.
ClassId 0x00000003	VT_CLSID	The class ID of the object, for example, WordPerfect or Word.
FileIndex 0x00000008	VT_UI8	The unique ID of the file.
USN 0x00000009	VT_I8	The update sequence number. NTFS drives only.

Friendly name/PropId	Datatype	Description
Filename 0x0000000A	VT_LPWSTR	The name of the file.
Path 0x0000000B	VT_LPWSTR	The full physical path to the file, including the file name.
Size 0x0000000C	VT_I8	The size, in bytes, of the file.
Attrib 0x0000000D	VT_UI4	The file attributes. Documented in Win32 SDK.
Write 0x0000000E	VT_FILETIME	The last time the file was written.
Create 0x0000000F	VT_FILETIME	The time the file was created.
Access 0x00000010	VT_FILETIME	The last time the file was accessed.
AllocSize 0x00000012	VT_I8	The size of the disk allocation for the file.
ShortFilename 0x00000014	VT_LPWSTR	The short (8.3) file name.

The following table lists the attribute flag values for the Attrib property.

Attribute/Value	Description
FILE_ATTRIBUTE_READONLY 0x00000001	The file or directory is read-only. Applications can read the file but cannot write to it or delete it. For a directory, applications cannot delete it.
FILE_ATTRIBUTE_HIDDEN 0x00000002	The file or directory is hidden. It is not included in an ordinary directory listing.
FILE_ATTRIBUTE_SYSTEM 0x00000004	The file or directory is part of the operating system, or is used exclusively by the operating system.
FILE_ATTRIBUTE_DIRECTORY 0x00000010	The handle identifies a directory.
FILE_ATTRIBUTE_ARCHIVE 0x00000020	The file or directory is an archive file. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_NORMAL 0x00000080	The file or directory does not have another attributes set. This attribute is valid only if used alone.
FILE_ATTRIBUTE_TEMPORARY 0x00000100	The file is being used for temporary storage. File systems avoid writing data back to mass storage if sufficient

Attribute/Value	Description
	cache memory is available, because often the application deletes the temporary file shortly after the handle is closed. In that case, the system can entirely avoid writing the data. Otherwise, the data is written after the handle is closed.
FILE_ATTRIBUTE_SPARSE_FILE 0x00000200	The file is a sparse file.
FILE_ATTRIBUTE_REPARSE_POINT 0x00000400	The file or directory has an associated reparse point.
FILE_ATTRIBUTE_COMPRESSED 0x00000800	The file or directory is compressed. For a file, this means that all the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_OFFLINE 0x00001000	The data of the file is not immediately available. This attribute indicates that the file data has been physically moved to offline storage. This attribute is used by Remote Storage, the hierarchical storage management software. Applications should not arbitrarily change this attribute.
FILE_ATTRIBUTE_ENCRYPTED 0x00004000	The file or directory is encrypted. For a file, this means that all data in the file is encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_VIRTUAL 0x00010000	A file is a virtual file.

Property Sets for Documents

```
#define  DocPropSetGuid      \
{0xf29f85e0,0x4ff9,0x1068,0xab,0x91,0x08,0x00,0x2b,0x27,0xb3,0xd9}
```

Friendly name/PropId	Datatype	Description
DocTitle 0x00000002	VT_LPWSTR	The title of the document.
DocSubject 0x00000003	VT_LPWSTR	The subject of the document.
DocAuthor 0x00000004	VT_LPWSTR	The author of the document.
DocKeywords 0x00000005	VT_LPWSTR	The document keywords.
DocComments	VT_LPWSTR	Comments about the document.

Friendly name/PropId	Datatype	Description
0x00000006		
DocTemplate 0x00000007	VT_LPWSTR	The name of the template for the document.
DocLastAuthor 0x00000008	VT_LPWSTR	The most recent user who edited document.
DocRevNumber 0x00000009	VT_LPWSTR	The current version number of the document.
DocEditTime 0x0000000A	VT_FILETIME	The total time spent editing the document.
DocLastPrinted 0x0000000B	VT_FILETIME	The time the document was last printed.
DocCreatedTm 0x0000000C	VT_FILETIME	The time the document was created.
DocLastSavedTm 0x0000000D	VT_FILETIME	The time the document was last saved.
DocPageCount 0x0000000E	VT_I4	The number of pages in the document.
DocWordCount 0x0000000F	VT_I4	The number of words in the document.
DocCharCount 0x00000010	VT_I4	The number of characters in the document.
DocThumbnail 0x00000011	VT_CF	A thumbnail of the document in clipboard format.
DocAppName 0x00000012	VT_LPWSTR	The name of the application that created the file.

Property Sets for Documents

```
#define  DocPropSetGuid2     \
{0xd5cdd502,0x2e9c,0x101b,0x93,0x97,0x08,0x00,0x2b,0x2c,0xf9,0xae}
```

Friendly name/PropId	Datatype	Description
DocCategory 0x00000002	VT_LPSTR	The type of document, such as a memo, schedule, or white paper.
DocPresentationTarget 0x00000003	VT_LPSTR	The target format (for example, 35mm, printer, or video) for a presentation in PowerPoint.

Friendly name/PropId	Datatype	Description
DocByteCount 0x00000004	VT_I4	The number of bytes in a document.
DocLineCount 0x00000005	VT_I4	The number of lines contained in a document.
DocParaCount 0x00000006	VT_I4	The number of paragraphs in a document.
DocSlideCount 0x00000007	VT_I4	The number of slides in a PowerPoint document.
DocNoteCount 0x00000008	VT_I4	The number of pages with notes in a PowerPoint document.
DocHiddenCount 0x00000009	VT_I4	The number of hidden slides in a Microsoft PowerPoint document.
DocPartTitles 0x0000000D	VT_LPWSTR|VT_VECTOR	The names of document parts. For example, in Excel, part titles are the names of spread sheets, in PowerPoint slide titles, and in Word for Windows, the names of the documents in the master document.
DocManager 0x0000000E	VT_LPSTR	The name of the manager of the document's author.
DocCompany 0x0000000F	VT_LPSTR	The name of the company for which the document was written.

Document characterization

```
#define  DocCharacterGuid     \
{0x560c36c0,0x503a,0x11cf,0xba,0xa1,0x00,0x00,0x4c,0x75,0x2a,0x9a}
```

Friendly name/PropId	Datatype	Description
Characterization 0x00000002	VT_LPWSTR	A characterization or abstract of the document. Computed by Indexing Service.

Music Property Set

```
#define PSGUID_MUSIC     \
{56A3372E-CE9C-11d2-9F0E-006097C686F6}
```

Friendly name/PropId	Datatype	Description
MusicArtist 0x00000002	VT_LPWSTR	The artist who recorded the song.
MusicAlbum 0x00000004	VT_LPWSTR	The album that the song was released on.
MusicYear 0x00000005	VT_LPWSTR	The year that the song was published.
MusicTrack 0x00000007	VT_UI4	The track number for the song.
MusicGenre 0x0000000B	VT_LPWSTR	The song's genre.

Digital Rights Management

```
#define PSGUID_DRM     \
{AEAC19E4-89AE-4508-B9B7-BB867ABEE2ED}
```

This property set contains properties that describe the digital rights associated with some media.

Friendly name/PropId	Datatype	Description
DrmLicense 0x00000002	VT_BOOL	TRUE if there is a license.
DrmDescription 0x00000003	VT_LPWSTR	The description of the license.
DrmPlayCount 0x00000004	VT_UI4	The number of times you can play the item.
DrmPlayStarts 0x00000005	VT_FILETIME	The date the play rights start.
DrmPlayExpires 0x00000006	VT_FILETIME	The date the rights expire.

Image Property Set

```
#define PSGUID_IMAGESUMMARYINFORMATION     \
{0x6444048fL,0x4c8b,0x11d1,0x8b,0x70,0x8,0x00,0x36,0xb1,0x1a,0x03}
```

Friendly name/PropId	Datatype	Description
ImageFileType 0x00000002	VT_LPWSTR	The type of image file.

Friendly name/PropId	Datatype	Description
ImageCx 0x00000003	VT_UI4	The horizontal size, in pixels.
ImageCy 0x00000004	VT_UI4	The vertical size, in pixels.
ImageResolutionX 0x00000005	VT_UI4	The horizontal resolution, in pixels per inch.
ImageResolutionY 0x00000006	VT_UI4	The vertical resolution, in pixels per inch.
ImageBitDepth 0x00000007	VT_UI4	The number of bits per pixel.
ImageColorSpace 0x00000008	VT_LPWSTR	The description of the image color space.
ImageCompression 0x00000009	VT_LPWSTR	The description of the image compression.
ImageTransparency 0x0000000A	VT_UI4	The degree of transparency from 0-100.
ImageGammaValue 0x0000000B	VT_UI4	The gamma correction value.
ImageFrameCount 0x0000000C	VT_UI4	The frame count for image.
ImageDimensions 0x0000000D	VT_LPWSTR	The description of the image dimensions.

Audio Property Set

```
#define PSGUID_AUDIO     \
{64440490-4C8B-11D1-8B70-080036B11A03}
```

Audio-Related Properties

Friendly name/PropId	Datatype	Description
AudioFormat 0x00000002	VT_LPWSTR	The audio format.
AudioTimeLength 0x00000003	VT_UI8	The duration, in 100 ns units.
AudioAvgDataRate 0x00000004	VT_UI4	The average encoding rate, in bits per second.

