[MS-DPDX]:
DirectPlay DXDiag Usage Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	0.1.1	Editorial	Revised and edited the technical content.
10/23/2007	0.2	Minor	Updated the technical content.
11/30/2007	0.3	Minor	Updated the technical content.
01/25/2008	1.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the DirectPlay DXDiag Usage Protocol as it is used natively in Windows. The DirectPlay DXDiag Usage Protocol is intended for peer-to-peer network video gaming. This document covers the DirectPlay DXDiag Usage Protocol as it is used by the **DXDiag application**, which is part of the Windows operating system. A revision of this document will cover the non-native aspects of the DirectPlay DXDiag Usage Protocol as it can be used by other applications.

1.1 Glossary

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

Client
Cyclic Redundancy Check (CRC)
Internet Protocol Security (IPsec)
ISO/OSI Reference Model
Little-Endian
Server

The following terms are specific to this document:

Acknowledgment (ACK): A signal passed between communicating processes or computers to signify receipt of response as part of a communications protocol.

Coalesced Payload: A special form of payload that consists of multiple traditional payloads combined into a single packet.

CRC-16-IBM Algorithm: The **CRC-16-IBM algorithm** polynomial is x^16 + x^15 + x^2 + 1. Normal and reversed representations are "0x8005" or "0xA001".

DirectPlay: A network communication library included with the Microsoft **DirectX** application programming interfaces. **DirectPlay** is a high-level software interface between applications and communication services that makes it easy to connect games over the Internet, a **modem link**, or a network.

DirectPlay Name Server (DPNSVR): A forwarding service for enumeration requests that eliminates problems caused by conflicts between port usages for multiple **DirectPlay** applications.

DirectPlay4: A programming library that implements the IDirectPlay4 programming interface. **DirectPlay4** provides peer-to-peer, session-layer services to applications, including session lifetime management, data management, and media abstraction. **DirectPlay4** first shipped with the DirectX4 multimedia toolkit. Improved versions continued to ship up to, and including, DirectX9. **DirectPlay4** was subsequently deprecated. The **DirectPlay4** library continues to ship in current versions of Windows operating systems, but is no longer supported by Microsoft development tools.

DirectX: Microsoft **DirectX** is a collection of application programming interfaces for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.

DirectX Diagnostic (DXDiag): DXDiag.exe is a diagnostic utility included with Windows that is used to test Microsoft **DirectX** functionality, including **DirectPlay** traffic.

DXDiag Application: See **DirectX Diagnostic (DXDiag)**.

Host: The creator of the **DirectPlay** session, or the oldest peer still participating in the session after the creator has left. **Hosts** are assigned the responsibility of coordinating the addition or removal of peers in the **DirectPlay** session.

Host Migration: **Host migration** is the process that occurs when the peer that is designated as the **host** leaves the **DirectPlay** session, and the next oldest peer becomes the **host**.

Internetwork Packet Exchange (IPX): A protocol maintained by Novell's NetWare product that provides connectionless datagram delivery of messages. The **Internetwork Packet Exchange (IPX)** is based on Xerox Corporation's Internetwork Packet protocol, XNS.

Modem Link (or Modem Transport): Running the **DXDiag application** over a modem-to-modem link.

Name Table: The list of systems participating in a **DXDiag** session used both for local use and for transmission to enable peer-to-peer connectivity when additional participants join. This could also be considered the player list. It has a version number that monotonically increases with every operation that changes the **name table** content, such as adding or removing a player.

Name Table Entry: The DN_NAMETABLE_MEMBERSHIP_INFO structure along with associated strings and data buffers for an individual participant in the **DXDiag** session. These could be considered players.

Network Address Translation (NAT): A process that involves rewriting the source and/or destination addresses of IP packets as they pass through a router or firewall to enable multiple hosts on a private network to access the Internet using a single public IP address. **NAT** is also referred to as Network Masquerading, Native Address Translation, or IP Masquerading.

Poll Packet (POLL): A packet in which the sender has specified **POLL** in the packet header. **POLL** indicates that the receiver must immediately acknowledge receipt of the packet when it arrives.

Round-Trip Time (RTT): The time that it takes a packet to go from one **client** in the session to another **client** and back; a measurement of latency between peers.

Selective Acknowledgment (SACK): The DirectPlay DXDiag Usage Protocol utilizes a cumulative **acknowledgment** scheme in which all packet IDs that are at or behind the left edge of the receive window are acknowledged as received. To avoid having to resend packets that successfully arrived but were received out of order, individual packet IDs within the receive window can be selectively acknowledged using an optional bit mask.

Serial Link (or Serial Transport): Running the **DXDiag application** over a null modem cable connecting two computers.

Session Packet (SESS): A **session packet (SESS)** is associated with **client**/**server** session management. A **SESS** packet begins with a zero byte and is used for locating sessions and testing network paths. See **Transport Packet (TRANS)**.

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. **TCP** handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

Transport Packet (TRANS): A **transport packet (TRANS)** has a nonzero first byte and is further divided into command, user data, and **acknowledgment** packet types. See **Session Packet (SESS)**.

User Datagram Protocol (UDP): The connectionless protocol within **TCP**/IP that corresponds to the transport layer in the **ISO/OSI reference model**.

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.

[IANAPORT] Internet Assigned Numbers Authority, "Port Numbers", November 2006, http://www.iana.org/assignments/port-numbers

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980, http://www.ietf.org/rfc/rfc768.txt

1.2.2 Informative References

This document does not contain any informative references.

1.3 Protocol Overview (Synopsis)

The DirectPlay DXDiag Usage Protocol is designed to handle the following two basic types of network messaging for networked gaming:

- Reliable versus unreliable messaging determines if messages are guaranteed to be delivered to the target application in the event of packet loss.
- Non-sequential versus sequential messaging determines if messages are received by the target application in the same order in which they are sent in the event of packet loss or misordering.

Games use messaging for a variety of purposes, each with different demands. To support the range of messaging needs, the DirectPlay DXDiag Usage Protocol designates a message as belonging to one of four categories, depending on whether the message is reliable and/or sequential. The categories are specified by setting the PACKET_COMMAND_RELIABLE and PACKET_COMMAND_SEQUENTIAL flags in the **bCommand** field of the TRANS_USERDATA_HEADER packet header as follows:

Message category	Flags set
Reliable and Sequential	PACKET_COMMAND_RELIABLE and PACKET_COMMAND_SEQUENTIAL
Unreliable and Sequential	PACKET_COMMAND_SEQUENTIAL
Reliable and Non-sequential	PACKET_COMMAND_RELIABLE
Unreliable and Non-sequential	none

The DirectPlay DXDiag Usage Protocol enables optimizing of messaging strategy by assigning categories on a message-by-message basis.

1.3.1 How DXDiag Uses DirectPlay

DXDiag.exe is a diagnostic utility included with Windows, which is used to test Microsoft **DirectX** functionality, including **DirectPlay** traffic. The DirectPlay DXDiag Usage Protocol includes two types of DirectPlay packets: **session** and **transport**, both of which are used by **DXDiag**. These packets are transported by means of the **User Datagram Protocol (UDP)**, as specified in [RFC768].

In the DXDiag chat session, an array of identifiers is contained in a **name table**. When a new **client** joins a chat session, the client receives a name table that lists all of the clients currently in the session. When a client departs the session, the identifier for that client is removed from the name table. The name table itself is kept current through the use of a version number.

The DirectPlay DXDiag Usage Protocol is a sliding window protocol that requires the receiver to acknowledge received UDP packets before more packets are transmitted. An **acknowledgment (ACK)** can be conveyed in one of two ways: either bundled within back traffic sent from the receiver, or, when no back traffic is flowing, sent from the receiver as a dataless **Selective Acknowledgment (SACK)** packet.

When the acknowledgment is bundled within back traffic, fields within the header are used to indicate the sequence number of the next expected packet. This acknowledges that all packets with sequence numbers less than the specified number have been received correctly. If an acknowledgment is not received within a specified amount of time (which is derived from a **round-trip time (RTT)** calculated using keep-alive packets), the original packet is resent with the same sequence number as was previously assigned. If the original sender specifies **POLL** (ACK now) in the packet header, the receiver must immediately acknowledge the packet when it arrives.

1.4 Relationship to Other Protocols

The DirectPlay DXDiag Usage Protocol has a dependency on UDP and **TCP**/IP for the transport layer. As a native Windows protocol, no other protocols depend on the DirectPlay DXDiag Usage Protocol.

1.5 Prerequisites/Preconditions

All multiple-byte fields used by the DirectPlay DXDiag Usage Protocol are in **little-endian** byte order, unless otherwise noted.

1.6 Applicability Statement

The DirectPlay DXDiag Usage Protocol was designed for multiplayer network gaming, but not for other uses of client/**server** or peer-to-peer messaging. It is not recommended for file transfer or for applications with robust security needs that cannot provide them at other layers such as **IPsec**. It is also not intended as a generic replacement for TCP/IP.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

Protocol Versions: The DirectPlay DXDiag Usage Protocol supports two feature version levels in which the version value is reported by the receiver in the dwCurrentProtocolVersion field of the TRANS_COMMAND_CONNECT or TRANS_COMMAND_CONNECT_ACCEPT message:

- (Base implementation) A version value between 0x00010000 and 0x00010005.

- A version value of 0x00010006, which enables coalescence support.

