[MS-DCHT]:

Desktop Chat Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
8/14/2009	0.1	Major	First Release.
9/25/2009	0.1.1	Editorial	Changed language and formatting in the technical content.
11/6/2009	0.1.2	Editorial	Changed language and formatting in the technical content.
12/18/2009	0.1.3	Editorial	Changed language and formatting in the technical content.
1/29/2010	0.2	Minor	Clarified the meaning of the technical content.
3/12/2010	0.2.1	Editorial	Changed language and formatting in the technical content.
4/23/2010	0.2.2	Editorial	Changed language and formatting in the technical content.
6/4/2010	0.3	Minor	Clarified the meaning of the technical content.
7/16/2010	0.3	None	No changes to the meaning, language, or formatting of the technical content.
8/27/2010	0.3	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	0.3	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	0.3	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	0.3	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	0.3	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	0.3	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	0.3	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	0.4	Minor	Clarified the meaning of the technical content.
9/23/2011	0.4	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	0.4	None	No changes to the meaning, language, or formatting of the technical content.
3/30/2012	0.4	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	0.4	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	0.4	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	0.4	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
8/8/2013	0.4	None	No changes to the meaning, language, or formatting of the technical content.
11/14/2013	0.4	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	0.5	Minor	Clarified the meaning of the technical content.
5/15/2014	0.5	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	0.5	None	No changes to the meaning, language, or formatting of the technical content.
10/16/2015	1.0	Major	Significantly changed the technical content.
7/14/2016	1.0	None	No changes to the meaning, language, or formatting of the technical content.
6/1/2017	1.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Desktop Chat Protocol specifies the mechanism by which the Windows Chat application in Windows communicates information between remote users.

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

1.1 Glossary

This document uses the following terms:

chat session: An active communication channel established between two computers using the Microsoft Desktop Chat Protocol.

client: A computer on which the remote procedure call (RPC) client is executing.

device: Any peripheral or part of a computer system that can send or receive data.

marshal: To encode one or more data structures into an octet stream using a specific remote procedure call (RPC) transfer syntax (for example, marshaling a 32-bit integer).

NetBIOS: A particular network transport that is part of the LAN Manager protocol suite. **NetBIOS** uses a broadcast communication style that was applicable to early segmented local area networks. A protocol family including name resolution, datagram, and connection services. For more information, see [RFC1001] and [RFC1002].

server: A computer on which the remote procedure call (RPC) server is executing.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

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

1.2 References

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

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ISO/IEC-8859-1] International Organization for Standardization, "Information Technology -- 8-Bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1", ISO/IEC 8859-1, 1998, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=28245

Note There is a charge to download the specification.

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

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

1.2.2 Informative References

[MSDN-DDE] Microsoft Corporation, "Dynamic Data Exchange (DDE) and DDE Management Library", http://msdn.microsoft.com/en-us/library/ms648712(VS.85).aspx

[MSDN-LOGFONT] Microsoft Corporation, "LOGFONT data structure", http://msdn.microsoft.com/en-us/library/dd145037(VS.85).aspx

[MSDN-NETDDE] Microsoft Corporation, "Network Dynamic Data Exchange", http://msdn.microsoft.com/en-us/library/aa365778(VS.85).aspx

[MSDN-Win32Edit] Microsoft Corporation, "Win32 Edit control", http://msdn.microsoft.com/en-us/library/cc656455(VS.85).aspx

1.3 Overview

This document specifies the Desktop Chat Protocol, used by the Windows Chat application. This Desktop Chat Protocol initiates a **chat session** between two machines and allows for the real-time transfer of textual data between them.

1.4 Relationship to Other Protocols

The Desktop Chat Protocol is implemented on top of the NetDDE protocol [MSDN-NETDDE].

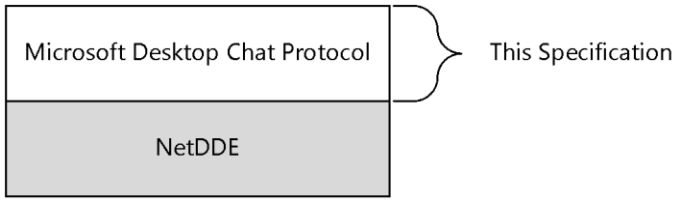


Figure 1: The Desktop Chat Protocol relationship to other protocols

1.5 Prerequisites/Preconditions

The Desktop Chat Protocol requires the NetDDE API. The NetDDE protocol [MSDN-NETDDE] service is required to be running on the participant machines.