Friendly name/PropId	Datatype	Description
AudioSampleRate 0x00000005	VT_UI4	The sample rate, in samples per second.
AudioSampleSize 0x00000006	VT_UI4	The sample size, in bits per sample.
AudioChannelCount 0x00000007	VT_UI4	The number of channels of audio.

Video Property Set

```
#define PSGUID_VIDEO     \
{64440491-4C8B-11D1-8B70-080036B11A03}
```

Friendly name/PropId	Datatype	Description
VideoStreamName 0x00000002	VT_LPWSTR	The name of the stream.
VideoFrameWidth 0x00000003	VT_UI4	The width, in pixels, of a frame.
VideoFrameHeight 0x00000004	VT_UI4	The height, in pixels, of a frame.
VideoTimeLength 0x00000005	VT_UI4	The duration, in 100 ns units.
VideoFrameCount 0x00000006	VT_UI4	The number of frames in video.
VideoFrameRate 0x00000007	VT_UI4	The frames per second.
VideoDataRate 0x00000008	VT_UI4	The bits per second.
VideoSampleSize 0x00000009	VT_UI4	The bits per sample.
VideoCompression 0x0000000A	VT_LPWSTR	The description of video compression.

Mime Properties

```
#define NNTPGuid     \
{0xAA568EEC,0xE0E5,0x11CF,0x8F,0xDA,0x00,0xAA,0x00,0xA1,0x4F,0x93}
```

Friendly name/PropId	Datatype	Description
MsgNewsgroup 0x00000002	VT_LPWSTR	The newsgroup for the message.
MsgSubject 0x00000005	VT_LPWSTR	The subject of the message.
MsgFrom 0x00000006	VT_LPWSTR	Who sent the message.
MsgMessageID 0x00000007	VT_LPWSTR	The unique ID for e-mail.
MsgDate 0x0000000C	VT_FILETIME	When the message was sent.
MsgReceivedDate 0x00000035	VT_FILETIME	When the message was received.
MsgArticleID 0x0000003C	VT_UI4	The unique identifier for the newsgroup article.

3 Protocol Details

The Windows Search Protocol message requests require only minimal sequencing. All messages MUST be preceded by an initial CPMConnectIn message (for example, at least one CPMConnectIn for each named pipe connection). Beyond the initial connection there is no other sequencing required by the protocol. It is advised that the higher layer adhere to a meaningful message sequence though, as the server will respond with an error for messages that are received out of this sequence or with invalid data. 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 query from a client to a remote computer is illustrated in the following diagram.

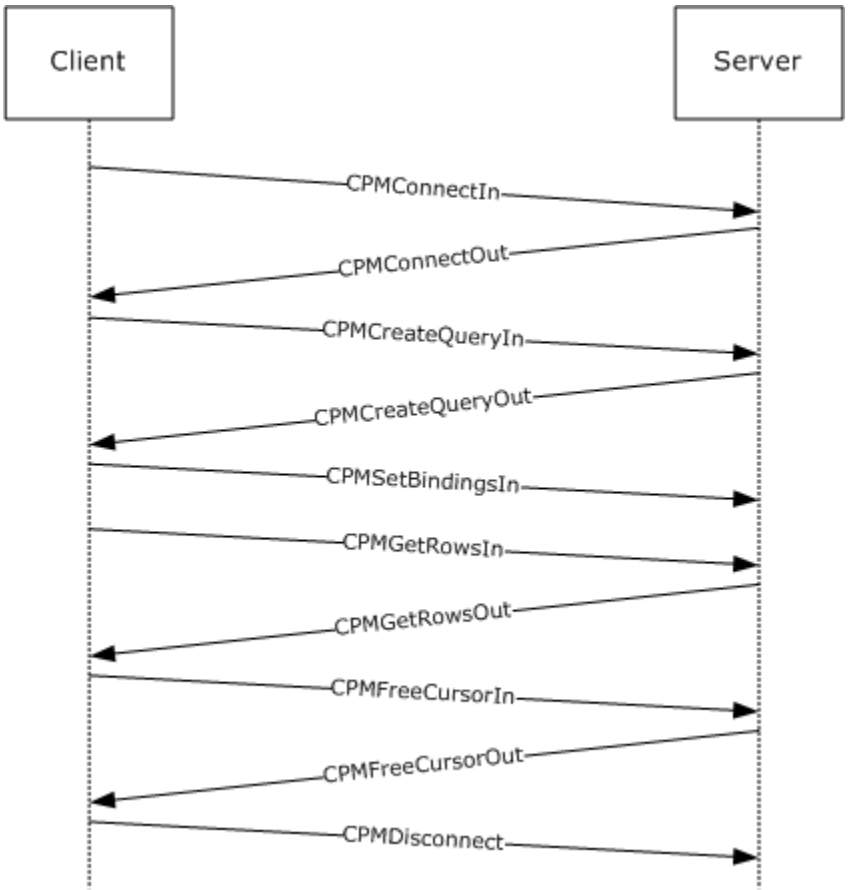


Figure 3: Windows Search Protocol session life cycle

The messages represented in the preceding diagram represent a subset of all of the Windows Search Protocol messages used for querying a remote Windows Search service catalog.

3.1 Server Details

3.1.1 Abstract Data Model

The following section specifies data and state maintained by the Windows Search 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 Windows Search service implementing the Windows Search Protocol MUST maintain the following abstract data elements:

- The list of clients connected to the server.
- Information about each client, which includes:
  - Client's version (as indicated in the CPMConnectIn message).
  - Catalog associated with the client (by a CPMConnectIn message).
  - Additional client properties, as specified in section 2.2.1.31.1.
  - Client's search query.
  - List of cursor handles for the query and position in result set for each handle.
  - Current set of bindings.
  - Current status of the query, which includes (for each cursor):
    - Number of rows in query result.
    - Numerator and denominator of query completion.
    - Last number of rows, reported by most recent CPMRatioFinishedOut message for this cursor.
    - Whether the query is monitored by the server for changes in query results, and, if it is monitored, what changed in the query results since it was last reported to the client by CPMSendNotifyOut.
    - List of chapter handles, served by this query to a client.
    - List of bookmark handles for each cursor, served by this query to a client.
- The current state of the Windows Search service, that can be "not initialized", "running", or "shutting down". Note that most of the time the server is in the "running" state. The following is the state machine diagram for the server.

Figure 4: Windows Search service state diagram (server perspective)

- Per-catalog information: number of documents indexed, size of inverted index, number of unique keys, and so on (see section 2.2.3.1 for a complete list).
- For each language supported, a database of alternate word forms including the base word, prefixes, and inflected forms as described for **_ulGenerateMethod** in section 2.2.1.3.

3.1.2 Timers

No timers are required for this protocol.

3.1.3 Initialization

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

3.1.4 Higher-Layer Triggered Events

There are no higher-layer triggered events for this protocol.

3.1.5 Message Processing and Sequencing Rules

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

- Stop processing the message sent by the client.
- Respond with the message header (only) of the message sent by the client, keeping the **_msg** field intact.
- Set the **_status** field to the error code value.

When a message arrives, the server MUST check the **_msg** field value to see if it is a known type (see section 2.2.2). If the type is not known, it MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

The 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)

To validate the **_ulChecksum** field value, the server MUST check the value the client specified in the **_iClientVersion** field in the CPMConnectIn message.

If the **_iClientVersion** field is not set to 0x00000102, and the **_ulChecksum** field is not set to 0x00000000, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error. The server MUST NOT validate the **_ulChecksum** field for clients that set the **_iClientVersion** field to a value less than 0x00000008.

If the **_iClientVersionfield** field value is 0x00000102 or greater, the server MUST validate that the **_ulChecksum** field was calculated as specified in section 3.2.4. If the **_ulChecksum** value is invalid, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

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

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

Some messages are only valid after a previous message has been sent. Typically, an ID or handle from the earlier message is required as input to the later message. These requirements are detailed in the following sections. The table below summarizes the relationship between messages.

	Prerequisite Message																
Message	CPMCiStateInOut	CPMUpdateDocumentsIn	CPMForceMergeIn	CPMConnectOut	CPMCreateQueryOut	CPMGetQueryStatusOut	CPMGetQueryStatusExOut	CPMSetBindingsOut	CPMGetRowsOut	CPMRatioFinishedOut	CPMFetchValueOut	CPMGetNotifyOut	CPMSendNotifyOut	CPMGetApproximatePositionOut	CPMCompareBmkOut	CPMRestartPositionOut	CPMFreeCursorOut
CPMCiStateInOut				X													
CPMUpdateDocumentsIn				X													
CPMForceMergeIn				X													
CPMConnectIn																	
CPMCreateQueryIn				X													
CPMGetQueryStatusIn				X	X												
CPMGetQueryStatusExIn				X	X												
CPMSetBindingsIn				X	X												
CPMGetRowsIn				X	X		X										
CPMRatioFinishedIn				X	X												
CPMFetchValueIn				X	X		X	X									
CPMGetNotifyIn				X	X												
CPMSendNotifyIn				X	X						X						
CPMGetApproximatePositionIn				X	X												
CPMCompareBmkIn				X	X												
CPMRestartPositionIn				X	X												
CPMFreeCursorIn				X	X												

Figure 5: Windows Search Protocol message sequence relationships

3.1.5.1 Remote Windows Search Service Catalog Management

3.1.5.1.1 Receiving a CPMCiStateInOut Request

When the server receives a CPMCIStateInOut message request from the client, the server MUST first check if the client is in a list of connected clients. If the client is not in the list, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error. Otherwise it MUST respond to the client with a CPMCIStateInOut message, containing information about the client's associated catalog, as specified in section 2.2.3.1.