Capability Negotiation: The DirectPlay DXDiag Usage Protocol detects the availability of the coalescence feature by inspecting the **dwCurrentProtocolVersion** field of the TRANS_COMMAND_CONNECT and TRANS_COMMAND_CONNECT_ACCEPT messages. If the receiver indicated a version value of 0x00010006, the sender may choose to use coalescence to minimize packet overhead when multiple payloads need to be delivered to the receiver. For more information about **coalesced payloads**, see section 2.2.15.1.

Note Prior to **DirectPlay4**, earlier versions of DirectPlay used the DirectPlay 4 Protocol, as described in [MC-DPL4CS]. These versions include:<1>

- DirectPlay (1)
- DirectPlay 2
- DirectPlay 2A
- DirectPlay 3
- DirectPlay 3A

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields in the DirectPlay DXDiag Usage Protocol.

1.9 Standards Assignments

The DirectPlay DXDiag Usage Protocol uses two well-known TCP and UDP port assignments.

Parameter	Value	Reference
TCP port for DirectPlay	6073/tcp	[IANAPORT]
UDP port for DirectPlay	6073/udp	[IANAPORT]

For the purpose and use of these port assignments, see section 3.

2 Messages

The following sections specify how DirectPlay DXDiag Usage Protocol messages are transported and DirectPlay DXDiag Usage Protocol message syntax.

2.1 Transport

The DirectPlay DXDiag Usage Protocol uses UDP, **IPX**, **serial**, or **modem** as the transport. The wire protocol format is the same for UDP and IPX. When a Serial or modem link is used, there is an extra header, as specified in section 2.2.2.

2.2 Message Syntax

2.2.1 DPNID

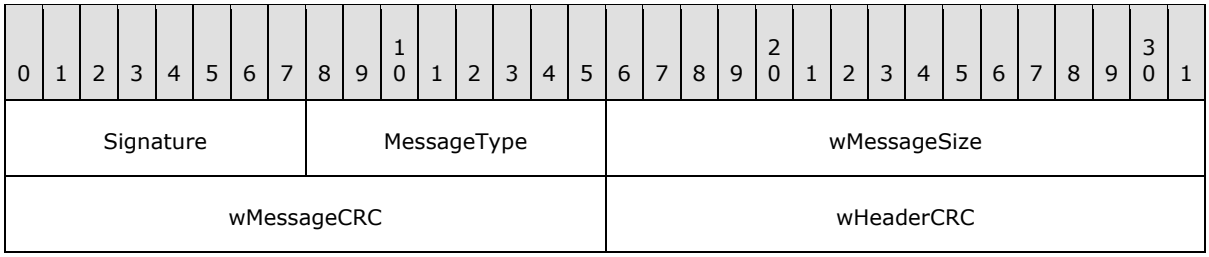
The **DPNID** identifier describes the DirectPlay network identifier for a player in a session.

This type is declared as follows:

```
typedef DWORD DPNID, *PDPNID;
```

2.2.2 _MESSAGE_HEADER

When a serial or modem link is used, the packet will contain the _MESSAGE_HEADER header in addition to the standard packet header defined in section 2.2.3. Exceptions to this are the packets SESS_ENUM_QUERY and SESS_ENUM_RESPONSE, which use _MESSAGE_HEADER in place of the first 32 bits.



Signature (1 byte): An 8-bit serial signature for the packet. This MUST be set to the value 0xCC.

MessageType (1 byte): An 8-bit token that indicates the message type. The high 4 bits MUST be one of the following values.

Value	Meaning
0x20	This message contains an enumeration response. See section 2.2.5.
0x40	This message contains a TRANS message following the header, and the low two bits MUST be ignored.

Value	Meaning
0x60	This message contains an enumeration query. See section 2.2.4. The low two bits of the **MessageType** value in an enumeration query MUST be echoed in the low two bits of the enumeration response (a message with **MessageType** value of "0x20"). The sender MAY use any identifier value it wants for the enumeration query. However, the sender SHOULD use this value to correlate queries with responses to calculate the RTT.

wMessageSize (2 bytes): The size, in bytes, of the message.

wMessageCRC (2 bytes): The **CRC**, in bytes, for the message data, which is calculated using the standardized **CRC-16-IBM algorithm**.

wHeaderCRC (2 bytes): The CRC, in bytes, for the message header, which is calculated using the standardized CRC-16-IBM algorithm.

2.2.3 DXDiag DirectPlay Packets

The DirectPlay DXDiag Usage Protocol includes two types of DirectPlay packets: session and transport, both of which are used by DXDiag. session (SESS) packets begin with a zero byte, and are used to locate sessions and to test network paths. transport (TRANS) packets have a nonzero first byte and are further divided into command, user data, and acknowledgment packet types.

A packet's purpose is determined by a combination of its command values, extended opcode values, or flag values within the packet header. For user data transport packets, the first byte that follows the 4-byte header declares the type of information included in the packet.

The DirectPlay DXDiag Usage Protocol uses the following packets.

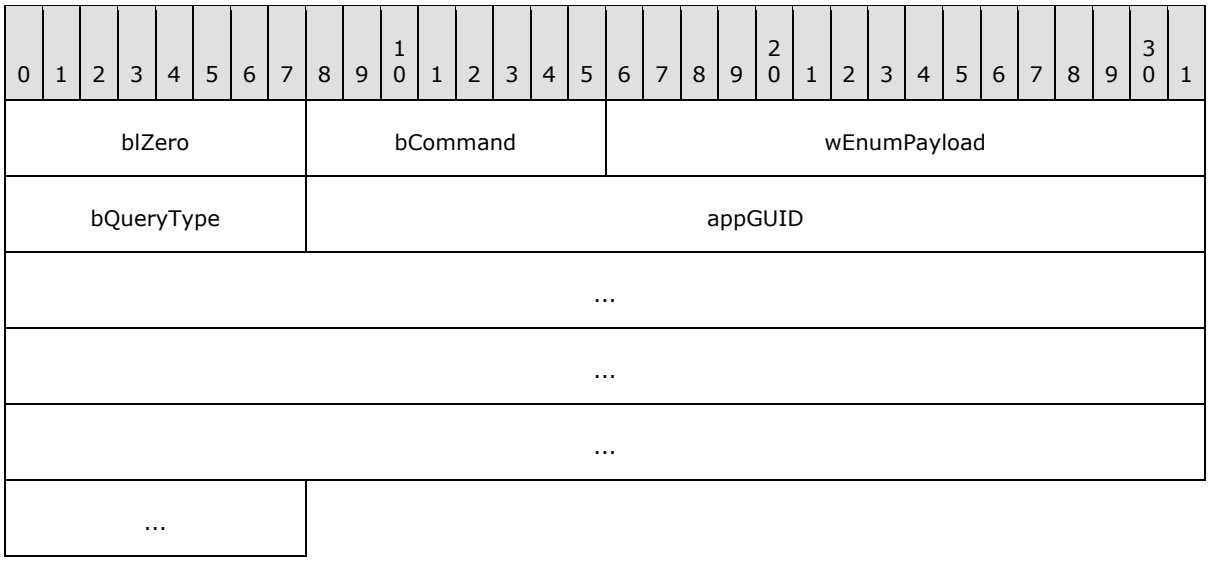
Packet	Description
SESS_ENUM_QUERY	Enumerates hosting servers.
SESS_ENUM_RESPONSE	Responds to an enumeration request.
SESS_PATH_TEST	Circumvents issues with **Network Address Translation (NAT)** devices.
TRANS_USERDATA_HEADER	Transport packet header that contains command, control, and acknowledgment information.
TRANS_USERDATA_PLAYER_CONNECT_INFO	Sends client connection information to the **host**.
TRANS_USERDATA_SEND_SESSION_INFO	Relays session information from the server to the client.
TRANS_USERDATA_ACK_SESSION_INFO	Sent from the client to the server to acknowledge the receipt of connection information.
TRANS_USERDATA_INSTRUCT_CONNECT	Instructs a client to connect to a designated client.
TRANS_USERDATA_NAMETABLE_VERSION	Specifies the version number of the name table.
TRANS_USERDATA_RESYNC_VERSION	Requests that the name table version number be resynchronized to the current version number.

Packet	Description
TRANS_USERDATA_SEND_PLAYER_DNID	Sends a user identification number to another client.
TRANS_USERDATA_KEEPALIVE	Used by DXDiag to calculate a round-trip time (RTT).
TRANS_USERDATA_CONNECT_FAILED	Indicates that a connection attempt failed.
TRANS_USERDATA_INSTRUCTED_CONNECT_FAILED	Indicates that a client was unable to carry out a server's instruction to connect to a new client.
TRANS_USERDATA_HOST_MIGRATE	Indicates that **host migration** is enabled and that the hostserver is terminating.
TRANS_USERDATA_HOST_MIGRATE_COMPLETE	Informs clients that the session-hosting responsibilities have successfully migrated from the departing host.
TRANS_USERDATA_ADD_PLAYER	Instructs clients to add the specified client to the session.
TRANS_USERDATA_DESTROY_PLAYER	Instructs clients to remove the specified user from the name table.
TRANS_USERDATA_END_OF_STREAM	Signals the disconnection of a user.
TRANS_USERDATA_REQ_INTEGRITY_CHECK	Requests that a host determine if a target client is still in the session.
TRANS_USERDATA_SEND_MESSAGE	Transmits a chat message to all other users in the session.
TRANS_COMMAND_CONNECT	Requests a connection.
TRANS_COMMAND_CONNECT_ACCEPT	Accepts a connection request.
TRANS_COMMAND_SACK	Acknowledges outstanding packets.

To reduce network traffic, several DirectPlay TRANS_USERDATA packets can be fused into a single packet using a special coalesced payload, as defined in section 2.2.15.1.