1.6 Applicability Statement

The Desktop Chat Protocol is used for real-time text communication.

1.7 Versioning and Capability Negotiation

The Desktop Chat Protocol does not have multiple versions.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

This protocol references commonly used data types as defined in [MS-DTYP]

2.1 Commonly Referenced Data Structures, Values, and Definitions

The Desktop Chat Protocol specifies the following data structures.

2.1.1 Common Definitions

2.1.2 Common Field Values

2.1.2.1 ChatMessageType

The **ChatMessageType** enumeration describes the type of message sent between participants in a desktop **chat session**.

Value	Meaning
CHT_CHAR 0x0100	The message contains information about a single character that has been inputted in a chat session.
CHT_FONTA 0x0101	The message contains information about the font being used in a chat session that does not support **Unicode**.
CHT_PASTEA 0x0102	The message contains information about a text paste operation that has been performed in a chat session that does not support Unicode.
CHT_DBCS_STRING 0x0103	The message contains multi-byte encoded string data entered into a chat session that does not support Unicode.
CHT_PROTOCOL 0x0105	The message contains protocol information related to establishing a chat session.
CHT_UNICODE 0x0110	The message describes the ability of the chat participant to use Unicode data.
CHT_FONTW 0x0111	The message contains information about the font being used in a chat session that supports Unicode.
CHT_PASTEW 0x0112	The message contains information about a text paste operation that has been performed in a chat session that supports Unicode.

2.2 Transport

The NetDDE API MUST be present. The NetDDE protocol service [MSDN-NETDDE] MUST be running on the participant machines.

Desktop Chat protocol messages are encapsulated in data blocks delivered using NetDDE [MSDN-NETDDE], as described in detail in section 3.1.3.

2.3 Message Syntax

All **CHATDATA** structures defined in section 2.3.1 and 2.3.2 begin with a **ChatMessageType** defining the type of message.

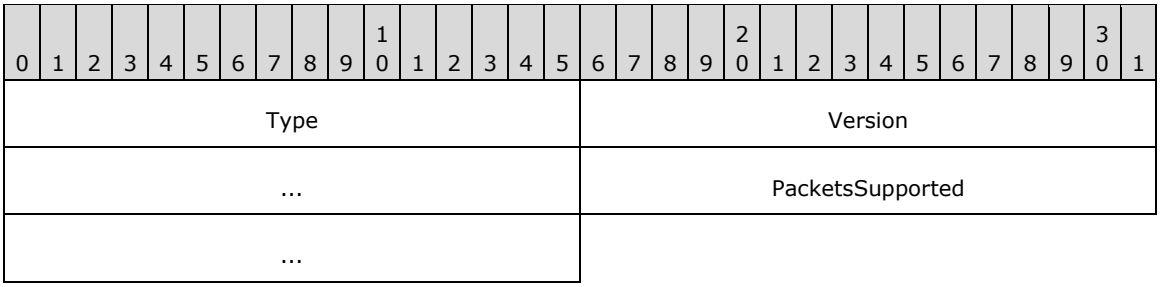
Note All unsigned 16-bit and unsigned 32-bit values are specified in little-endian format. Depending on the hardware architectures of the **client** and the **server**, multiple-byte little-endian versus big-endian reordering can determine how this variable is **marshaled** by the sender and interpreted by the receiver.

2.3.1 Session Information

The Desktop Chat Protocol uses session information to establish the capabilities of the remote participant.

2.3.1.1 CHATDATA_PROTOCOL

The CHATDATA_PROTOCOL structure specifies version information for future use.



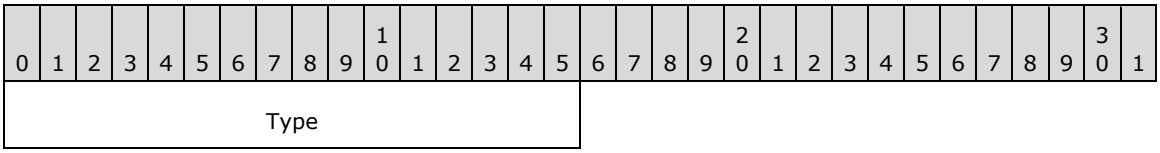
Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0105 (**CHT_PROTOCOL**).

Version (4 bytes): A 32-bit unsigned integer specifying the version information. MUST be set to 0x00000100.

PacketsSupported (4 bytes): A 32-bit unsigned integer specifying capability information. MUST be set to 0x00000001.