3.1.5.1.2 Receiving a CPMUpdateDocumentsIn Request

When the server receives a CPMUpdateDocumentsIn message request, the server MUST do the following:

1. Check whether the client is in a list of connected clients (which have a catalog associated). If the client is not in the list, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check whether the client has administrative access. If the client does not have administrative access to the server, the server MUST report a STATUS_ACCESS_DENIED (0xC0000022) error.

3. Begin the process of indexing the path specified by the client, doing a full or incremental scan, depending on the value of the **_flag** field in the CPMUpdateDocumentsIn message. If the path was not previously indexed, it MUST be added to the collection of indexed locations and a full scan performed. If an illegal value of the **_flag** field is specified, the server MUST act as if **_flag** was set to UPD_INIT and perform a full scan. This operation MUST be performed in the catalog associated with the client.

4. Respond to the client with the message header for the CPMUpdateDocumentsIn, and set the **_status** field to the results of the request.

3.1.5.1.3 Receiving a CPMForceMergeIn Request

When the server receives a CPMForceMergeIn message request, the server MUST do the following:

1. Check if the client is in a list of connected clients (which have a catalog associated). If the client is not in the list, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.

2. Check that client has administrative access. If the client does not have administrative access, the server MUST report a STATUS_ACCESS_DENIED (0xC0000022) error.

3. Begin any process of maintenance necessary to improve query performance on a catalog associated with the client.

4. Respond to the client with a message header for the CPMForceMergeIn, and set the **_status** field to the results of the request.

   Note The process of maintenance is asynchronous and can continue after the client receives the response message. This process does not directly impact the protocol in any way (other than response time).

3.1.5.2 Remote Windows Search Service Querying

3.1.5.2.1 Receiving a CPMConnectIn Request

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

1. Check if the client is in the list of connected clients. If this is the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Checks if the specified catalog exists and is not in the "stopped" state. If this is not the case, the server MUST report a CI_E_NO_CATALOG (0x8004181D) error.
3. Add the client to the list of connected clients.
4. Associate the catalog with the client.
5. Store the information passed in the CPMConnectIn message (such as catalog name or client version) in the client state.
6. Respond to the client with a CPMConnectOut message.

3.1.5.2.2 Receiving a CPMCreateQueryIn Request

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

1. Check if the client is in the list of connected clients. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Check if the client already has a query associated with it. If this is the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
3. Check if the catalog associated with the client is in a state that allows the query to be processed (CICAT_READONLY or CICAT_WRITABLE). If this is not the case, the server MUST report a QUERY_S_NO_QUERY (0x8004160C) error.
4. Parse the restriction set, sort orders, and groupings that are specified in the query. If the server encounters an error, it MUST report an appropriate error. If this step fails for any other reason, the server MUST report the error encountered. For information about Windows Search service query errors, see [MSDN-QUERYERR].
5. Save the search query in the state for the client.
6. Make any preparations needed to serve rows to a client and associate the query with appropriate cursor handles (depending on information passed in the CPMCreateQueryIn message).
7. Add those handles to the client's list of cursor handles, and initialize lists of chapter handles and bookmarks.
8. Initialize the list of chapter handles for every cursor in this query to DB_NULL_HCHAPTER.
9. Initialize the list of bookmark handles for every cursor in this query to a set of DBBMK_FIRST and DBBMK_LAST.
10. Mark the query as not monitored for changes.

11. Initialize the number of rows to the currently calculated number of rows (which can be 0 if the query did not start to execute, or a number if the query is in the process of execution), and initialize the numerator and denominator of a query-completion ratio used to indicate the completion percentage of the query.

12. Respond to the client with a CPMCreateQueryOut message.

3.1.5.2.3 Receiving a CPMGetQueryStatusIn Request

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

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

2. Check whether the cursor handle is in a list of the client's cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Prepare a CPMGetQueryStatusOut message. The server MUST retrieve the current query status and set it in the **_QStatus** field (see section 2.2.3.9 for possible values). If this step fails for any reason, the server MUST report an error.

4. Respond to the client with the CPMGetQueryStatusOut message.

3.1.5.2.4 Receiving a CPMGetQueryStatusExIn Request

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

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

2. Check whether the cursor handle passed is in a list of the client's cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Prepare a CPMGetQueryStatusExOut message. The server MUST retrieve the current query status and query progress and set QStatus (see section 2.2.3.9 for possible values), **_dwRatioFinishedDenominator** and **_dwRatioFinishedNumerator** respectively. It MUST then set the number of rows in the query results to **_cRowsTotal**. If this step fails for any reason, the server MUST report that an error was encountered.

4. Retrieve information about the client's catalog and fill in **_cFilteredDocuments** and **_cDocumentsToFilter**. If this step fails for any reason, the server MUST report that an error was encountered.

5. Retrieve the position of the bookmark indicated by the handle in the **_bmk** field, and fill in the remaining **_iRowBmk** field in the CPMGetQueryStatusExOut message. If this step fails for any reason, the server MUST report that an error was encountered.

6. Respond to the client with the CPMGetQueryStatusExOut message.

3.1.5.2.5 Receiving a CPMRatioFinishedIn Request

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

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

2. Check whether the cursor handle passed is in the list of the clients cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Prepare a CPMRatioFinishedOut message. The server MUST retrieve the client's query status and fill in the **_ulNumerator**, **_ulDenominator**, and **_cRows** fields. If this step fails for any reason, the server MUST report that an error was encountered.

4. If **_cRows** is equal to the last reported number of rows for this query, set **_fNewRows** to 0x00000000; otherwise set it to 0x00000001.

5. Update the last reported number of rows for this query to the value of **_cRows**.

6. Respond to the client with the CPMRatioFinishedOut message.

3.1.5.2.6 Receiving a CPMGetRowsIn Request

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

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

2. Check if the cursor handle passed is in the list of the clients cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Check whether the client has a current set of bindings. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

4. Prepare a CPMGetRowsOut message. The server MUST position the cursor in the query results as indicated by the seek description. If this step fails for any reason, the server MUST report that an error was encountered.

5. Fetch as many rows as fit in a buffer, the size of which is indicated by **_cbReadBuffer**, but not more than indicated by **_cRowsToTransfer**. When fetching rows, the server MUST compare each selected column's property value type to the type that is specified in the client's current set of bindings (see section 3.1.1). If the type in the binding is not VT_VARIANT, the server MUST attempt to convert the column's property value to that type. Otherwise, if the DBPROP_USEEXTENDEDDBTYPES flag is set in the client's DBPROPSET_QUERYEXT property set, or if the column's property value is not a VT_VECTOR type, the property value MUST be returned in its normal type. If none of these are the case (that is, the server has a VT_VECTOR type, and the client does not support VT_Vector), the server MUST attempt to convert it to a VT_ARRAY type as follows:

   - VT_I8, VT_UI8, VT_FILETIME, and VT_CLSID array elements cannot be converted and instead fail.
   - VT_LPSTR and VT_LPWSTR array elements MUST be converted to VT_BSTR.
   - Array elements of all other types MUST remain unchanged.

   Finally, if row columns contain chapter handle or bookmark handle, the server MUST update the corresponding lists. If this step fails for any reason, the server MUST report that an error was encountered.

6. Store the actual number of rows fetched in **_cRowsReturned**.

7. Copy the seek description and **chapter** field from CPMGetRowsIn to a CPMGetRowsOut message to be sent.

8. Store fetched rows in the **Rows** field (see section 2.2.3.14 for details on the structure of the **Rows** field).

   Note Regarding status byte field: If StatusUsed is set to 0x01 in the CTableColumn of the CPMSetBindingIn message for the column, the server MUST set the status byte (which is located at StatusOffset from the start of the rows) for this column to one of the following values.

Value	Meaning
0x00	StatusOK
0x01	StatusDeferred
0x02	StatusNull

9. Respond to the client with the CPMGetRowsOut message.

If the property value is absent for this row, the server MUST set the status byte to StatusNull. If the value is too big to be transferred in the CPMGetRowsOut message, the server MUST set the status byte to StatusDeferred. Otherwise the server MUST set the status byte to StatusOK.

3.1.5.2.7 Receiving a CPMFetchValueIn Request

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

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

2. Prepare a CPMFetchValueOut message. If this step fails for any reason, the server MUST report an error.

3. Find the document indicated by the **_wid** field and check if the property ID for this document (later referred to as property value) indicated by the PropSpec structure is available for this client. If this value is not available, the server MUST set **_fValueExists** to 0x00000000, and otherwise set **_fValueExists** to 0x00000001. If this step fails for any reason, the server MUST report an error.

4. If **_fValueExists** is equal to 0x00000001, the server MUST do the following:

   - Serialize the property value to a SERIALIZEDPROPERTYVALUE structure and copy, starting from the **_cbSoFar** offset, at most **_cbChunk** bytes (but not past the end of the serialized property) to the **vValue** field. If this step fails for any reason, the server MUST report an error.
   - Set **_cbValue** to the number of bytes copied in previous step.
   - Set **vType** to the property type of the property value.
   - If the length of the serialized property is greater than **_cbSoFar** added to **_cbValue**, set **_fMoreExists** to 0x00000001; otherwise set it to 0x00000000.