2.2.4 SESS_ENUM_QUERY

The SESS_ENUM_QUERY packet is used to enumerate hosting servers. The server sends a SESS_ENUM_RESPONSE to the client.



blZero (1 byte): Leading zero byte.

Value	Meaning
0	Leading zero that denotes a session packet.

bCommand (1 byte): Command code.

Value	Meaning
0x02	Enumeration query.

wEnumPayload (2 bytes): Logical AND between a random 2-byte enumeration key and the round-trip time (RTT) index. The index value increases with each query attempt, and the enumeration key is expected in the SESS_ENUM_RESPONSE reply from the session server.

bQueryType (1 byte): Query type.

Value	Meaning
0x01	Enumeration query with application globally unique identifier (GUID).

appGUID (16 bytes): Application GUID.

Value	Meaning
61EF80DA-691B-4247-9ADD-1C7BED2BC13E	GUID for DXDiag.

2.2.5 SESS_ENUM_RESPONSE

The SESS_ENUM_RESPONSE packet is sent from the session server to the client in response to the clients' SESS_ENUM_QUERY packet.

applicationGUID
...
...
...
sessionName (variable)
...

blZero (1 byte): Leading zero byte.

Value	Meaning
0	Leading zero that denotes a session packet.

bCommand (1 byte): Command code.

Value	Meaning
0x03	Enumeration response.

wEnumPayload (2 bytes): Logical AND between an enumeration key delivered in a previous SESS_ENUM_QUERY from a client and the round-trip time (RTT) index. The index value increases with each query attempt.

dwReplyOffset (4 bytes): Offset in bytes from **wEnumPayload** to the start of the reply. If this field is zero, the packet does not contain a reply.

dwReplySize (4 bytes): Size in bytes of the reply.

dwApDescSz (4 bytes): Application description size.

dwSessionFlags (4 bytes): Combination of the following individual flags.

Value	Meaning
0x00000001	Not a client/server session.
0x00000004	Host migration is allowed.
0x00000020	Using **DirectPlay Name Server (DPNSVR)**.
0x00000040	No password is required.
0x00000080	Enumerations are allowed.

dwMaxNPlayers (4 bytes): Maximum number of clients allowed in the session.

Value	Meaning
0x00000000	Unlimited number of clients.

dwCurrentNPlayers (4 bytes): Current number of clients in the session.

dwSessionOffset (4 bytes): Offset in bytes from **wEnumPayload** to the start of the session name.

dwSessionSize (4 bytes): Size in bytes of the session name.

dwPasswordOffset (4 bytes): Offset in bytes from **wEnumPayload** to the start of the password. This field MUST be zero to indicate that the packet does not contain a password.

dwPasswordSize (4 bytes): Size in bytes of the password. This field MUST be zero to indicate that the packet does not contain a password.

dwReservedDataOffset (4 bytes): Offset in bytes from **wEnumPayload** to the start of the reserved data. If this field is zero, the packet does not contain application reserved data.

dwReservedDataSize (4 bytes): Size in bytes of the reserved data.

dwApReservedDataOffset (4 bytes): Offset in bytes from **wEnumPayload** to the start of the application reserved data. If this field is zero, the packet does not contain application reserved data.

dwApReservedDataSize (4 bytes): Size in bytes of the application reserved data.

instanceGUID (16 bytes): Instance globally unique identifier (GUID).

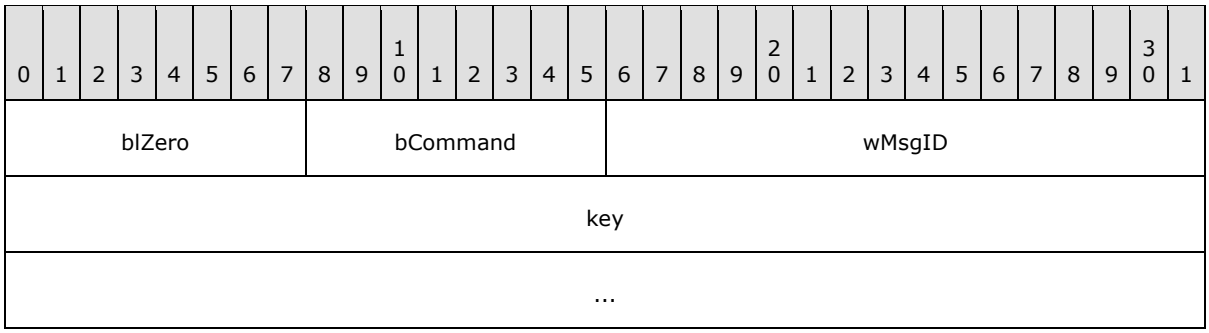
applicationGUID (16 bytes): Application GUID.

Value	Meaning
61EF80DA-691B-4247-9ADD-1C7BED2BC13E	DXDiag application GUID.

sessionName (variable): Array of Unicode characters that describe the session name with the size specified by **dwSessionSize** and the offset from the beginning of the packet specified by **dwSessionOffset**.

2.2.6 SESS_PATH_TEST

The SESS_PATH_TEST packet is used to circumvent issues with Network Address Translation (NAT) devices.



blZero (1 byte): Leading zero byte.

Value	Meaning
0	Leading zero that denotes a session packet.

bCommand (1 byte): Command code.

Value	Meaning
0x05	Path test.

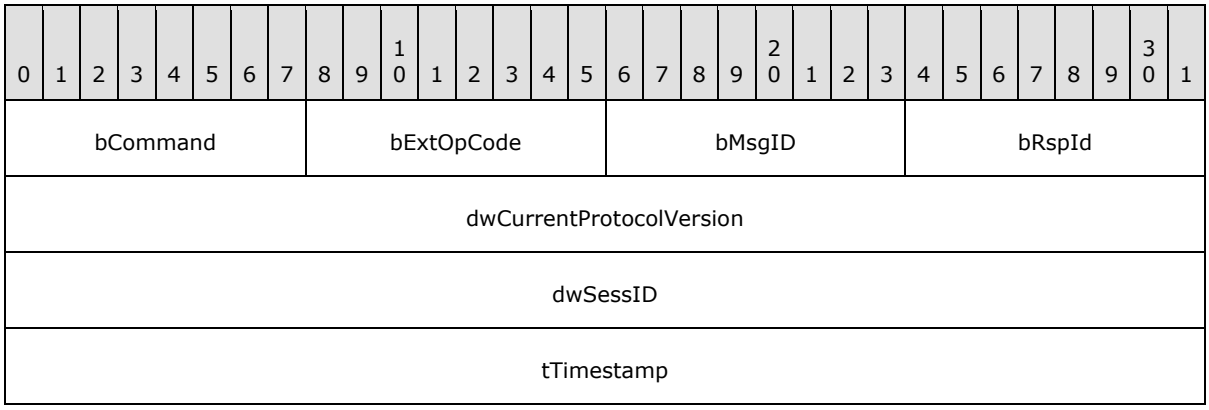
wMsgID (2 bytes): Message identification.

key (8 bytes): Unique key.

If DXDiag receives a path test from an address it is not aware of, and there are outstanding connects, DXDiag compares the unique key placed in the path test to the one specified when connecting. If they match, NAT port shuffling has occurred, and DXDiag redirects all future traffic for the connecting endpoint to the source of this path test.

2.2.7 TRANS_COMMAND_CONNECT

The TRANS_COMMAND_CONNECT packet is used to request a connection. The response is a TRANS_COMMAND_CONNECT_ACCEPT packet.



bCommand (1 byte): Command code.

Value	Meaning
0x80	CFRAME.
0x88	CFRAME, ACK now.

bExtOpCode (1 byte): Extended operation code.

Value	Meaning
0x01	Connect.

bMsgID (1 byte): Message identifier. The initial value of zero is incremented if the packet is retried.

bRspId (1 byte): Not used in the connection packet.

dwCurrentProtocolVersion (4 bytes): Version number of the requestor's DirectPlay protocol.

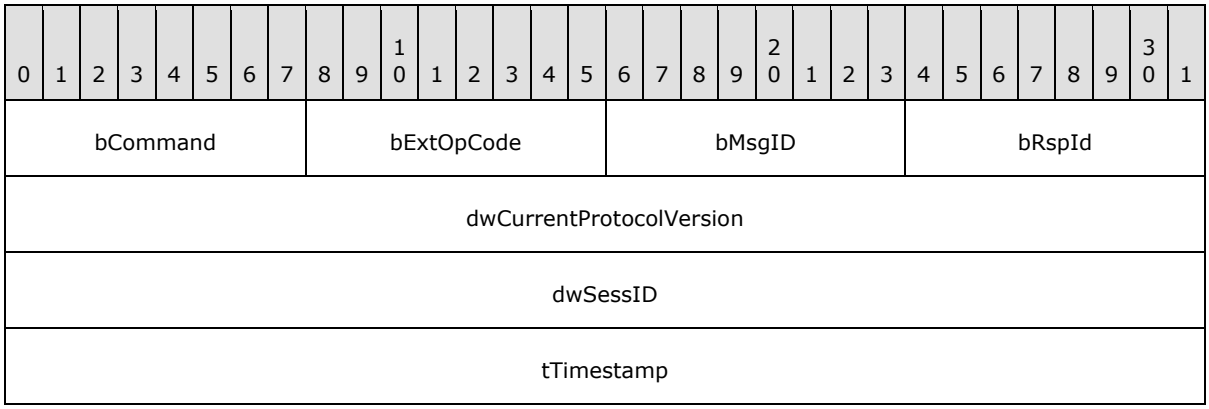
Value	Meaning
0x00010000	Protocol version number 1.0, indicating base functionality.
0x00010006	Protocol version number 1.6, indicating **coalescence** support.

dwSessID (4 bytes): Session identifier, which is dependent on the implementation.

tTimestamp (4 bytes): Requestor's system tick count.