2.3.1.2 CHATDATA_UNICODE

The CHATDATA_UNICODE structure indicates that the sender supports Unicode text.



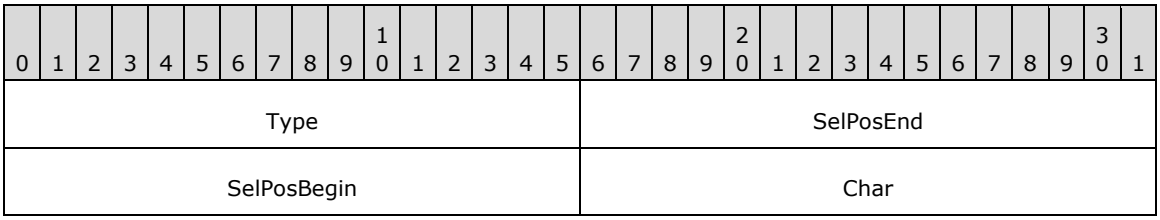
Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0110 (**CHT_UNICODE**).

2.3.2 Chat Information

The Desktop Chat Protocol transfers chat information between two participants, representing textual user input that has been entered into the chat application.

2.3.2.1 CHATDATA_CHAR

The CHATDATA_CHAR structure defines a user-input character that is sent in the **chat session**.



Type (2 bytes): A 16-bit unsigned integer (ChatMessageType) specifying the type of this message. MUST be set to 0x0100 (CHT_CHAR).

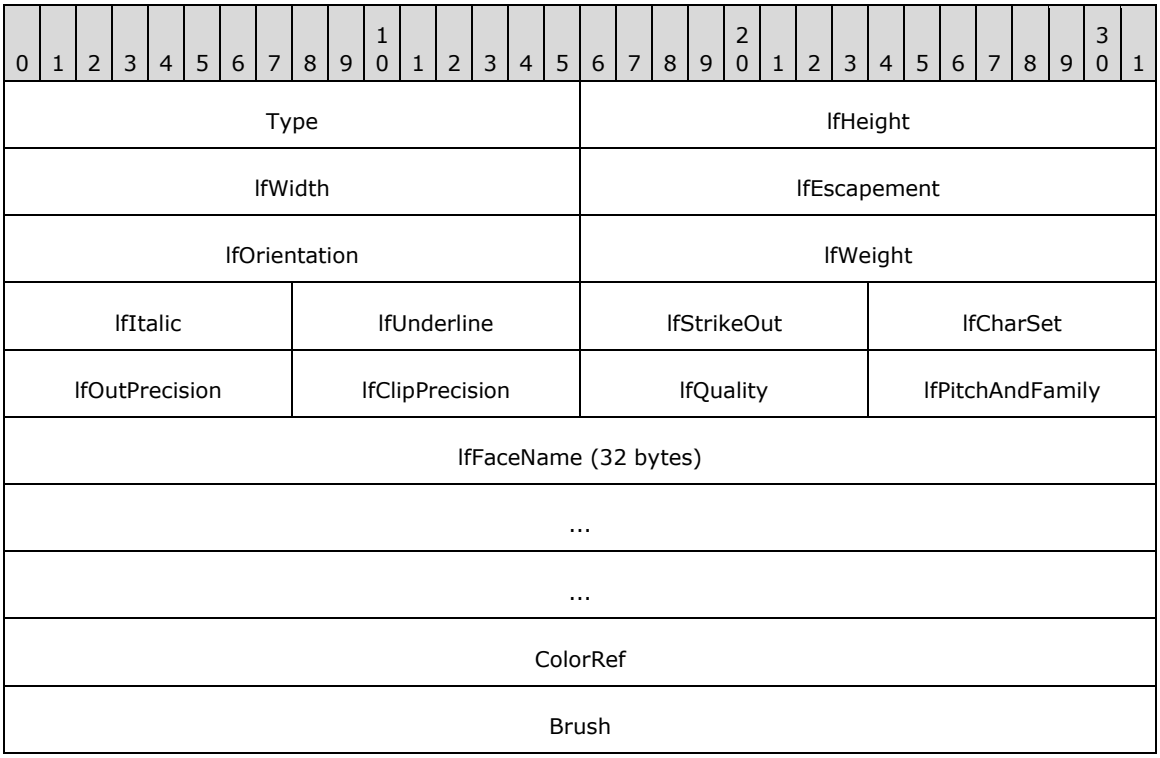
SelPosEnd (2 bytes): A 16-bit unsigned integer that specifies the zero-based position of the first character after the last selected character. <1>

SelPosBegin (2 bytes): A 16-bit unsigned integer that specifies the zero-based starting position of the selection. <2>

Char (2 bytes): A 16-bit unsigned integer that specifies the character that the user has inputted. If the Desktop chat session has been negotiated as Unicode (**SessionState.Unicode** field is true), this MUST be a Unicode character. If the Desktop chat session has not been negotiated as Unicode (**SessionState.Unicode** field is false), this MUST be an ANSI (as specified in [ISO/IEC-8859-1]) or Double-Byte Character Sets (DBCS) character.