5. Respond to the client with the CPMFetchValueOut message.

3.1.5.2.8 Receiving a CPMSetBindingsIn Request

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

1. Check whether the client has a query associated with it. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Check whether the cursor handle passed is in the list of the client's cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.
3. Verify that bindings information is valid (that is, column at least specifies 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) and if not report a DB_E_BADBINDINFO (0x80040E08) error.
4. Save the binding information associated with the columns specified in the **aColumns** field. If this step fails for any reason, the server MUST report that an error was encountered.
5. Respond to the client with a message header (only) with **_msg** set to CPMSetBindingsIn, and **_status** set to the results of the specified binding.

3.1.5.2.9 Receiving a CPMGetNotify Request

When the server receives a CPMGetNotify message from a client, the server MUST do the following:

1. Check whether the client has a query associated with it. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. If there were no changes in the query result set since the last CPMSendNotifyOut message for this client, or if the query is not currently monitored for changes in the results set, the server MUST respond with a CPMGetNotify message and start to monitor the query for changes in the results set. If at a later time there is a change in the query results set, the server MUST send exactly one CPMSendNotifyOut message to the client and MUST specify the change in the **_watchNotify** field.
3. If there were changes to the query result set since the last CPMSendNotifyOut message, the server MUST reply with CPMSendNotifyOut and MUST specify the change in the **_watchNotify** field. Note that in the case of many changes to query results, DBWATCHNOTIFY_ROWSCHANGED takes priority (that is, if the query was performed, re-executed, and then the number of rows changed and the query was performed again, then the event reported would be DBWATCHNOTIFY_ROWSCHANGED).

3.1.5.2.10 Receiving a CPMGetApproximatePositionIn Request

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

1. Check whether the client has a query associated with it. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Check whether the cursor handle, chapter handle, and bookmark handle passed are in corresponding lists. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Find a row that is associated with the bookmark handle in the query results and approximate the position of the row in the rowset, referred to by the chapter handle, and determine the numerator and denominator for the position. Note that when the chapter handle is DB_NULL_HCHAPTER, the corresponding chapter is the main rowset of the query. If this step fails for any reason, the server MUST report an error.

4. Respond to the client with a CPMFetchValueOut message.

3.1.5.2.11 Receiving a CPMCompareBmkIn Request

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

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

2. Check if the cursor handle, chapter handle and bookmark handles passed are in corresponding lists. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Prepare a CPMCompareBmkOut message.

   - If the bookmark handles are equal, **_dwComparison** MUST be set to DBCOMPARE_EQ.

   - Else, if one of the bookmark handles is DBBMK_FIRST or DBBMK_LAST, **_dwComparison** MUST be set to DBCOMPARE_NE.

   - Otherwise, the server MUST do the following:

     - Find rows that are referred to by each bookmark handle in the query results.

     - If any one of the rows is not in the chapter indicated by the chapter handle in CPMCompareBmkIn, then **_dwComparison** MUST be set to DBCOMPARE_NOTCOMPARABLE.

     - Otherwise, when both rows are in the same chapter, the server MUST approximate a position of those rows in the rowset referred to by this chapter's handle. It MUST then compare position values and set **_dwComparison** to DBCOMPARE_LT if the position of the first row is smaller than the position of the second row; otherwise **_dwComparison** MUST be set to DBCOMPARE_GT.

4. Respond to the client with the filled CPMCompareBmkOut message.

3.1.5.2.12 Receiving a CPMRestartPositionIn Request

When the server receives the CPMRestartPositionIn message request from the client, the server MUST do the following:

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

2. Check whether the cursor handle and chapter handle passed are in corresponding lists. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.

3. Move the cursor to the beginning of the chapter, identified by the chapter handle. Note that when the chapter handle is DB_NULL_HCHAPTER, the corresponding chapter is the main rowset of the query. If this step fails for any reason, the server MUST report an error.

4. Respond to the client with a CPMRestartPositionIn message.

3.1.5.2.13 Receiving a CPMStopAsynchIn Request

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

1. Check whether the client has a query associated with it. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Check whether the cursor handle passed is in the list of the client's cursor handles and the cursor was created with the eAsynchronous flag (0x00000008) set. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.
3. Stop asynchronous query processing.
4. Respond to the client with the message header for the CPMStopAsynchIn, and set the **_status** field to the result of step 3.

3.1.5.2.14 Receiving a CPMFreeCursorIn Request

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

1. Check whether the client has a query associated with it. If this is not the case, the server MUST report a STATUS_INVALID_PARAMETER (0xC000000D) error.
2. Check whether the cursor handle passed is in the list of the client's cursor handles. If this is not the case, the server MUST report an E_FAIL (0x80004005) error.
3. Release the cursor and associated resources (see section 3.1.1 for a complete list) for this cursor handle.
4. Remove the cursor from the list of cursors for that client.
5. Respond with a CPMFreeCursorOut message, setting the **_cCursorsRemaining** field with the number of cursors remaining in this client's list.
6. If there are no more cursors for this client, the server MUST release the query and associated resources (see section 3.1.1).

3.1.5.2.15 Receiving a CPMDisconnect Request

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

3.1.6 Timer Events

None.

3.1.7 Other Local Events

When the 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 specifies data and state maintained by the Windows Search Protocol client. The provided data is to 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 that described in this document.

A client has the following abstract state:

State	Description
Last Message Sent	A copy of the last message sent to the 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 the Current Property Value so far.
Named Pipe Connection	A connection to the server.

3.2.2 Timers

There are no timers required for this protocol.

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

A header MUST be pre-pended 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 contents of the message after the **_ulReserved2** field in the message header MUST be interpreted as a sequence of 32-bit integers. The 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 Remote Windows Search Service Catalog Management

Each message is triggered by a request from the higher layer. There is no message sequence enforced by the client for the Windows Search Protocol message requests for remotely managing the catalog. However, the server will only reply with success if the client previously connected by means of a CPMConnectIn message.

3.2.4.1.1 Sending a CPMCiStateInOut Request

Typically, the higher layer asks the protocol client to send a CPMCiStateInOut message when it requires information about the Windows Search service on the server.

When requested to send this message, the client MUST do the following:

1. Send a CPMCiStateInOut message to the server.
2. Wait to receive a CPMCiStateInOut message from the server, silently discarding all other messages.
3. Report the value of the **_status** field of the response, and if it was successful, report the informational structure back to the higher layer.

3.2.4.1.2 Sending a CPMUpdateDocumentsIn Request

The higher layer typically asks to send this message when it needs to either update documents in an existing path or add a new file path to the inverted index. Thus the higher layer is to provide the path and type of scan, which is specified in section 2.2.3.2, where an incremental or full update is meant for existing paths and new initialization is meant for new paths.

To serve the higher-layer request, the client MUST do the following:

1. Send a CPMUpdateDocumentsIn message to the server.
2. Wait to receive a CPMUpdateDocumentsIn message back from the server, silently discarding all other messages.
3. Report the value of the **_status** field of the response back to the higher layer.

3.2.4.1.3 Sending a CPMForceMergeIn Request

Typically, the higher layer makes a request to send this message when there is a need to improve query performance, or it is a part of scheduled Windows Search service maintenance.

To serve the higher layer the client MUST do the following:

1. Send a CPMForceMergeIn message to the server.
2. Wait to receive a CPMUpdateDocumentsIn message back from the server, silently discarding all other messages.
3. Report the value of the **_status** field of the response back to the higher layer.

3.2.4.2 Remote Windows Search Service Catalog Query Messages

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

For illustrative purposes, the client portion of the diagram in section 3 illustrates this sequence for a simple Windows Search service query.

3.2.4.2.1 Sending a CPMConnectIn Request

This message is typically the very first request from the higher layer. The higher level provides the protocol client with information necessary to connect.

To serve the higher layer, the client MUST do the following:

1. Fill in the message, using information provided by the higher-layer client (see section 2.2.3.4) in **_iClientVersion**, MachineName, UserName, PropertySet1, PropertySet2, and aPropertySet.
2. Set **_fClientIsRemote**, **_cbBlob**, **_cbBlob2**, **cPropSet**, and **cExtPropSet**, as specified in section 2.2.3.4.
3. Set the checksum in the **_ulChecksum** field.
4. Send the CPMConnectIn message to the server.
5. Wait to receive a CPMConnectOut message back from the server, silently discarding all other messages.
6. Report the value of the **_status** field of the response and, if it was successful, report the value of the **_serverVersion** back to the higher layer.

For informative purposes, it is expected that higher layers will typically do the following actions upon successful connection, but these are not enforced by the Windows Search Protocol client:

- Use remote Windows Search service catalog management messages for administrative tasks.
- Use a CPMCreateQueryIn request to create a search query for the purpose of retrieving results from the catalog.

3.2.4.2.2 Sending a CPMCreateQueryIn Request

The higher layer will typically provide information for the query creation after the protocol client is connected. The higher layer provides the client with a restrictions set, column set, sort order rules, and categorization set (each of which can be omitted), rowset properties, and property ID mapper structure.

When this request is received from a higher layer, the client MUST perform the following:

1. Prepare a CPMCreateQueryOut as follows.
   - If a columns set is present, set CColumnsSetPresent to 0x01 and fill the **ColumnsSet** field.
   - If restrictions are present, set CRestrictionPresent to 0x01 and fill the **Restriction** field.
   - If a sort set is present, fill the **SortSet** field.
   - If a categorization set is present, set CSortSetPresent to 0x01 and fill the **CategorizationSet** field.
   - Set the rest of fields as specified in section 2.2.3.6.
2. Calculate **_ulCheckSum** field in the header.
3. Send the CPMCreateQueryIn message to the server.
4. Wait to receive the CPMCreateQueryOut message, silently discarding all other messages.