2.2.8 TRANS_COMMAND_CONNECT_ACCEPT

The TRANS_COMMAND_CONNECT_ACCEPT packet is used to accept a connection request.



bCommand (1 byte): Command code.

Value	Meaning
0x80	CFRAME.
0x88	CFRAME, ACK now.

bExtOpCode (1 byte): Extended operation code.

Value	Meaning
FRAME_OPCODE_CONNECTED 0x02	Connected.

bMsgID (1 byte): Message identifier. The initial value of zero is incremented if the packet is retried.

bRspId (1 byte): Not used in the connection packet.

dwCurrentProtocolVersion (4 bytes): Version number of the requestor's DirectPlay protocol.

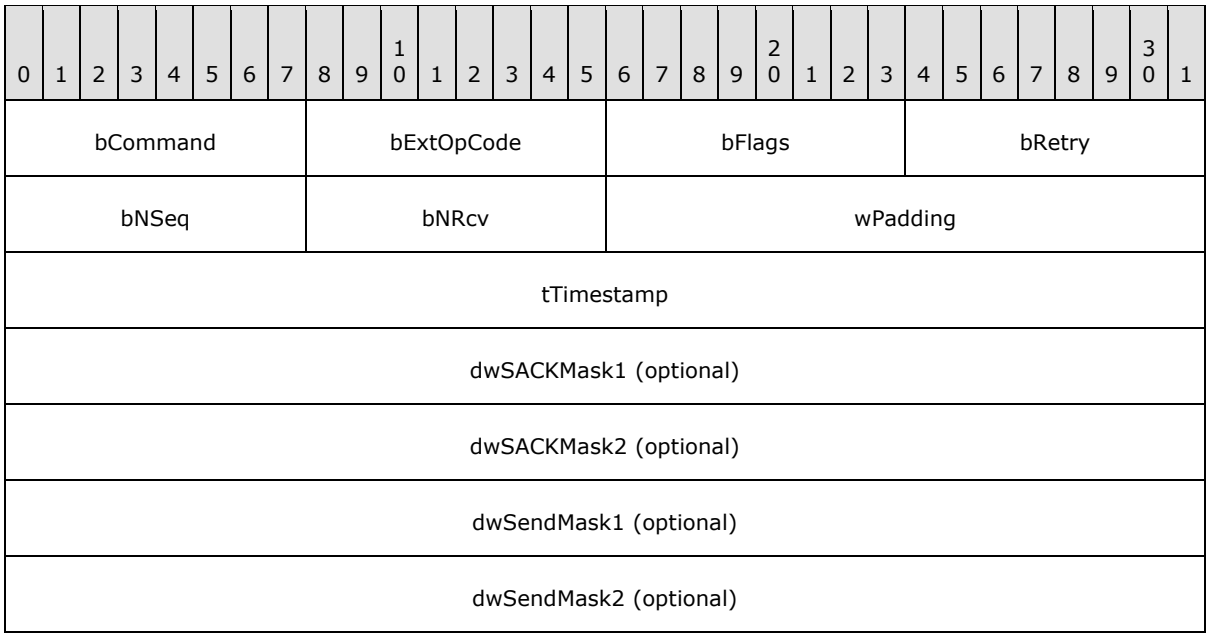
Value	Meaning
0x00010000	Protocol version number 1.0, indicating base functionality.
0x00010006	Protocol version number 1.6, indicating coalescence support.

dwSessID (4 bytes): Session identifier specified in the request packet.

tTimestamp (4 bytes): Sender's system tick count.

2.2.9 TRANS_COMMAND_SACK

The TRANS_COMMAND_SACK packet is used to acknowledge outstanding packets. Packet acknowledgment is typically bundled in all user data packets using the **bSeq** and **bNRec** fields found in the TRANS_USERDATA_HEADER; however, the TRANS_COMMAND_SACK packet is used when a dedicated acknowledgment is requested (that is, when the PACKET_COMMAND_POLL bit in the **bCommand** header field is set) or when no user data remains for further bundled acknowledgments.



bCommand (1 byte): Command code.

Value	Meaning
0x80	CFRAME.
0x88	CFRAME, ACK now.

bExtOpCode (1 byte): Extended operation code.

Value	Meaning
FRAME_OPCODE_SACK 0x06	Selective acknowledgment (SACK).

bFlags (1 byte): Status flags. The value can be one or more of the following.

Value	Meaning
SACK_FLAGS_RESPONSE 0x01	**bRetry** field is valid.
SACK_FLAGS_SACK_MASK1 0x02	Low 32 bits of the SACK mask are present.
SACK_FLAGS_SACK_MASK2 0x04	High 32 bits of the SACK mask are present.
SACK_FLAGS_SEND_MASK1 0x08	Low 32 bits of the send mask are present.
SACK_FLAGS_SEND_MASK2 0x10	High 32 bits of the send mask are present.

bRetry (1 byte): Boolean indicating if the last received packet was a retry. This value MUST be ignored if SACK_FLAGS_RESPONSE is not set. Otherwise, the value SHOULD be 0 if the last received DFRAME for the connection was not marked as a retry; otherwise, it SHOULD be nonzero. Recipients MUST NOT require any particular bit (or bits) to be set in the nonzero case, only that at least one is set.

bNSeq (1 byte): SACK packets do not have sequence numbers of their own. This field represents the sequence number of the next data frame to send.

bNRcv (1 byte): Expected sequence number of the next packet received. If the SACK_FLAGS_SACK_MASK1 flag is set, the **bNRcv** field is supplemented with an additional **DWORD** bitmask field that selectively acknowledges frames with sequence numbers higher than **bNRcv**.

wPadding (2 bytes): Unused. This SHOULD be set to 0, and MUST be ignored by the recipient.

tTimestamp (4 bytes): Local tick count used as a time stamp.

dwSACKMask1 (4 bytes): Optional low 32 bits of the selective acknowledgment mask in little-endian byte order. The existence of this field in the packet is dependent on the **bFlags** field having SACK_FLAGS_SACK_MASK1 set.

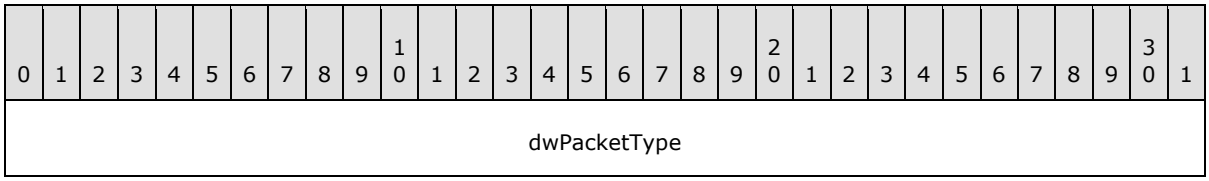
dwSACKMask2 (4 bytes): Optional high 32 bits of the selective acknowledgment mask in little-endian byte order. The existence of this field in the packet is dependent on the **bFlags** field having SACK_FLAGS_SACK_MASK2 set.

dwSendMask1 (4 bytes): Optional low 32 bits of the send mask in little-endian byte order. The existence of this field in the packet is dependent on the **bFlags** field having SACK_FLAGS_SEND_MASK1 set.

dwSendMask2 (4 bytes): Optional high 32 bits of the send mask in little-endian byte order. The existence of this field in the packet is dependent on the **bFlags** field having SACK_FLAGS_SEND_MASK2 set.

2.2.10 TRANS_USERDATA_ACK_SESSION_INFO

The TRANS_USERDATA_ACK_SESSION_INFO packet is sent from the client to the server to acknowledge the receipt of connection information. This packet contains no user data beyond the packet type field, and begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_ACK_SESSION_INFO 0x000000C3	Acknowledges the receipt of session information.

2.2.11 TRANS_USERDATA_ADD_PLAYER

The TRANS_USERDATA_ADD_PLAYER packet instructs clients to add a specified client to the session. This packet begins with a TRANS_USERDATA_HEADER.

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
dwPacketType																															
dpnid																															
dpnidOwner																															
dwFlags																															
dwVersion																															
dwVersionNotUsed																															
dwDNETClientVersion																															
dwNameOffset																															
dwNameSize																															
dwDataOffset																															
dwDataSize																															
dwURLOffset																															
dwURLSize																															
data (variable)																															
...																															
url (variable)																															
...																															
name (variable)																															

...

dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_ADD_PLAYER 0x000000D0	Instructs clients to add the specified client to the session.

dpnid (4 bytes): Identifier of the client to add.

dpnidOwner (4 bytes): Identifier of the session owner.

dwFlags (4 bytes): Application flags.

dwVersion (4 bytes): Current name table version number.

dwVersionNotUsed (4 bytes): Not used.

dwDNETClientVersion (4 bytes): DirectPlay version of the client being added to the chat session.

Value	Meaning
00000001	DirectX 8.0
00000002	DirectX 8.1
00000003	Pocket PC
00000004	Not used.
00000005	Windows Server 2003
00000006	DirectX 8.2
00000007	DirectX 9.0

dwNameOffset (4 bytes): Offset from the header to the client name. If this field is zero, the packet does not include the client name.

dwNameSize (4 bytes): Size, in bytes, of the name.

dwDataOffset (4 bytes): Offset from the header to client data. If this field is zero, the packet does not include client data.

dwDataSize (4 bytes): Size, in bytes, of the client data.

dwURLOffset (4 bytes): Offset from header to the client uniform resource locator (URL). If this field is zero, the packet does not include the client URL.

dwURLSize (4 bytes): Size, in bytes, of the connecting client's URL address.