2.3.2.2 CHATDATA_FONTA

The CHATDATA_FONTA structure defines the font that is being used in the **chat session** by a user. <3>



Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0101 (**CHT_FONTA**).

lfHeight (2 bytes): A 16-bit signed integer that specifies the height, in logical units, of the font's character cell or character.

lfWidth (2 bytes): A 16-bit signed integer that specifies the average width, in logical units, of characters in the font.

lfEscapement (2 bytes): A 16-bit signed integer that specifies the angle, in tenths of degrees, between the escapement vector and the x-axis of the **device**. The escapement vector is parallel to the base line of a row of text.

lfOrientation (2 bytes): A 16-bit signed integer that specifies the angle, in tenths of degrees, between each character's base line and the x-axis of the device.

lfWeight (2 bytes): A 16-bit signed integer that specifies the weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

lfItalic (1 byte): An 8-bit unsigned integer that specifies an italic font if set to 0x01. MUST be set to 0x00 or 0x01.

lfUnderline (1 byte): An 8-bit unsigned integer that specifies an underlined font if set to 0x01. MUST be set to 0x00 or 0x01.

lfStrikeOut (1 byte): An 8-bit unsigned integer that specifies a strikeout font if set to 0x01. MUST be set to 0x00 or 0x01.

lfCharSet (1 byte): An 8-bit unsigned integer that specifies the character set.

lfOutPrecision (1 byte): An 8-bit unsigned integer that specifies the output precision.

lfClipPrecision (1 byte): An 8-bit unsigned integer that specifies the clipping precision.

lfQuality (1 byte): An 8-bit unsigned integer that specifies the output quality.

lfPitchAndFamily (1 byte): An 8-bit unsigned integer that specifies the pitch and family of the font.

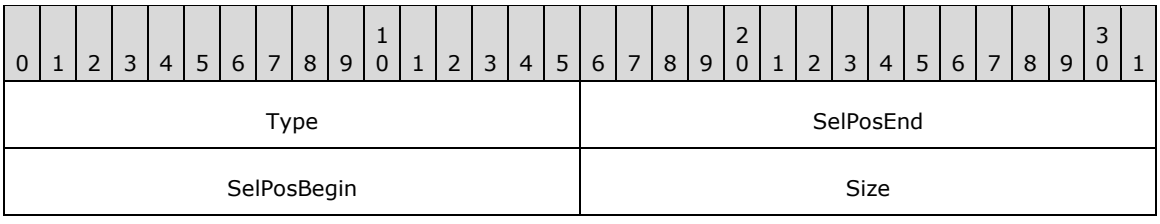
lfFaceName (32 bytes): A NULL-terminated ANSI character string that specifies the typeface name of the font.

ColorRef (4 bytes): A **COLORREF** 32-bit value that specifies the RGB color used for the edit control text [MSDN-Win32Edit].

Brush (4 bytes): A COLORREF 32-bit value that specifies the RGB color used for the edit control background [MSDN-Win32Edit].

2.3.2.3 CHATDATA_PASTEA

The CHATDATA_PASTEA structure indicates that a user has pasted text from the clipboard into the chat session.



...	Unused (50 bytes)
...	
...	
PastedText (variable)	
...	

Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0102 (**CHT_PASTEA**).

SelPosEnd (2 bytes): A 16-bit unsigned integer that specifies the zero-based position of the first character after the last selected character. <4>

SelPosBegin (2 bytes): A 16-bit unsigned integer that specifies the zero-based starting position of the selection. <5>

Size (4 bytes): A 32-bit unsigned integer that specifies the length of the pasted text in bytes, not including the NULL terminator.

Unused (50 bytes): Unused. MUST be set to zero.

PastedText (variable): A NULL-terminated ANSI character string specifying the pasted text.