5. Report the value of the **_status** field of the response and, if it was successful, report the array of cursor handles and informative Boolean values (as specified in section 2.2.3.7) back to the higher layer.

3.2.4.2.3 Sending a CPMSetBindingsIn Request

Typically, the higher layer will set bindings for each column to be returned in the rows when it already has a valid cursor handle (after successfully receiving CPMCreateQueryOut). 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 client MUST perform 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 completed CPMSetBindingsIn message to the server.
5. Wait to receive a CPMSetBindingsIn message from server, discarding other messages.
6. Indicate the status from the **_status** field of the response to the higher layer.

For informative purposes, it is expected that higher layers will typically request a client to send a CPMGetRowsIn message, but this is not enforced by the Windows Search Protocol.

3.2.4.2.4 Sending a CPMGetRowsIn Request

When the higher layer is about to receive rows information, it will provide the protocol client with valid cursor and chapter handles and give an appropriate seek description. Typically, a higher layer is expected to do so when it has a valid cursor and/or chapter handle, and the bindings had been set with the CPMSetBindingsIn message. To access the rowset in a chapter, the higher layer is to use chapter handle received from the server in a previous CPMGetRowsOut message.

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

1. Determine what unsigned integer value to specify for the **_cbReadBuffer** field. To determine this value, the client MUST take the maximum value from the following:
   - One thousand times the value of the **c_RowsToTransfer** field.
   - Value of **_cbRowWidth**, rounded up to the nearest 512 byte multiple.
   - Take the higher of these two values, up to the 16K limit.
   - In cases where a single row is larger than 16K, the 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.<10>
3. Calculate the size of the seek description and set it in the **_cbSeek** field.

4. Set the value of **cbReserved** (which would act as an offset for the start of the rows contained in the **Rows** field in the CPMGetRowsOut message) to the value of **_cbSeek** plus 0x14.

5. Send a CPMConnectIn message to the server.

3.2.4.2.5 Sending a CPMFetchValueIn Request

If the client receives a CPMGetRowsOut response from the server with the column's **Status** field set to StatusDeferred (0x01), it means that the property value was not included in the **Rows** field of the CPMGetRowsOut message. In this case, the higher layer typically asks the protocol client to retrieve the value by means of a CPMFetchValueIn message, and provides the PropSpec and **_wid** value for a deferred property, which the protocol client MUST use in the first CPMFetchValueIn message.

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

1. Set all the fields in a message, as specified in section 2.2.3.17.

2. Set _cbSoFar to 0x00000000.

3. Set current bytes received to 0.

4. Send the CPMFetchValueIn message to the server.

3.2.4.2.6 Sending a CPMFreeCursorIn Request

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

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

3.2.4.2.7 Sending a CPMDisconnect Message

If the higher layer has no more queries for the Windows Search service, to free up more server resources, the application may request that the client send a CPMDisconnect message to the server. When this query is received, the client MUST simply send the message as requested. There is no response to this message from the server.

3.2.5 Message Processing and Sequencing Rules

When the client receives a message response from the server, the client MUST use the Last Sent Message to determine if the message received from the server is the one expected by the 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 client receives a CPMCreateQueryOut message response from the server, the client MUST return **_status**, and if the status is successfully, return the cursor handle values back to the higher layer. Any further actions are up to the higher layer.

As the higher layer is aware of query structure, it is expected that the correct number of cursor handles will be returned in the CPMCreateQueryOut message. The cursor handles are returned in the following order: the first handle is to the unchaptered rowset and the second handle is to the first

chaptered rowset (which is the grouping of results based on the first category specified in the **CategorizationSet** field of the CPMCreateQueryIn message).

For informative purposes, it is expected that higher layers can perform the following actions, but these are not enforced by the Windows Search Protocol client:

- Use CPMSetBindingsIn to set bindings for individual columns and perform any subsequent actions on query the path.
- Use CPMGetQueryStatusIn to check on the execution progress of a query.
- Use CPMRatioFinishedIn to request the completion percentage of the query.

3.2.5.2 Receiving a CPMGetRowsOut Response

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

1. Check whether the **_status** field in the header indicates success or failure.
2. If the **_status** value is STATUS_BUFFER_TOO_SMALL (0xC0000023), the 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 client MUST send to the 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 the Last Message Sent already has **_cbReadBuffer** equal to 0x4000, the client MUST report the error up to the higher level.
3. If the **_status** value is any other error value, the 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.

For informative purposes, it is expected that higher layers will typically perform the following actions, but these are not enforced by the Windows Search Protocol client:

- If the values in rows represent the document IDs, chapter, or bookmark handles, the higher layer will typically store them for use in subsequent operations that involve valid document IDs, chapter, or bookmark handles.
- The higher layer will typically store or display or otherwise use the data from row values.
- For the values that were marked as deferred, the higher layer will fetch the value using CPMFetchValueIn messages.
- The seek description is returned back to the higher layer as well, and can be reused or examined by the higher layer.

For informative purposes, if the higher layer requested handles to chapters and bookmarks that were received in the rows, it may perform the following:

- Use CPMGetQueryStatusExIn to check on the execution progress of a query as well as additional status information, such as the number of filtered documents, documents remaining to be filtered, the ratio of documents processed by the query, the total number of rows in the query, and the position of the bookmark in the rowset.

- Use CPMGetNotify to request that the server notify the client of rowset changes.
- Use CPMGetApproximatePositionIn to request the approximate position of a bookmark in a chapter.
- Use CPMCompareBmkIn to request a comparison of two bookmarks in a chapter.
- Use CPMRestartPositionIn to request the server changes the location of the cursor to the start of rowset.

3.2.5.3 Receiving a CPMFetchValueOut Response

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

1. Check whether the **_status** field in the header indicates success or failure. In a case of failure, notify the higher layer. Otherwise, continue as per the following:
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 the Current Property Value.
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 required by the higher layer.
5. If **_fMoreExists** is set to 0x00000000, indicate the property value from the Current Property Value to the higher layer.

3.2.5.4 Receiving a CPMFreeCursorOut Response

When the client receives a successful CPMFreeCursorOut message response from the server, the 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 Windows Search Protocol client. The higher layer is expected to keep track of cursor handles and not to use those which have already been freed. When the value of **_cCursorsRemaining** is equal to 0x00000000, the higher layer can use the connection to specify another query (using a CPMCreateQueryIn message).

3.2.6 Timer Events

No timer events are required by this protocol.

3.2.7 Other Local Events

No other local events are required by this protocol.

4 Protocol Examples

4.1 Example 1

In the following example, consider a scenario in which the user, UserA on machine UserA-2A, wants to obtain the path files that contain the word "flowers" from the set of files stored on server UserA-4 in catalog SYSTEM. The query, as expressed in the Microsoft Windows Search SQL syntax, is:

[SQL]

```
SELECT Path FROM UserA-4.SystemIndex..Scope() WHERE "SCOPE"
= 'file://UserA-4/Users/UserA/Pictures' AND CONTAINS(*, '"flowers"')
```

Assume that machine UserA-2A is running a 32-bit Windows Vista operating system, and machine UserA-4 is running a 64-bit Windows Vista operating system.

1. The user launches a search 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 search server UserA-4. The protocol client uses the named pipe \pipe\MsFteWds to connect to the server UserA-4 over SMB.
3. The protocol client then prepares a CPMConnectIn message with the following values:
   - 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.
   - The body of the message is populated as follows:
     - **_iClientVersion** is set to 0x00000102, indicating that the server is to validate the **checksum** field and that the client is Windows Vista or later.
     - **_fClientIsRemote** is set to 0x00000001, indicating that the server is a remote server.
     - **_cbBlob1** is set to 340, which is the size, in bytes, of the **cPropSet**, **PropertySet1** and **PropertySet2** fields combined.
     - **_cbBlob2** is set to 1124, which is the size, in bytes of the **cExPropSet** and **aPropertySet** fields combined.
     - **MachineName** is set to USERA-2A.
     - **UserName** is set to UserA.
     - **cPropSets** is set to 0x00000002.

Note In the following descriptions of CDbProp structures, several fields always contain default values. They are omitted from the description to improve clarity. The fields with default values are:

- **DBPROPOPTIONS** is set to 0x0000000.
- **DBPROPSTATUS** is set to 0x00000000.
- For the **colid** element:
  - **eKind** is set to 0x00000001 (DBKIND_GUID_PROPID).
  - GUID is null (all zeros), meaning the value applies to the query, not to just a single column.
  - **ulID** is set to 0x00000000.