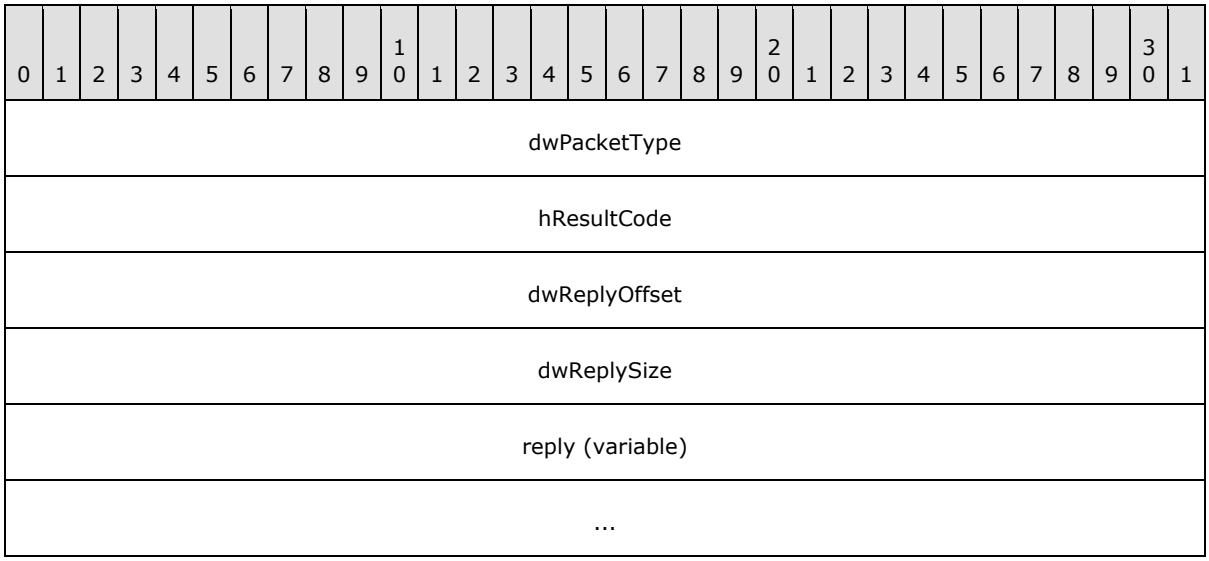
data (variable): Array of characters that contain user data, including the NULL termination character.

url (variable): Array of characters that contain the client URL, including the NULL termination character.

name (variable): Array of Unicode characters that contain the client name, including the NULL termination character.

2.2.12 TRANS_USERDATA_CONNECT_FAILED

The TRANS_USERDATA_CONNECT_FAILED packet indicates that a connection attempt failed. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_CONNECT_FAILED 0x000000C5	Connection attempt failed.

hResultCode (4 bytes): Result code.

dwReplyOffset (4 bytes): Offset of the **reply[]** field from the start of the packet header.

dwReplySize (4 bytes): Size in bytes of the data in the **reply[]** field.

reply (variable): Array of characters that contains a reply message identifying the connection failure, including the NULL termination character.

2.2.13 TRANS_USERDATA_DESTROY_PLAYER

The TRANS_USERDATA_DESTROY_PLAYER packet instructs the client to remove a specified user from the name table. This packet begins with a TRANS_USERDATA_HEADER.

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
dwPacketType																															
dpnidLeaving																															
dwVersion																															
dwVersionNotUsed																															
dwDestroyReason																															

dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_DESTROY_PLAYER 0x000000D1	Instructs the client to remove the specified user from the name table.

dpnidLeaving (4 bytes): Identifier of the client or server to remove from the name table.

dwVersion (4 bytes): Current name table version number.

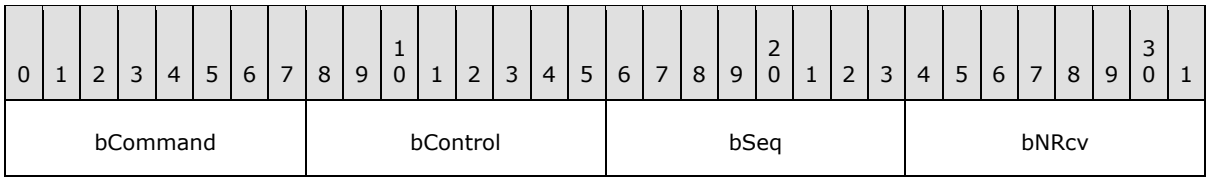
dwVersionNotUsed (4 bytes): Not used.

dwDestroyReason (4 bytes): Reason for terminating the specified client or server.

Value	Meaning
DPNDESTROYPLAYERREASON_NORMAL 0x00000001	Client or server is leaving.
DPNDESTROYPLAYERREASON_CONNECTIONLOST 0x00000002	Connection was lost.
DPNDESTROYPLAYERREASON_SESSIONTERMINATED 0x00000003	Session was terminated.
DPNDESTROYPLAYERREASON_HOSTDESTROYEDPLAYER 0x00000004	Server removed the client.

2.2.14 TRANS_USERDATA_END_OF_STREAM

The TRANS_USERDATA_END_OF_STREAM packet is used to signal the disconnection of a user. This packet is identified by the **PACKET_CONTROL_END_STREAM** bit set in the **bControl** field in the TRANS_USERDATA_HEADER (see below). This packet consists of only the packet header.



bCommand (1 byte): Command field.

Value	Meaning
PACKET_COMMAND_DATA 0x01	Frame contains user data.
PACKET_COMMAND_RELIABLE 0x02	Frame should be delivered reliably.
PACKET_COMMAND_SEQUENTIAL 0x04	Frame should be indicated sequentially.
PACKET_COMMAND_POLL 0x08	Partner should acknowledge immediately.
PACKET_COMMAND_NEW_MSG 0x10	Data frame is first in message.
PACKET_COMMAND_END_MSG 0x20	Data frame is last in message.
PACKET_COMMAND_USER_1 0x40	First user-controlled flag.
PACKET_COMMAND_CFRAME 0x80	Set high bit on command frames.

bControl (1 byte): Control field.

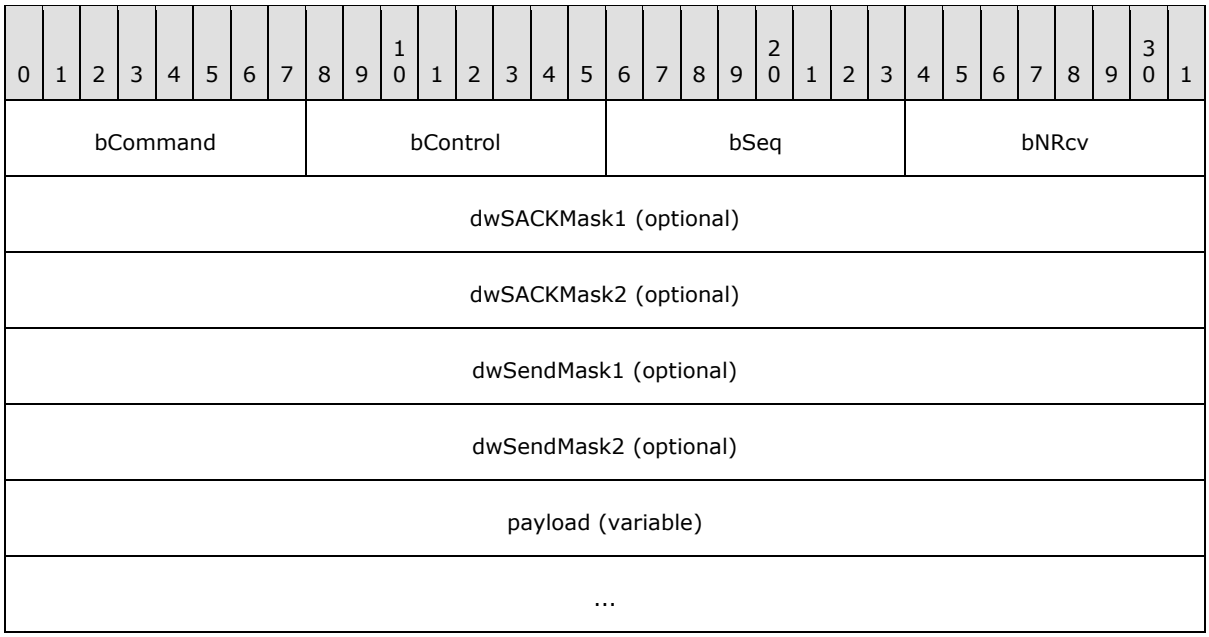
Value	Meaning
PACKET_CONTROL_END_STREAM 0x08	Last packet in the stream; indicates disconnect.

bSeq (1 byte): Sequence number of the packet.

bNRcv (1 byte): Expected sequence number of the next packet received.

2.2.15 TRANS_USERDATA_HEADER

The TRANS_USERDATA_HEADER is a transport packet header that contains command, control, and acknowledgment information. It is included with all TRANS_USERDATA DirectPlay packets.



bCommand (1 byte): Command field. Two or more of the following flags can be combined to form complex values.