2.3.2.4 CHATDATA_DBCS_STRING

The CHATDATA_DBCS_STRING structure indicates that a user has entered text into the **chat session** that is being sent to a receiver that does not support Unicode.

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
Type																SelPosEnd															
SelPosBegin																Size															
...																unused (50 bytes)															
...																															
...																															
DBCSText (variable)																															
...																															

Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0103 (**CHT_DBCS_STRING**).

SelPosEnd (2 bytes): A 16-bit unsigned integer that specifies the zero-based position of the first character after the last selected character. <6>

SelPosBegin (2 bytes): A 16-bit unsigned integer that specifies the zero-based starting position of the selection. <7>

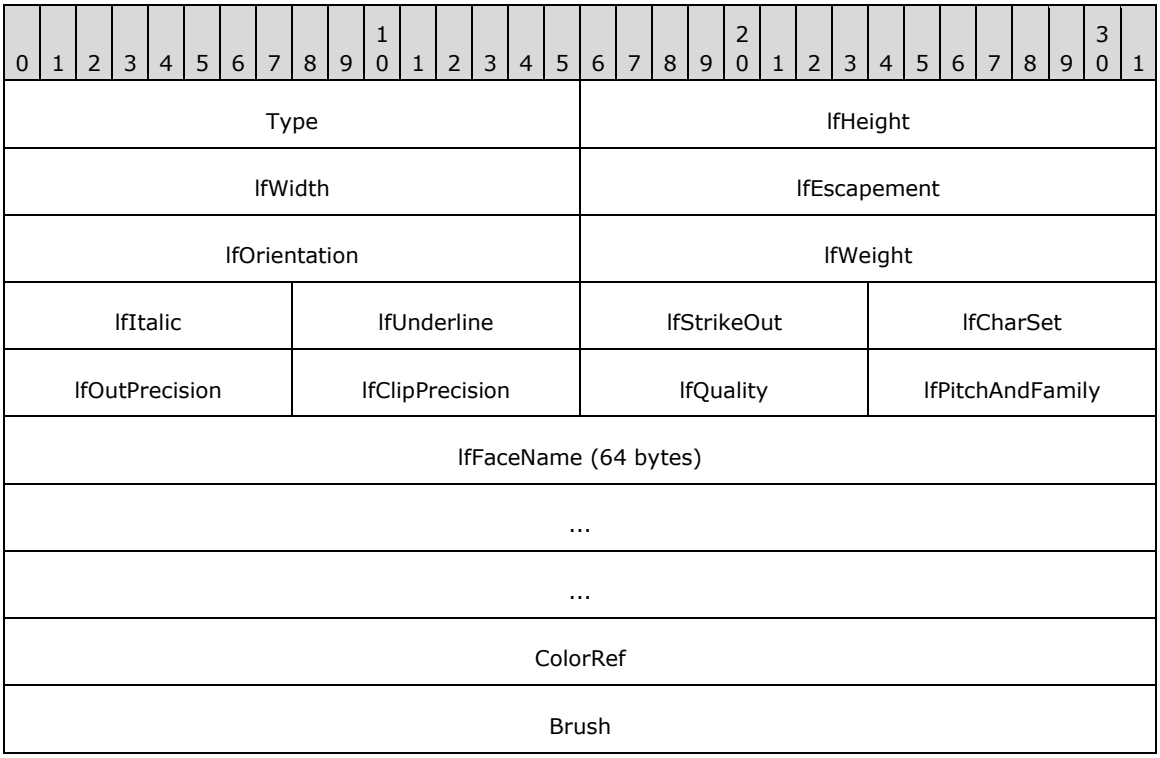
Size (4 bytes): A 32-bit unsigned integer that specifies the length of the DBCS string in bytes, not including the NULL terminator.

unused (50 bytes): Unused. MUST be zeroed.

DBCSText (variable): A NULL-terminated DBCS character string.

2.3.2.5 CHATDATA_FONTW

The CHATDATA_FONTW structure defines the font that is being used in the **chat session** by a user. <8>



Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0111 (**CHT_FONTW**).

lfHeight (2 bytes): A 16-bit signed integer that specifies the height, in logical units, of the font's character cell or character.

lfWidth (2 bytes): A 16-bit signed integer that specifies the average width, in logical units, of characters in the font.

lfEscapement (2 bytes): A 16-bit signed integer that specifies the angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the baseline of a row of text.

lfOrientation (2 bytes): A 16-bit signed integer that specifies the angle, in tenths of degrees, between each character's baseline and the x-axis of the device.