- The **PropertySet1** field is of type CDbPropSet. The CDbPropSet structure comprising the **PropertySet1** 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 0x00000004.
  - The **aProps** field is an array of CDbProp structures.
  - For the **aProps[0]** element:
    - PropId is set to 0x00000002 (DBPROP_CI_CATALOG_NAME).
    - For the **vValue** element:
      - **vType** is set to 0x001F (VT_LPWSTR).
      - **vValue** is set to "Windows\SYSTEMINDEX", the name of the desired catalog.
  - For the **aProps[1]** element:
    - PropId is set to 0x00000007 (DBPROP_CI_QUERY_TYPE).
    - For the **vValue** element:
      - **vType** is set to 0x0003 (VT_I4).
      - **vValue** is set to 0x00000000 (CiNormal), meaning it is a regular query.
  - For the **aProps[2]** element:
    - PropId is set to 0x00000004 (DBPROP_CI_SCOPE_FLAGS).
    - For the **vValue** element:
      - **vType**: 0x1003 (VT_VECTOR | VT_I4).
      - **vValue**: 0x00000001 / 0x00000001 (one element with value 1), meaning search subfolders.
  - For the **aProps[3]** element:

        - PropId is set to 0x00000003 (DBPROP_CI_INCLUDE_SCOPES).
        - For the **vValue** element:
            - **vType** is set to 0x101F (VT_VECTOR | VT_LPWSTR).
            - **vValue** is set to 0x00000001 / 0x00000002 / "\" (one element with a two-character null-terminated string), meaning the root scope.
    - The **PropertySet2** field is of type CDbPropSet. The CDbPropSet structure comprising the **PropertySet1** field is populated as follows:
        - **GuidPropSet** is set to AFAFACA5-B5D1-11D0-8C62-00C04FC2DB8D (DBPROPSET_CIFRMWRKCORE_EXT).
        - The **cProperties** field is set to 0x00000001.
        - The **aProps** field is an array of CDbProp structures.
        - For the **aProps[0]** element:
            - **PropId** is set to 0x00000002 (DBPROP_MACHINE).
            - For **vValue** element:
                - **vType**: 0x0008 (VT_BSTR).
                - **vValue**: 0x10 / "USERA-4" (16 bytes / null-terminated Unicode string), meaning "USERA-4" – name of a server.
    - The **cExPropSet** field is set to 0x00000004.
    - **aPropertySets[0]** is of type CDbPropSet and contains the following fields:
        - The **GuidPropSet** field is set to GUID: AA6EE6B0-E828-11D0-B23E-00AA0047FC01 (DBPROPSET_MSIDXS_ROWSETEXT).
        - The **cProperties** field is set to 0x00000006.
        - The **aProps** field is an array of CDbProp structures.
        - For the **aProps[0]** element:
            - **PropId** is set to 0x00000002 (MSIDXSPROP_ROWSETQUERYSTATUS).
            - For the **vValue** element:
                - **vType** is set to 0x0003 (VT_I4).
                - **vValue** is set to 0x0000, meaning nothing. This value is ignored by all Windows implementations of the protocol.
        - For the **aProps[1]** element:
            - **PropId** is set to 0x00000003 (MSIDXSPROP_COMMAND_LOCALE_STRING).
            - For the **vValue** element:
                - **vType** is set to 0x0008 (VT_BSTR).

- **vValue** is set to 0x6 / "EN" (6 bytes / null-terminated Unicode string), meaning the user locale is English.

- For the **aProps[2]** element:
  - **PropId** is set to 0x00000004 (MSIDXSPROP_QUERY_RESTRICTION).
  - For the **vValue** element:
    - **vType** is set to 0x0008 (VT_BSTR).
    - **vValue** is set to 0x2 / "" (2 bytes / null-terminated empty Unicode string), meaning nothing. This value is ignored by all Windows implementations of the protocol.
- For the **aProps[3]** element:
  - PropId is set to 0x00000005 (MSIDXSPROP_PARSE_TREE).
  - For the **vValue** element:
    - **vType** is set to 0x0008 (VT_BSTR).
    - **vValue** is set to 0x2 / "" (2 bytes / null-terminated empty Unicode string), meaning nothing. This value is ignored by all Windows implementations of the protocol.
- For the **aProps[4]** element:
  - PropId is set to 0x00000006 (MSIDXSPROP_MAX_RANK).
  - For the **vValue** element:
    - **vType** is set to 0x0003 (VT_I4).
    - **vValue** is set to 0x0000, meaning nothing. This value is ignored by all Windows implementations of the protocol.
- For the **aProps[5]** element:
  - PropId is set to 0x00000007 (MSIDXSPROP_RESULTS_FOUND).
  - For the **vValue** element:
    - **vType** is set to 0x0003 (VT_I4).
    - **vValue** is set to 0x0000, meaning nothing. This value is ignored by all Windows implementations of the protocol.

- aPropertySets[1] is of type CDbPropSet and contains the following fields:
  - The **GuidPropSet** field is set to GUID: A7AC77ED-F8D7-11CE-A798-0020F8008025 (DBPROPSET_QUERYEXT).
  - The **cProperties** field is set to 0x0000000A.
  - The **aProps** field is an array of CDbProp structures.
  - For the **aProps[0]** element
    - **PropId** is set to 0x00000002 (DBPROP_USECONTENTINDEX).

        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning forced use of the full text index is FALSE.
    - For the **aProps[1]** element:
        - PropId is set to 0x00000003 (DBPROP_DEFERNONINDEXEDTRIMMING).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning trimming of security results should not be deferred.
    - For the **aProps[2]** element:
        - PropId is set to 0x00000004 (DBPROP_USEEXTENDEDDBTYPES).
        - For the **vValue** element
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning do not use extended DB types.
    - For the **aProps[3]** element:
        - PropId is set to 0x00000005 (DBPROP_IGNORENOISEONLYCLAUSES).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x00 (FALSE), meaning full text clauses consisting entirely of noise words will result in an error being returned.
    - For the **aProps[4]** element:
        - PropId is set to 0x00000006 (DBPROP_GENERICOPTIONS_STRING).
        - For the **vValue** element:
            - **vType** is set to 0x0008 (VT_BSTR).
            - **vValue** is set to 0x2 / "" (2 bytes / null-terminated empty Unicode string), meaning no generic options were set.
    - For the **aProps[5]** element:
        - PropId is set to 0x00000008 (DBPROP_DEFERCATALOGVERIFICATION).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning catalog verification is not deferred.
    - For the **aProps[6]** element:

        - PropId is set to 0x0000000E (DBPROP_IGNORESBRI).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning the query can use the sort-by-rank index optimization.
    - For the **aProps[7]** element:
        - **PropId** is set to 0x0000000A (DBPROP_GENERATEPARSETREE).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning a parse tree is not generated for debugging.
    - For the **aProps[8]** element:
        - **PropId** is set to 0x0000000C (DBPROP_FREETEXTANYTERM).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning all terms from a FREETEXT clause must appear in every matching document.
    - For the **aProps[9]** element:
        - **PropId** is set to 0x0000000D (DBPROP_FREETEXTUSESTEMMING).
        - For the **vValue** element:
            - **vType** is set to 0x000B (VT_BOOL).
            - **vValue** is set to 0x0000 (FALSE), meaning stemming is not used when interpreting a FREETEXT clause.
- aPropertySets[2] is of type CDbPropSet and contains the following fields:
    - The **GuidPropSet** field is set to GUID: B5D1AFAF-11D0-628C-00C0-4FC2DB8D0000 (DBPROPSET_CIFRMWRKCORE_EXT).
    - The **cProperties** field is set to 0x00000001.
    - The **aProps** field is an array of CDbProp structures.
    - For the **aProps[0]** element:
        - PropId is set to 0x00000002 (DBPROP_MACHINE).
        - For the **vValue** element:
            - **vType** is set to 0x0008 (VT_BSTR).

- **vValue** is set to 0x0010 / "USERA-4" (7 Unicode characters + null = 0x10 bytes), meaning the target server is named USERA-4.

- **aPropertySets[3]** is of type CDbPropSet and contains the following fields:
  - The **GuidPropSet** field is set to GUID: A9BD1526-6A80-11D0-8C9D-0020AF1D740E (DBPROPSET_FSCIFRMWRK_EXT)
  - The **cProperties** field is set to 0x00000003.
  - The **aProps** field is an array of CDbProp structures.
  - For the **aProps[0]** element:
    - **PropId** is set to 0x00000003 (DBPROP_CI_INCLUDE_SCOPES).
    - For the **vValue** element:
      - **vType** is set to 0x2008 (VT_ARRAY | VT_BSTR).
      - **vValue** is set to:
        - cDims = 0x0001
        - fFeatures = <ignore 2 bytes>
        - cbElements = 0x00000004 (4 bytes per element)
        - SAFEARRAYBOUND[0].cElements = 0x00000001 (1 element)
        - SAFEARRAYBOUND[0].lLbound = 0x00000000 (0-based)
        - element = 0x00000004 / "\" (1 Unicode character plus null), meaning search everywhere ("\" is the root scope)
  - For the **aProps[1]** element:
    - PropId is set to 0x00000004 (DBPROP_CI_SCOPE_FLAGS).
    - For the **vValue** element:
      - **vType** is set to 0x2003 (VT_ARRAY | VT_I4).
      - **vValue** is set to:
        - cDims = 0x0001
        - fFeatures = <ignore 2 bytes>
        - cbElements = 0x00000004 (4 bytes per element)
        - SAFEARRAYBOUND[0].cElements = 0x00000001 (1 element)
        - SAFEARRAYBOUND[0].lLbound = 0x00000000 (0-based)
        - element = 0x00000001, meaning QUERY_DEEP.
  - For the **aProps[2]** element:
    - PropId is set to 0x00000002 (DBPROP_CI_CATALOG_NAME).