Value	Meaning
PACKET_COMMAND_DATA 0x01	Frame contains user data.
PACKET_COMMAND_RELIABLE 0x02	Frame should be delivered reliably and requires a packet acknowledgment.
PACKET_COMMAND_SEQUENTIAL 0x04	Frame should be indicated sequentially.
PACKET_COMMAND_POLL 0x08	Partner should acknowledge immediately.
PACKET_COMMAND_NEW_MSG 0x10	Data frame is first in message.
PACKET_COMMAND_END_MSG 0x20	Data frame is last in message.
PACKET_COMMAND_USER_1 0x40	First user-controlled flag.
PACKET_COMMAND_CFRAME 0x80	Set high bit on command frames.

bControl (1 byte): Control field. Two or more of the following flags can be combined to form complex values.

Value	Meaning
PACKET_CONTROL_RETRY 0x01	Indicates if the frame is a retry for this sequence number.
PACKET_CONTROL_KEEPALIVE_OR_CORRELATE 0x02	For versions 0x00010005 and higher, this flag indicates that the frame is a keep-alive frame, and applies only to DirectX version 9.0 and later. When the version is lower than 0x00010005, this flag requests a dedicated acknowledgment from the receiver, and applies only to versions of DirectX prior to version 9.0. For information about versions, see section 1.7.
PACKET_CONTROL_COALESCE 0x04	Packet contains multiple fused packets. Not supported by DirectPlay version 8.0.
PACKET_CONTROL_END_STREAM 0x08	Last packet in the stream; indicates disconnect.
PACKET_CONTROL_SACK1 0x10	Low 32 bits of the selective acknowledgment (SACK) mask are present.
PACKET_CONTROL_SACK2 0x20	High 32 bits of the SACK mask are present.
PACKET_CONTROL_SEND1 0x40	Low 32 bits of the cancel-send mask are present.
PACKET_CONTROL_SEND2 0x80	High 32 bits of the cancel-send mask are present.
PACKET_CONTROL_VARIABLE_MASKS 0xF0	All four packet control mask bits are present.

bSeq (1 byte): Sequence number of the packet.

bNRcv (1 byte): Expected sequence number of the next packet received.

dwSACKMask1 (4 bytes): Optional low 32 bits of the selective acknowledgment mask in little-endian byte order. The existence of this field in the packet is dependent on the **bFlags** field having SACK_FLAGS_SACK_MASK1 set in the TRANS_COMMAND_SACK packet.

dwSACKMask2 (4 bytes): Optional high 32 bits of the selective acknowledgment mask in little-endian byte order. The existence of this field in the packet is dependent on **bFlags** field having SACK_FLAGS_SACK_MASK2 set in the TRANS_COMMAND_SACK packet.

dwSendMask1 (4 bytes): Optional low 32 bits of the send mask in little-endian byte order. The existence of this field in the packet is dependent on **bFlags** field having SACK_FLAGS_SEND_MASK1 set in the TRANS_COMMAND_SACK packet.

dwSendMask2 (4 bytes): Optional high 32 bits of the send mask in little-endian byte order. The existence of this field in the packet is dependent on **bFlags** field having SACK_FLAGS_SEND_MASK2 set in the TRANS_COMMAND_SACK packet.

payload (variable): Consumer payload data. The payload size is the total UDP frame size minus the amount of data consumed by data frame headers up to this point. If the PACKET_CONTROL_COALESCE flag is set, the payload is not a single message or portion of a

message, but is instead organized according to the coalesced payload format, as specified in section 2.2.15.1.

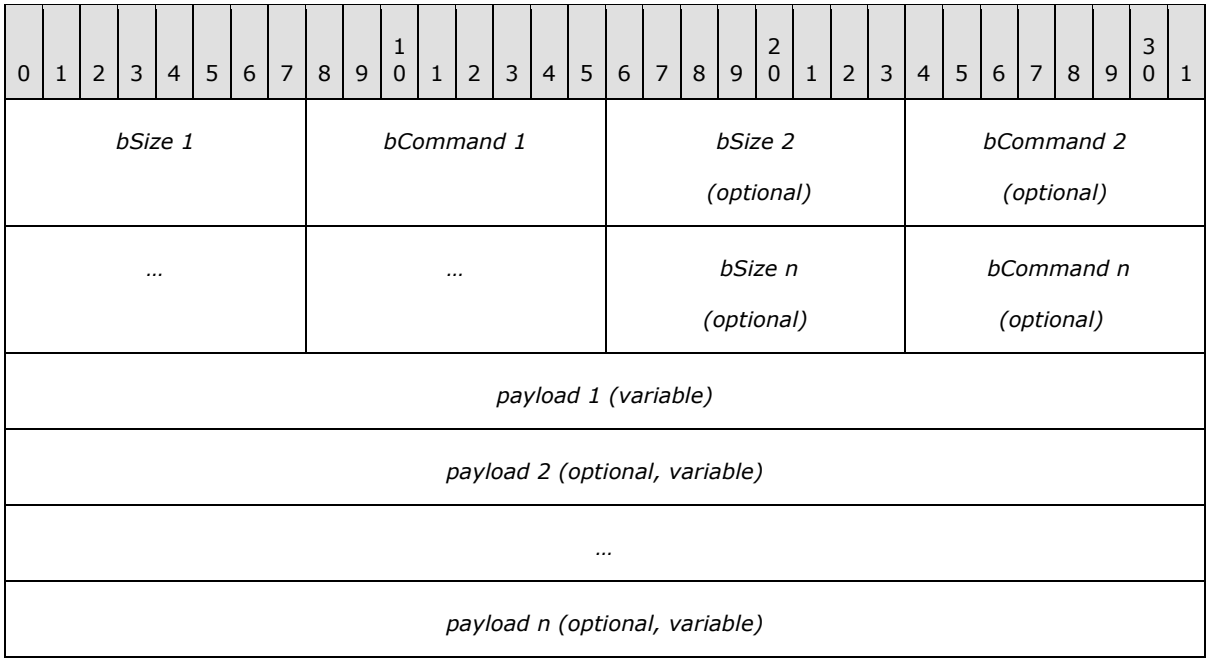
2.2.15.1 Coalesced Payloads

Coalesced payloads are a special form of payload within standard data frames. When the PACKET_CONTROL_COALESCE flag is set on the outer data frame header **bControl** field of the TRANS_USERDATA_HEADER packet, the payload is interpreted using this format. Frames with coalesced payloads MUST have the PACKET_COMMAND_NEW_MSG and PACKET_COMMAND_END_MSG flags set on the outer data frame header **bCommand** field.

Between 1 and 32 two-byte headers are placed at the beginning of the buffer. The buffer MUST NOT contain more than 32 coalesce headers. If there is an odd number of coalesce headers, two extra bytes of zero padding MUST be added at the end to align the subsequent data on a 32-bit boundary. The last non-padded coalesce header MUST have the PACKET_COMMAND_END_COALESCE flag set in its **bCommand** field.

Following the headers are 1 to 32 payloads where the sizes of each are indicated in the corresponding headers that were added in the same order. If the payload size is not a multiple of 32 bits, and it is not the last payload in the message, one to three bytes of zero padding MUST be added to align the beginning of the next payload on a 32-bit boundary. The sizes indicated in the coalesce headers MUST NOT include any padding so as to preserve the message size as originally sent. The receiver MUST infer alignment padding when processing the payloads, and SHOULD indicate the messages to the consumer using the unpadded size.

The following is an example of a standard data frame for a coalesced payload.



In the example above, the following field types are represented.

bSize 1 through bSize *n*: Least significant 8 bits of the size of the coalesced payload. The value is combined with the optional PACKET_COMMAND_COALESCE_BIG_1, PACKET_COMMAND_COALESCE_BIG_2, and PACKET_COMMAND_COALESCE_BIG_3 flags to

determine the actual size of the payload. This MUST NOT be larger than what can fit in a standard data frame, including any size already used to store previous coalesce headers and payloads.

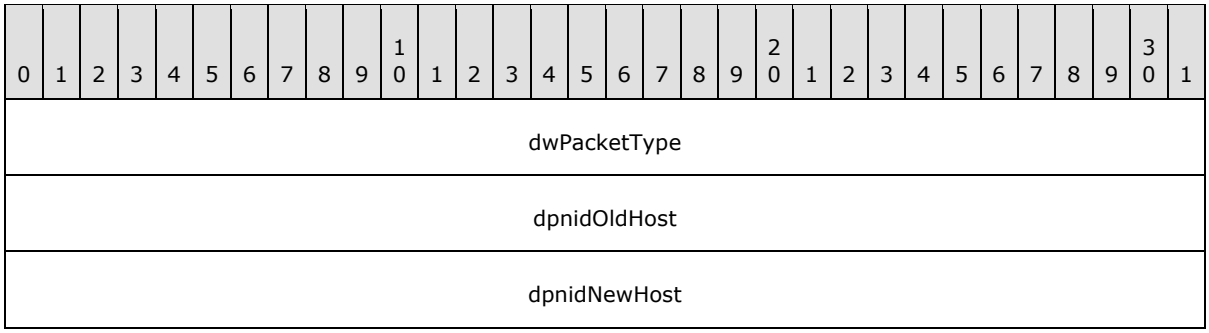
bCommand 1 through bCommand *n*: Command field for the coalesced message. The PACKET_COMMAND_DATA flag MUST be set. If a TRANS_USERDATA_KEEPALIVE packet, the PACKET_COMMAND_RELIABLE, PACKET_COMMAND_SEQUENTIAL, and PACKET_COMMAND_END_MSG flags MUST be set. All other flags are optional.