lfWeight (2 bytes): A 16-bit signed integer that specifies the weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used.

lfItalic (1 byte): An 8-bit unsigned integer that specifies an italic font if set to 0x01. MUST be set to 0x00 or 0x01.

lfUnderline (1 byte): An 8-bit unsigned integer that specifies an underlined font if set to 0x01. MUST be set to 0x00 or 0x01.

lfStrikeOut (1 byte): An 8-bit unsigned integer that specifies a strikeout font if set to 0x01. MUST be set to 0x00 or 0x01.

lfCharSet (1 byte): An 8-bit unsigned integer that specifies the character set.

lfOutPrecision (1 byte): An 8-bit unsigned integer that specifies the output precision.

lfClipPrecision (1 byte): An 8-bit unsigned integer that specifies the clipping precision.

lfQuality (1 byte): An 8-bit unsigned integer that specifies the output quality.

lfPitchAndFamily (1 byte): An 8-bit unsigned integer that specifies the pitch and family of the font.

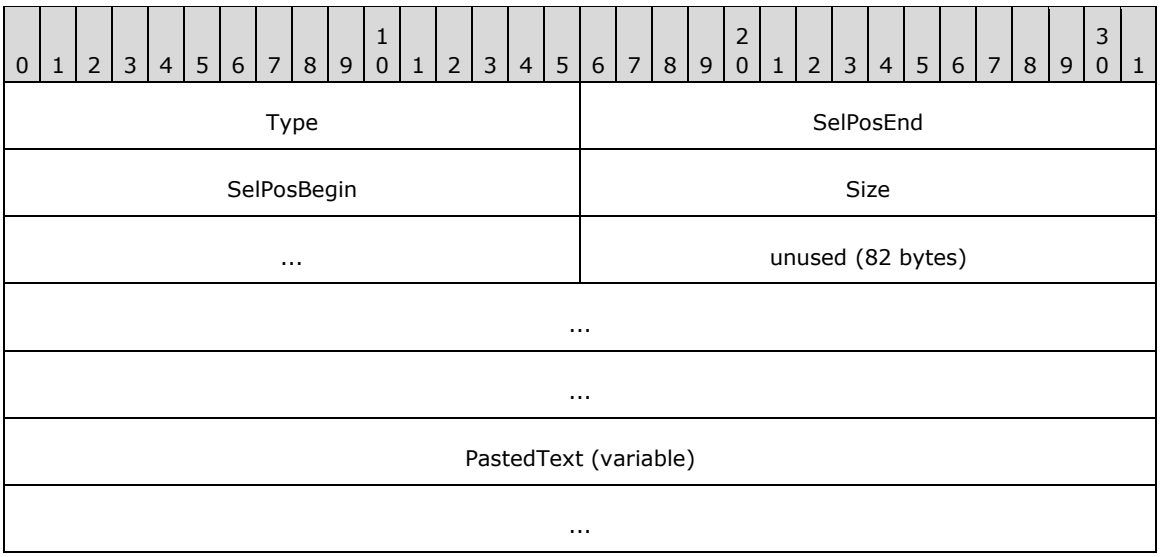
lfFaceName (64 bytes): A NULL-terminated Unicode string that specifies the typeface name of the font.

ColorRef (4 bytes): A COLORREF 32-bit value that specifies the RGB color used for the edit control text [MSDN-Win32Edit].

Brush (4 bytes): A COLORREF 32-bit value that specifies the RGB color used for the edit control background [MSDN-Win32Edit].

2.3.2.6 CHATDATA_PASTEW

The CHATDATA_PASTEW structure indicates that a user has pasted Unicode text from the clipboard into the **chat session**.



Type (2 bytes): A 16-bit unsigned integer (**ChatMessageType**) specifying the type of this message. MUST be set to 0x0112 (**CHT_PASTEW**).

SelPosEnd (2 bytes): A 16-bit unsigned integer that specifies the zero-based position of the first character after the last selected character. <9>

SelPosBegin (2 bytes): A 16-bit unsigned integer that specifies the zero-based starting position of the selection. <10>

Size (4 bytes): A 32-bit unsigned integer that specifies the length of the pasted text, in bytes, not including the NULL terminator.

unused (82 bytes): Unused. MUST be zeroed.

PastedText (variable): A NULL-terminated Unicode character string specifying the pasted text.

3 Protocol Details

3.1 Desktop Chat Details

3.1.1 Abstract Data Model

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

Participants in the Desktop Chat Protocol SHOULD maintain the following state.