        - For the **vValue** element:
            - **vType** is set to 0x0008 (VT_BSTR).
            - **vValue** is set to 0x00000028 / "Windows\SYSTEMINDEX" (19 Unicode characters plus null), meaning the catalog to use is named Windows\SYSTEMINDEX.
    - Various padding fields are filled in as needed. The message is sent to the server.
4. The server verifies that the **_ulChecksum** is correct, verifies that the user is authorized to make this request, and responds with a CPMConnectOut message.
    - 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.
    - The body of the message is populated as 0x00010102 (64-bit Windows Vista).
5. The **_reserved** fields are filled with arbitrary data.
6. The client prepares a CPMCreateQueryIn message.
    - 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 3.2.5.
        - **_ulReserved2** is set to 0x00000000.
    - 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 is 0x00000000, indicating the first entry in **_aPropSpec**.
    - The **CRestrictionPresent** field is set to 0x01, indicating the **RestrictionArray** field is present.
        - Set **RestrictionArray** such that:
            - **Count** is set to 0x01.
            - **isPresent** is set to 0x01 (restriction present).

- The **Restriction** field is of type CRestriction and is set to:
  - **_ulType** is set to 0x00000001 (RTAnd).
  - **_weight** is set to 0x000003E8.
- The rest of the field contains a CNodeRestriction structure:
  - **_cNode** is set to 0x00000002, meaning two clauses are AND-ed together.
  - **_paNode[0]** is of type CRestriction and is set to:
    - **_ulType** is set to 0x00000005 (RTProperty.
    - **_weight** is set to 0x000003E8.
    - The rest of the field contains a CPropertyRestriction structure:
      - **_relop** is set to 0x00000004 (PREQ), meaning a property is tested for equality.
      - **_Property** is a CFullPropspec structure that is set to GUID B725F130-47EF-101A-A5-F1-02608C9EEBAC / 0x00000001 (for PRSPEC_PROPID) /0x00000016, which represents the Windows scope property.
      - **_PrVal** is a CBaseStorageVariant:
        - **vType** is set to 0x001F (VT_LPWSTR).
        - **vValue** is set to the Unicode string "file://UserA-4/Users/UserA/Pictures" which is the scope to match.
      - **_lcid** is 0x0409, meaning U.S. English.
  - **_paNode[1]** contains:
    - **_ulType** is set to 0x00000004 (RTContent.
    - **_weight** is set to 0x000003E8.
    - The rest of the field contains a CContentRestriction structure:
      - **_Property** is set to GUID 49691C90-7E17-101A-A91C-08002B2ECDA9/ 0x00000001 (for PRSPEC_PROPID) / 0x00000006, meaning DISPID_QUERY_ALL (for example, search across all properties).
      - **_Cc** is set to 0x00000007.
      - **_pwcsphrase** is set to the string "flowers".
      - **_lcid** is set to 0x409 (for English).
      - **_ulGenerateMethod** is set to 0x00000000 (exact match).
  - **CSortPresent** is set to 0x00.
  - **CCategorizationSetPresent** is set to 0x00.
- **RowSetProperties** is set as follows:

- **_uBooleanOptions** is set to 0x00000001 (sequential).
- **_ulMaxOpenRows** is set to 0x00000000.
- **_ulMemoryUsage** is set to 0x00000000.
- **_cMaxResults** is set to 0x00000000 (return all rows).
- **_cCmdTimeOut** is set to 0x0000001E (timeout after 30ms).

- **PidMapper** is set to:
  - **_count** is set to 0x00000003.
  - **_aPropSpec[0]** is set to GUID B725F130-47EF-101A-A5-F1-02608C9EEBAC / 0x00000001 (for PRSPEC_PROPID)/0x0000000B, which represents the Windows path property.
  - **_aPropSpec[1]** is set to GUID B725F130-47EF-101A-A5-F1-02608C9EEBAC / 0x00000001 (for PRSPEC_PROPID)/0x00000016, which represents the Windows scope property.
  - **_aPropSpec[2]** is set to GUID 49691C90-7E17-101A-A91C-08002B2ECDA9/ 0x00000001 (for PRSPEC_PROPID)/0x00000006, which represents the "all properties" property.
  - **ColumnGroupArray** is set to 0x00000000 meaning no weights for columns are specified.
  - **_lcid** is set to 0x00000409 meaning U.S. English is the default locale for the query.

7. The server processes it and responds with a CPMCreateQueryOut message.
   - 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 (or any other arbitrary value).
   - The body of the message is populated as follows:
     - **_fTrueSequential** is set to 0x00000001, indicating that the query cannot use an index.
     - **_fWorkIdUnique** is set to 0x00000001.
     - The **aCursors** array contains only one element, representing a cursor handle to this query. The value depends on the state of the server, assuming that the returned value is 0xAAAAAAAA.
8. The client issues a CPMSetBindingsIn request message to define the format of a row.
   - 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** is set to 0x00000000 (or any other arbitrary value).

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

- The body of the message is populated as follows:
  - **_hCursor** is set to 0xAAAAAAAA.
  - **_cbRow** is set to 0x20 (big enough to fit columns).
  - **_cbBindingDesc** is set to 0x61, the size of the **_cColumns** and **_aColumns** fields combined.
  - **_dummy** is set to an arbitrary value such as 0x0A00000C.
  - **_cColumns** is set to 0x00000002.
  - The **_aColumns** array is set to contain two CTableColumn structures.
  - **_aColumns[0]** contains:
    - **_PropSpec** is set to GUID B725F130-47EF-101A-A5-F1-02608C9EEBAC / 0x00000001 (for PRSPEC_PROPID) / 0x0000000B, which represents the Windows path property.
    - **vType** is set to 0x000C (VT_VARIANT).
    - **_AggregateStored** is set to 0x01, meaning aggregate info is present.
    - **_AggregateType** is set to 0x00, meaning the query is not aggregated.
    - **_ValueUsed** is set to 0x01 (column transferred in row).
    - **_ValueOffset** is set to 0x0008 (8 bytes from beginning of row).
    - **_ValueSize** is set to 0x0010 (size of a VT_VARIANT).
    - **_StatusUsed** is set to 0x01 (status is reported in row).
    - **_StatusOffset** is set to 0x0002.
    - **_LengthUsed** is set to 0x01 (length is reported).
    - **_LengthOffset** is set to 0x0004.
  - **_aColumns[1]** contains:
    - **_PropSpec** is set to GUID 49691C90-7E17-101A-A91C-08002B2ECDA9/ 0x00000001 (for PRSPEC_PROPID)/0x00000005, which represents the private internal document ID of an indexed document.
    - **_vType** is set to 0x0003 (VT_I4).
    - **_AggregateStored** is set to 0x01, meaning aggregate info is present.
    - **_AggregateType** is set to 0x00, meaning the query is not aggregated.
    - **_ValueUsed** is set to 0x01 (column transferred in row).
    - **_ValueOffset** is set to 0x0018 (24 bytes from beginning of row).
    - **_ValueSize** is set to 0x0004 (size of a VT_I4).

- **_StatusUsed** is set to 0x01 (status is reported in row).
- **_StatusOffset** is set to 0x0003.
- **_LengthUsed** is set to 0x00 (length is not reported).

9. The 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 (or any other arbitrary value).
10. The client issues a CPMGetRowsIn request message, assuming that the client is prepared to accept 32 rows at this point, and wants them in ascending order.
    - 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 according to section 3.2.4.
      - **_ulReserved2** is set to 0x00000000.
    - The body of the message is populated as follows:
      - **_hCursor** is set to 0xAAAAAAAA.
      - **_cRowsToTransfer** is set to 0x00000014.
      - **_cRowWidth** is set to 0x00000020 (from bindings).
      - **_cbSeek** is set to 0x0000000C, which is the size of the **eType**, **_chapt**, and **CRowSeekNext** fields combined.
      - **_cbReserved** is set to 0x0x20 (0x14 plus **_cbSeek**).
      - **_cbReadBuffer** is set to 0x4000 (because 1000 * 0x20 > 0x4000).
      - **_ulClientBase** is set to 0x03C924C8 (execution dependent).
      - **_fBwdfetch** is set to 0x00000000, indicating that the rows are to be fetched in forward order.
      - **eType** is set to 0x0000001, indicating that the client wants next rows.
      - **_chapt** is set to 0 (not a chaptered result).
      - **eType** is set to 0x00000000 (eRowsSeekNone), meaning start with next 0x00000000.
    - The body of the message is populated as follows:
      - **_CRowsReturned** is set to 0x00000002.

- **eType** is set to 0x00000000 (eRowsSeekNone).
- **_chapt** is set to 0x00000000 (not a chaptered result).
- Rows contain the size of the two documents that contain the word "flowers". The raw buffer will look something like (-- indicates unused space) the following:

```
-- -- 00 00 (status OK for both columns)
7E 00 00 00 (length = 0x7E - 0x10 (inline) = 0x6E bytes of string)
1F 00 -- -- (VT_LPWSTR)
-- -- -- --
58 64 C9 03 (address of VT_LPWSTR: 0x03C96458 - base 0x03C924C8 =
             offset 0x3F90 into buffer)
-- -- -- --
96 02 00 00 (WorkId = 0x296)
-- -- -- --

-- -- 00 00 (status OK for both columns)
86 00 00 00 (length = 0x86 - 0x10 (inline) = 0x76 bytes of string)
1F 00 -- -- (VT_LPWSTR)
-- -- -- --
E0 63 C9 03 (address of VT_LPWSTR: 0x03C963E0 - base 0x03C924C8 =
             offset 0x3F18 into buffer)
-- -- -- --
97 02 00 00 (WorkId = 0x297)
-- -- -- --
```

- And at offset 0x3F90 into the buffer will be a Unicode string of length 55 (including null) "file://UserA-4/Users/UserA/Pictures/forest flowers.jpg".
- And at offset 0x3F18 into the buffer will be a Unicode string of length 59 (including null) "file://UserA-4/Users/UserA/Pictures/frangipani flowers.jpg".