Value	Meaning
0x01	PACKET_COMMAND_END_COALESCE (this is the final coalesced payload in the frame).
0x02	PACKET_COMMAND_RELIABLE (payload SHOULD be delivered reliably).
0x04	PACKET_COMMAND_SEQUENTIAL (payload SHOULD be indicated sequentially).
0x08	PACKET_COMMAND_COALESCE_BIG_1 (bit 9 of the coalesced payload size).
0x10	PACKET_COMMAND_COALESCE_BIG_2 (bit 10 of the coalesced payload size).
0x20	PACKET_COMMAND_COALESCE_BIG_3 (bit 11 of the coalesced payload size; the most significant bit).
0x40	PACKET_COMMAND_USER_1 (payload is an internal session management message).

payload 1 through payload *n*: Consumer payload data.

2.2.16 TRANS_USERDATA_HOST_MIGRATE

The TRANS_USERDATA_HOST_MIGRATE packet indicates that host migration is enabled, and the host server is terminating. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

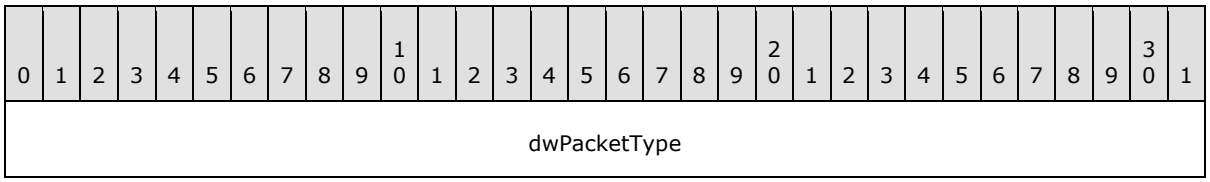
Value	Meaning
TRANS_USERDATA_HOST_MIGRATE 0x000000CD	Indicates that the host migration procedure has started.

dpnidOldHost (4 bytes): Identifier for the old host.

dpnidNewHost (4 bytes): Identifier for the new host.

2.2.17 TRANS_USERDATA_HOST_MIGRATE_COMPLETE

The TRANS_USERDATA_HOST_MIGRATE_COMPLETE packet informs clients that the session-hosting responsibilities have successfully migrated from the departing old host. This packet begins with a TRANS_USERDATA_HEADER and contains no user data.

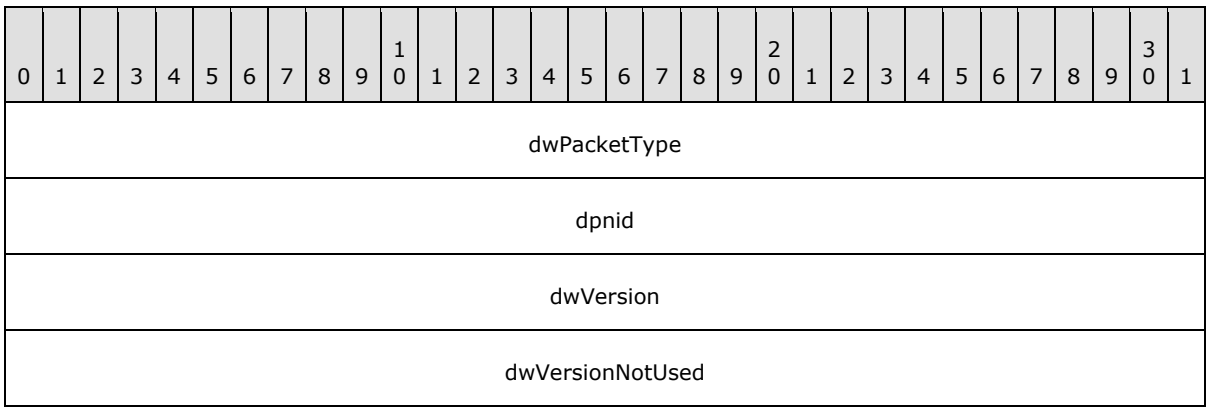


dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_HOST_MIGRATE_COMPLETE 0x000000CE	Informs clients that the session-hosting responsibilities have successfully migrated from the departing old host.

2.2.18 TRANS_USERDATA_INSTRUCT_CONNECT

The TRANS_USERDATA_INSTRUCT_CONNECT packet instructs a client to connect to a designated client. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_INSTRUCT_CONNECT 0x000000C6	Instructs a client to connect to a designated client.

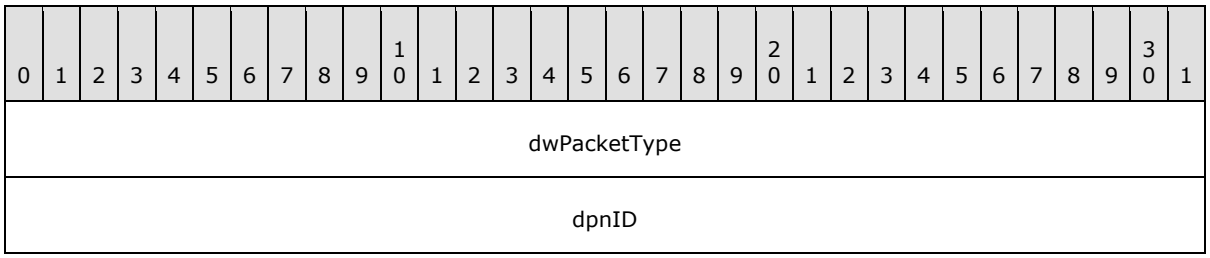
dpnid (4 bytes): Identifier of the designated client to which the connection is being made.

dwVersion (4 bytes): Current version of the name table.

dwVersionNotUsed (4 bytes): Not used.

2.2.19 TRANS_USERDATA_INSTRUCTED_CONNECT_FAILED

The TRANS_USERDATA_INSTRUCTED_CONNECT_FAILED packet indicates that a client was unable to carry out a server instruction to connect to a new client. This packet begins with a TRANS_USERDATA_HEADER.



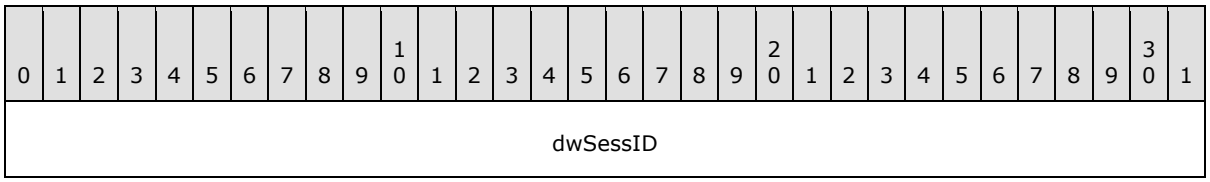
dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_INSTRUCTED_CONNECT_FAILED 0x000000C7	Indicates that a client was unable to carry out a host's instruction to connect to a new client.

dpnID (4 bytes): Identifier for the client.

2.2.20 TRANS_USERDATA_KEEPALIVE

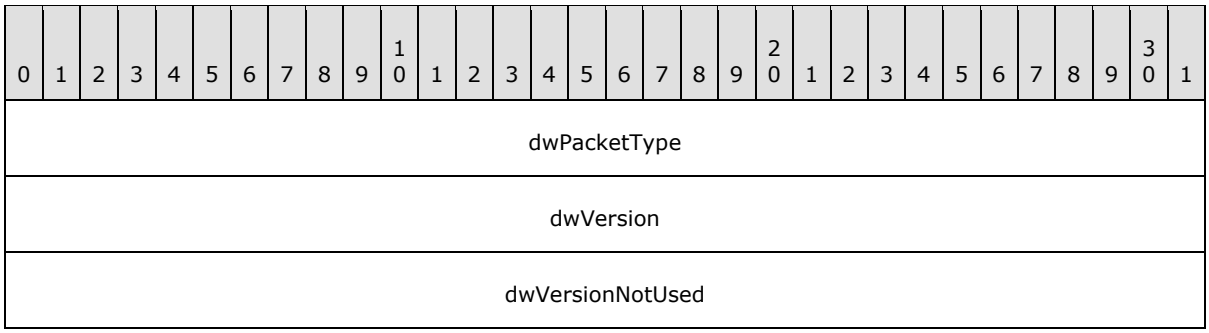
The TRANS_USERDATA_KEEPALIVE packet is used by DXDiag to calculate a round-trip time (RTT). This packet is identified by the **PACKET_CONTROL_KEEPALIVE** bit, which is set in the **bControl** field found in the TRANS_USERDATA_HEADER at the beginning of the packet.



dwSessID (4 bytes): Session identifier attached to the application.