Session: An application using the Desktop Chat Protocol has a certain state associated with the chat.

Session.Unicode: Initialized to false. If true, the participant MUST NOT send **CHATDATA_PASTEA** or **CHATDATA_FONTA** messages and MUST pass Unicode information in the char field of **CHATDATA_CHAR** messages. If false, the participant MUST NOT send **CHATDATA_PASTEW** or **CHATDATA_FONTW** messages and MUST pass ANSI or DBCS information in the char field of **CHATDATA_CHAR** messages.

In addition to the above state, the server participant in the Desktop Chat protocol MUST maintain the following state.

Session.ClientName: The **NetBIOS** name of the client.

3.1.2 Timers

None.

3.1.3 Initialization

A NetDDE conversation is first established between a client and server machine as follows (see [MSDN-NETDDE] and [MSDN-DDE] for further details):

- The NetDDE server machine creates a static network DDE share using **NddeShareAdd** method. The DDE share MUST be named "CHAT$" and MUST have the static topic name of "Chat".
- The NetDDE client machine initiates the DDE conversation using **DdeConnect** method. The DdeConnect service name MUST be of the form "**computername**\NDDE$", where **computername** is the **NetBIOS** name of the server machine. The DdeConnect topic name MUST be "CHAT$".
- The NetDDE server machine MUST acquire the client name using **DdeQueryString** method during the XTYP_ADVSTART callback. This name is stored as **Session.ClientName**.
- Chat message data as defined in section 2.3 is encapsulated in NetDDE data blocks for transfer between chat participants.

Storing message in DDE data block: The participant uses **DdeCreateDataHandle** method to create a data block of the correct size as specified by section 2.3. The string specifying the data item MUST be "ChatText". The clipboard format MUST be the result from RegisterClipboardFormat ("Chat Data"). The participant copies the **CHATDATA** message to the DDE data block using **DdeAddData method**.

Extracting message from DDE data block: The participant uses **DdeGetData** method to retrieve the first two bytes of data from the DDE data block, which it interprets as **ChatMessageType**. Based on this, the participant determines the message type and the remaining message size as specified by section 2.3 and reads the data with subsequent **DdeGetData** method calls.

Message sent from client to server: The client **stores** the message in a DDE data block. The DDE data is sent using **DdeClientTransaction** with XTYP_POKE as the transaction type. The server receives this information via an XTYP_POKE transaction processed by its **DdeCallback** function, and **extracts** the message from the DDE data block.

Message sent from server to client: The server MUST use **DdePostAdvise method** to inform the client that data is available. The DDE topic name MUST be "Chat" and the DDE item name MUST be **Session.ClientName**. The client will present the server with an XTYP_ADVREQ transaction. The server **stores** the message and returns the resulting data block to complete the XTYP_ADVREQ transaction. The client receives an XTYP_ADVDATA transaction in its **DdeCallback** function, and **extracts** the message from the DDE data block.

After initialization of the NetDDE conversation, the participant SHOULD send a **CHATDATA_PROTOCOL** message. If the participant supports Unicode, the participant MUST send a **CHATDATA_UNICODE** message.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

Malformed, unrecognized, and out-of-sequence packets MUST be ignored by the **server** and the **client**.

The participant inspects the first two bytes of the message to determine the message type, as indicated by the **ChatMessageType** enumeration and the message structures in section 2.3.

3.1.5.1 CHATDATA_PROTOCOL

This message MUST be ignored.

3.1.5.2 CHATDATA_UNICODE

Upon receiving this message, if the participant supports **Unicode**, the participant MUST set **SessionState.Unicode** to true.

3.1.5.3 CHATDATA_CHAR

Upon receiving this message, the participant MAY present the remote user's text using the data in the message.

3.1.5.4 CHATDATA_FONTA

Upon receiving this message, if presenting the chat data in visual form to the user, the participant SHOULD adjust the visual representation of the remote user's text using the data in the message.

3.1.5.5 CHATDATA_PASTEA

Upon receiving this message, the participant MAY present the remote user's text using the data in the message.

3.1.5.6 CHATDATA_DBCS_STRING

Upon receiving this message, the participant MAY present the remote user's text using the data in the message.

3.1.5.7 CHATDATA_FONTW

Upon receiving this message, if presenting the chat data in visual form to the user, the participant SHOULD adjust the visual representation of the remote user's text using the data in the message.

3.1.5.8 CHATDATA_PASTEW

Upon receiving this message, the participant MAY present the remote user's text using the data in the message.