11. The client issues another CPMGetRowsIn request message to look for more rows.

- 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 according to section 3.2.4.
  - **_ulReserved2** is set to 0x00000000.
- The body of the message is populated as follows:
  - **_hCursor** is set to 0xAAAAAAAA.
  - **_cRowsToTransfer** is set to 0x00000014.
  - **_cRowWidth** is set to 0x00000020 (from bindings).
  - **_cbSeek** is set to 0x0000000C, which is the size of the **eType**, **_chapt** and **CRowSeekNext** fields combined.

- **_cbReserved** is set to 0x0x20 (0x14 plus _cbSeek).
- **_cbReadBuffer** is set to 0x4000 (because 1000 * 0x20 > 0x4000).
- **_ulClientBase** is set to 0x03C924C8 (execution dependent).
- **_fBwdfetch** is set to 0x00000000 indicating that the rows are to be fetched in forward order.
- **eType** is set to 0x0000001 indicating that the client wants next rows.
- **_chapt** is set to 0 (not a chaptered result).
- **eType** is set to 0x00000000 (eRowsSeekNone) meaning start with next available row.

12. The server processes it and responds with a CPMGetRowsOut message, assuming the server found no more documents that contain the word "flowers".
    - 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.
    - The body of the message is populated as follows:
      - **_CRowsReturned** is set to 0x00000000, meaning no more rows are available.
13. The client sends a CPMFreeCursor message to close the handle to the query.
    - The header of the message is populated as follows:
      - **_msg** is set to 0x000000CB, indicating that this is a CPMFreeCursorIn message.
      - **_status** is set to 0x00000000.
      - **_ulChecksum** contains the checksum, computed according to section 3.2.4.
      - **_ulReserved2** is set to 0x00000000.
    - The body of the message is populated as follows:
      - **_hCursor** is set to 0xAAAAAAAA.
14. The server processes it and responds with a CPMFreeCursor message.
    - The header of the message is populated as follows:
      - **_msg** is set to 0x000000CB, indicating that this is a CPMFreeCursorIn message.
      - **_status** is set to 0x00000000.
      - **_ulChecksum** contains the checksum, computed according to section 3.2.4.
      - **_ulReserved2** is set to 0x00000000.

- The body of the message is populated as follows:
  - **_cCursorsRemaining** is set to 0x00000000, meaning no more cursors are active for the query.

15.The 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.

The server processes the message and removes all client states.

5 Security

The following sections specify security considerations for administrators.

5.1 Security Considerations for Implementers

Indexing implementations that index secure content should consider using the user context provided by [MS-SMB] to trim search results and return only those results accessible to the caller.

5.2 Index of Security Parameters

Security parameter	Section
Impersonation level	2.1

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.4: Applications typically interact with an OLE DB interface wrapper (as specified in [MSDN-OLEDBP]), (for example, a protocol client) and not directly with the protocol.

<2> Section 1.5: In Windows Vista the default catalog name is SystemIndex.

<3> Section 1.6: This protocol is implemented on Windows Vista.

<4> Section 1.8: Windows only uses the values specified in [MS-ERREF].

<5> Section 2.2.2: Always set by the client.

Note The **_status** field MUST be set to 0x00000000.

<6> Section 2.2.3.1: This value is usually zero, except immediately after indexing has been started or after a notification queue overflows.

<7> Section 2.2.3.4: On Windows-based clients, the **iClientVersion** is set as follows.

Value	Meaning
0x00000102	Client OS is either 32-bit Windows XP, 32-bit Windows Server 2003, or 32-bit Windows Vista.
0x00010102	Client OS is either Windows XP 64-Bit Edition, 64-bit Windows Server 2003, or 64-bit Windows Vista.

<8> Section 2.2.3.17: This field is set for all versions of Windows.

Note This field MUST be set to 0x00004000.

<9> 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.

<10> Section 3.2.4.2.4: For a 32-bit client talking to a 32-bit server, or a 64-bit client talking to a 64-bit 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 Index

A

Abstract data model
  client
  server
Administration - remote
Applicability

C

CAggregSet packet
CAggregSortKey packet
CAggregSpec packet
Capability negotiation
CBaseStorageVariant packet
CCategorizationSet packet
CCategorizationSpec packet
CCategSpec packet
CCoercionRestriction packet
CColumnGroup packet
CColumnGroupArray packet
CColumnSet packet
CContentRestriction packet
CDbColId packet
CDbProp packet
CDbPropSet packet
CFeedbackRestriction packet
CFullPropSpec packet
CInGroupSortAggregSet packet
CInGroupSortAggregSets packet
CInternalPropertyRestriction packet
CKey packet
Client
  abstract data model
  higher-layer triggered events
  initialization
  local events
  message processing
  overview
  sequencing rules
  timer events
  timers
CNatLanguageRestriction packet
CNodeRestriction packet
CPidMapper packet
CPMCiStateInOut (section 3.1.5.1.1, section 3.2.4.1.1)
CPMCiStateInOut packet
CPMCompareBmkIn
CPMCompareBmkIn packet
CPMCompareBmkOut packet
CPMConnectIn (section 3.1.5.2.1, section 3.2.4.2.1)
CPMConnectIn packet
CPMConnectOut packet
CPMCreateQueryIn (section 3.1.5.2.2, section 3.2.4.2.2)
CPMCreateQueryIn packet
CPMCreateQueryOut
CPMCreateQueryOut packet
CPMDisconnect (section 3.1.5.2.15, section 3.2.4.2.7)
CPMDisconnect packet
CPMFetchValueIn (section 3.1.5.2.7, section 3.2.4.2.5)
CPMFetchValueIn packet
CPMFetchValueOut
CPMFetchValueOut packet
CPMForceMergeIn (section 3.1.5.1.3, section 3.2.4.1.3)
CPMForceMergeIn packet
CPMFreeCursorIn (section 3.1.5.2.14, section 3.2.4.2.6)
CPMFreeCursorIn packet
CPMFreeCursorOut
CPMFreeCursorOut packet
CPMGetApproximatePositionIn
CPMGetApproximatePositionIn packet
CPMGetApproximatePositionOut packet
CPMGetNotify
CPMGetNotify packet
CPMGetQueryStatusExIn
CPMGetQueryStatusExIn packet
CPMGetQueryStatusExOut packet
CPMGetQueryStatusIn
CPMGetQueryStatusIn packet
CPMGetQueryStatusOut packet
CPMGetRowsIn (section 3.1.5.2.6, section 3.2.4.2.4)
CPMGetRowsIn packet
CPMGetRowsOut
CPMGetRowsOut packet
CPMRatioFinishedIn
CPMRatioFinishedIn packet
CPMRatioFinishedOut packet
CPMRestartPositionIn
CPMRestartPositionIn packet
CPMSendNotifyOut packet
CPMSetBindingsIn (section 3.1.5.2.8, section 3.2.4.2.3)
CPMSetBindingsIn packet
CPMStopAsynchIn
CPMStopAsynchIn packet
CPMUpdateDocumentsIn (section 3.1.5.1.2, section 3.2.4.1.2)
CPMUpdateDocumentsIn packet
CProbRestriction packet
CPropertyRestriction packet
CRangeCategSpec packet
CRelDocRestriction packet
CRestriction packet
CRestrictionArray packet
CRowSeekAt packet
CRowSeekAtRatio packet
CRowSeekByBookmark packet
CRowSeekNext packet
CRowsetProperties packet
CRowVariant packet
CScopeRestriction packet
CSort packet
CSortAggregSet packet
CSortSet packet
CTableColumn packet
CVectorRestriction packet

D

Data model - abstract
  client
  server
DECIMAL packet

E

Errors
Errors packet
Examples
  overview
  query example

F

Fields - vendor-extensible

G

Glossary

H

Headers - message
Higher-layer triggered events
  client
  server

I

IDs - property
Implementer - security considerations
Index of security parameters
Informative references
Initialization
  client
  server
Introduction

L

Local events
  client
  server

M

Message Headers packet
Message processing
  client
  server
Messages
  descriptions
  headers
  overview
  structures
  syntax
  transport

N

Normative references

O

Overview (synopsis)

P

Parameters - security index
Preconditions
Property IDs

Q

Query example
Querying - remote

R

RANGEBOUNDARY packet
References
  informative
  normative
  overview
Relationship to other protocols
Remote administration
Remote querying
Remote Windows search service catalog management (section 3.1.5.1, section 3.2.4.1)
Remote Windows search service catalog query
Remote Windows search service query

S

SAFEARRAY packet
SAFEARRAY2 packet
SAFEARRAYBOUND packet
Security
  implementer considerations
  overview
  parameter index
Sequencing rules
  client
  server
SERIALIZEDPROPERTYVALUE packet
Server
  abstract data model
  higher-layer triggered events
  initialization
  local events
  message processing
  overview
  sequencing rules
  timer events
  timers
SProperty packet
Standards assignments
Structures
Syntax - message

T

Timer events
client
server
Timers
client
server
Transport
Triggered events - higher-layer
client
server

V

Vendor-extensible fields
Versioning
VT_COMPRESSED_LPWSTR packet
VT_Vector packet

W

Windows behavior