4 Protocol Examples

4.1 Sample Chat Session

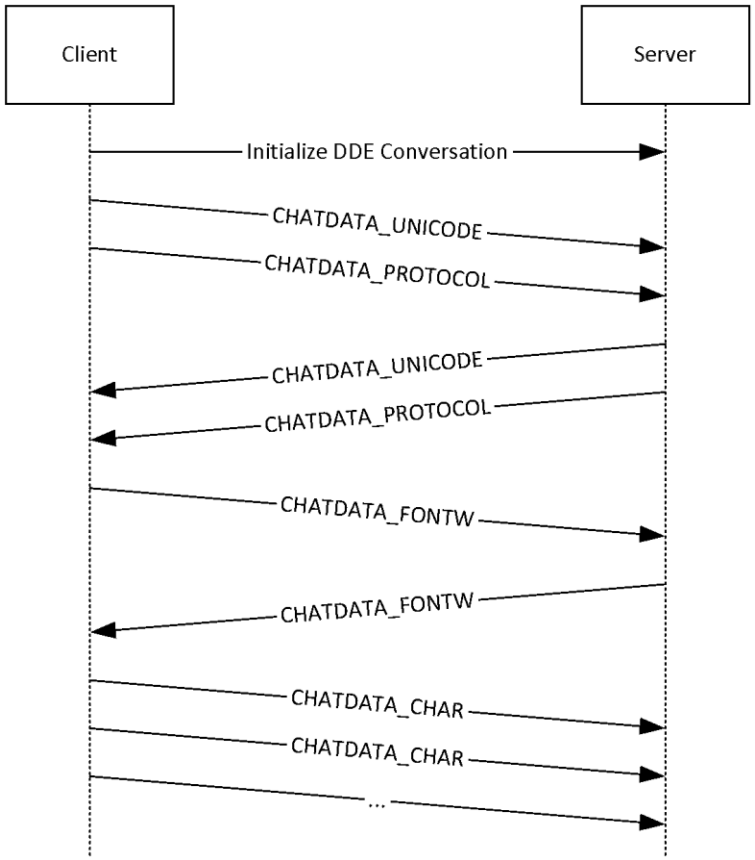


Figure 2: Sample chat session

In the preceding diagram, a **client** and **server** first negotiate a DDE conversation as described in section 3.1.3.

The following messages are then transmitted:

1. The client sends a CHATDATA_UNICODE message, indicating **Unicode** support.
2. The client sends a CHATDATA_PROTOCOL message, indicating versioning information.
3. The server sends a CHATDATA_UNICODE message, indicating Unicode support.
4. The server sends a CHATDATA_PROTOCOL message, indicating versioning information.
5. The client sends a CHATDATA_FONTW message, providing the font information that is currently being used by the client. The server updates the font used for display of the client's text.
6. The server sends a CHATDATA_FONTW message, providing the font information that is currently being used by the server. The client updates the font used for display of the server's text.
7. The client sends CHATDATA_CHAR messages, corresponding to user input in the chat application on the client machine. The server processes these messages and displays the received text in an edit control presented to the user on the server.

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

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

- Windows 2000 operating system Service Pack 4 (SP4)
- Windows XP operating system
- Windows Server 2003 operating system

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

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

<1> Section 2.3.2.1: In Windows environments, **SelPosEnd** is equivalent to the high-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<2> Section 2.3.2.1: In Windows environments, **SelPosBegin** is equivalent to the low-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<3> Section 2.3.2.2: In Windows environments, these "lf" structure fields map to the LOGFONT structure [MSDN-LOGFONT].

<4> Section 2.3.2.3: In Windows environments, **SelPosEnd** is equivalent to the high-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<5> Section 2.3.2.3: In Windows environments, **SelPosBegin** is equivalent to the low-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<6> Section 2.3.2.4: In Windows environments, **SelPosEnd** is equivalent to the high-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<7> Section 2.3.2.4: In Windows environments, **SelPosBegin** is equivalent to the low-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<8> Section 2.3.2.5: In Windows environments, these "lf" structure fields map to the **LOGFONT** structure [MSDN-LOGFONT].

<9> Section 2.3.2.6: In Windows environments, **SelPosEnd** is equivalent to the high-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

<10> Section 2.3.2.6: In Windows environments, **SelPosBegin** is equivalent to the low-order word of the return value of the **EM_GETSEL** window message sent to an edit control [MSDN-Win32Edit].

7 Change Tracking

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

8 Index

R

S

T

V