2.2.21 TRANS_USERDATA_NAMETABLE_VERSION

The TRANS_USERDATA_NAMETABLE_VERSION packet specifies the version number of the name table. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

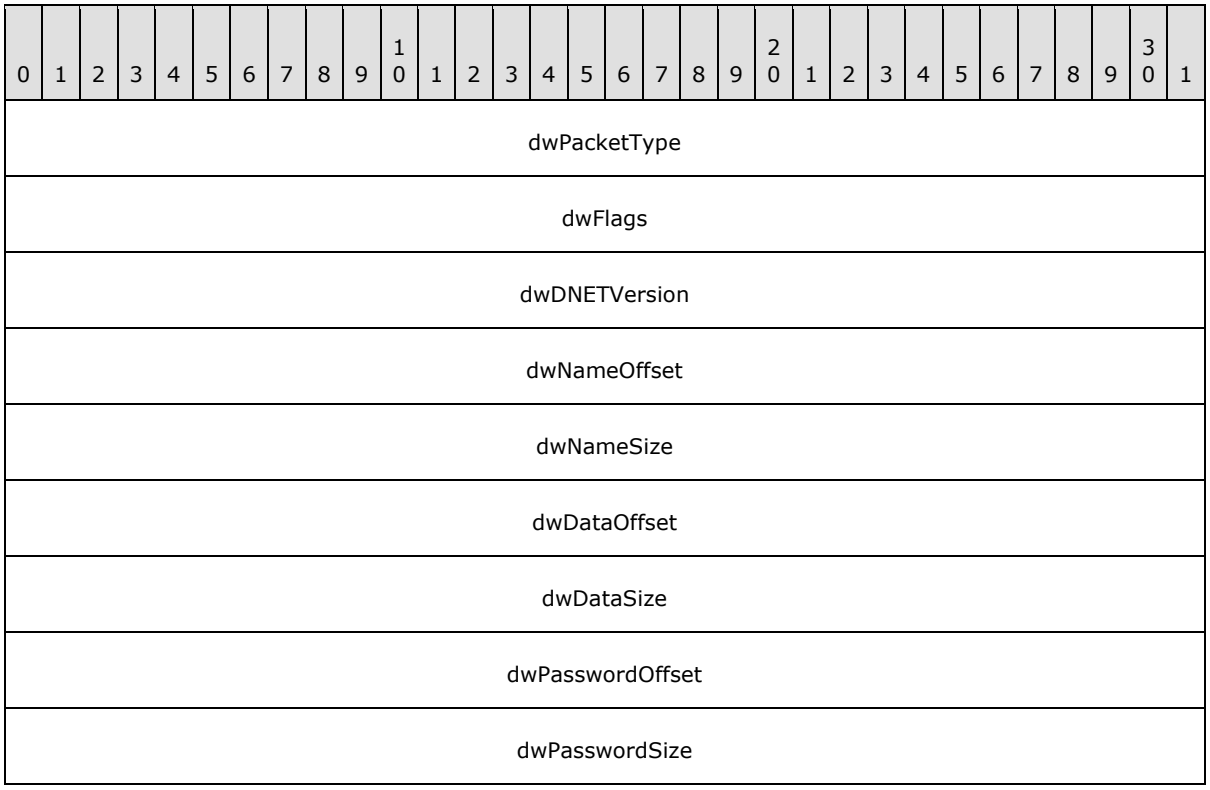
Value	Meaning
TRANS_USERDATA_NAMETABLE_VERSION 0x000000C9	Specifies the version number of the name table.

dwVersion (4 bytes): Current name table version number.

dwVersionNotUsed (4 bytes): Not used.

2.2.22 TRANS_USERDATA_PLAYER_CONNECT_INFO

The TRANS_USERDATA_PLAYER_CONNECT_INFO packet is used to send client connection information to the host. This packet begins with a TRANS_USERDATA_HEADER.



dwConnectDataOffset
dwConnectDataSize
dwURLOffset
dwURLSize
guidInstance
...
...
...
guidApplication
...
...
...
dwAlternateAddressDataOffset
dwAlternateAddressDataSize
alternateAddressData (variable)
...
name (variable)
...
data (variable)
...
connectData (variable)

...
url (variable)
...

dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_PLAYER_CONNECT_INFO 0x000000C1	Sends client connection information to the host server.

dwFlags (4 bytes): Connect flags.

Value	Meaning
DPNPLAYER_LOCAL 0x00000002	Connecting locally.
DPNPLAYER_HOST 0x00000004	Connecting to a host.

dwDNETVersion (4 bytes): DirectPlay version.

Value	Meaning
0x00000001	DirectX version 8.0
0x00000002	DirectX version 8.1
0x00000003	Pocket PC
0x00000004	Not used.
0x00000005	Windows Server 2003
0x00000006	DirectX version 8.2
0x00000007	DirectX version 9.0

dwNameOffset (4 bytes): Offset from the packet header of the connecting client's name field.

dwNameSize (4 bytes): Size, in bytes, of the data in the **name[]** field.

dwDataOffset (4 bytes): Offset from the packet header of the **data[]** field. If **dwDataOffset** is zero, the packet does not include client data.

dwDataSize (4 bytes): Size, in bytes, of the **data[]** field.

dwPasswordOffset (4 bytes): Offset from the packet header to the start of the **password[]**. This MUST be set to zero to indicate that the packet does not include a **password**.

dwPasswordSize (4 bytes): Size, in bytes, of the **password**. This MUST be set to zero to indicate that the packet does not include a **password**.

dwConnectDataOffset (4 bytes): Offset from the packet header to the **connectData[]** field. If **dwConnectDataOffset** is zero, the packet does not include connection data.

dwConnectDataSize (4 bytes): Size, in bytes, of the **connectData[]** field.

dwURLOffset (4 bytes): Offset from the packet header to the **url[]** field. If **dwURLOffset** is zero, the packet does not include the client URL.

dwURLSize (4 bytes): Size, in bytes, of the **url[]** field.

guidInstance (16 bytes): The instance globally unique identifier (GUID) of the connecting client.

guidApplication (16 bytes): Application GUID.

Value	Meaning
61EF80DA-691B-4247-9ADD-1C7BED2BC13E	DXDiag application GUID.

dwAlternateAddressDataOffset (4 bytes): Offset from the packet header to the **alternateAddressData[]** field. If **dwAlternateAddressDataOffset** is zero, the packet does not include the alternate address data. This field is used in DirectPlay version 9.

dwAlternateAddressDataSize (4 bytes): Size, in bytes, of the **alternateAddressData[]** field. This field is used in DirectPlay version 9.

alternateAddressData (variable): Alternative address data used to connect the client. This field's position is determined by **dwAlternateAddressDataOffset** and the size stated in **dwAlternateAddressDataSize**. This field is used in DirectPlay version 9.

name (variable): Zero-terminated character array that contains the client name. This field's position is determined by **dwNameOffset** and the size stated in **dwNameSize**.

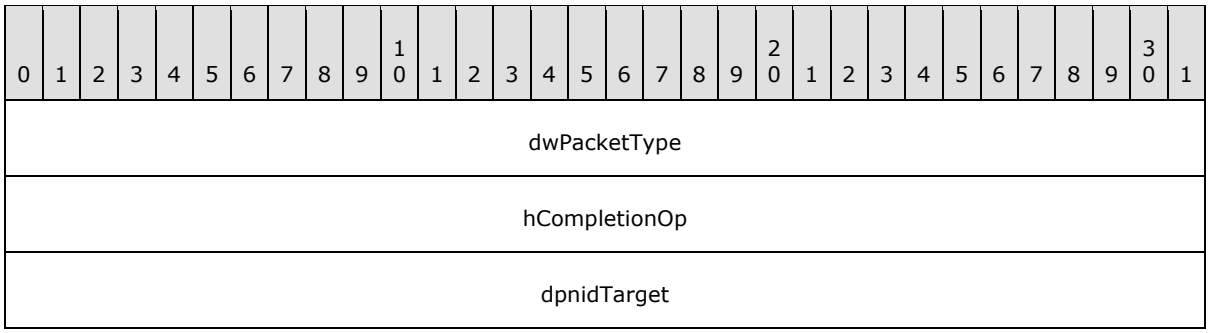
data (variable): Zero-terminated character array that contains the client data. This field's position is determined by **dwDataOffset** and the size stated in **dwDataSize**.

connectData (variable): Zero-terminated character array that contains the connection data. This field's position is determined by **dwConnectDataOffset** and the size stated in **dwConnectDataSize**.

url (variable): Zero-terminated character array that contains the client URL. This field's position is determined by **dwURLOffset** and the size stated in **dwURLSize**.

2.2.23 TRANS_USERDATA_REQ_INTEGRITY_CHECK

The TRANS_USERDATA_REQ_INTEGRITY_CHECK packet requests that a host determine if a target client is still in the session. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

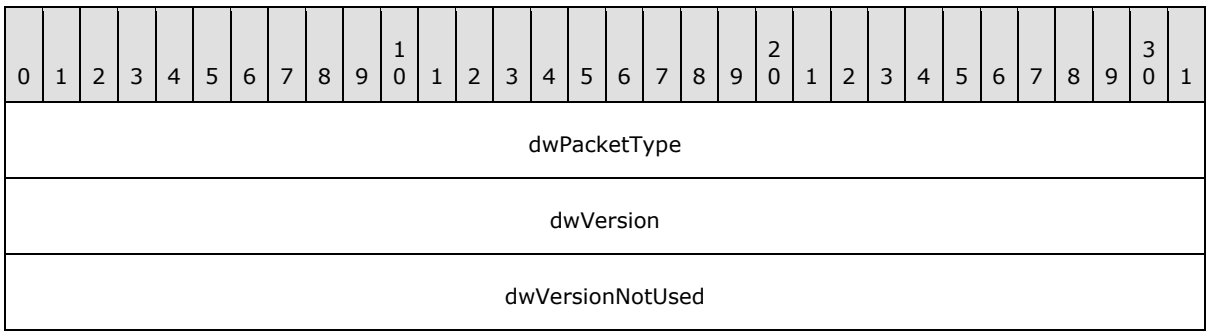
Value	Meaning
TRANS_USERDATA_REQ_INTEGRITY_CHECK 0x000000E2	Requests that the host server determine if a target client is still in the session.

hCompletionOp (4 bytes): Handle to the completed operation.

dpnidTarget (4 bytes): Identifier of the selected target client.

2.2.24 TRANS_USERDATA_RESYNC_VERSION

The TRANS_USERDATA_RESYNC_VERSION packet is used to request that the name table version number be resynchronized to the current version number. This packet begins with a TRANS_USERDATA_HEADER.



dwPacketType (4 bytes): Packet type field.

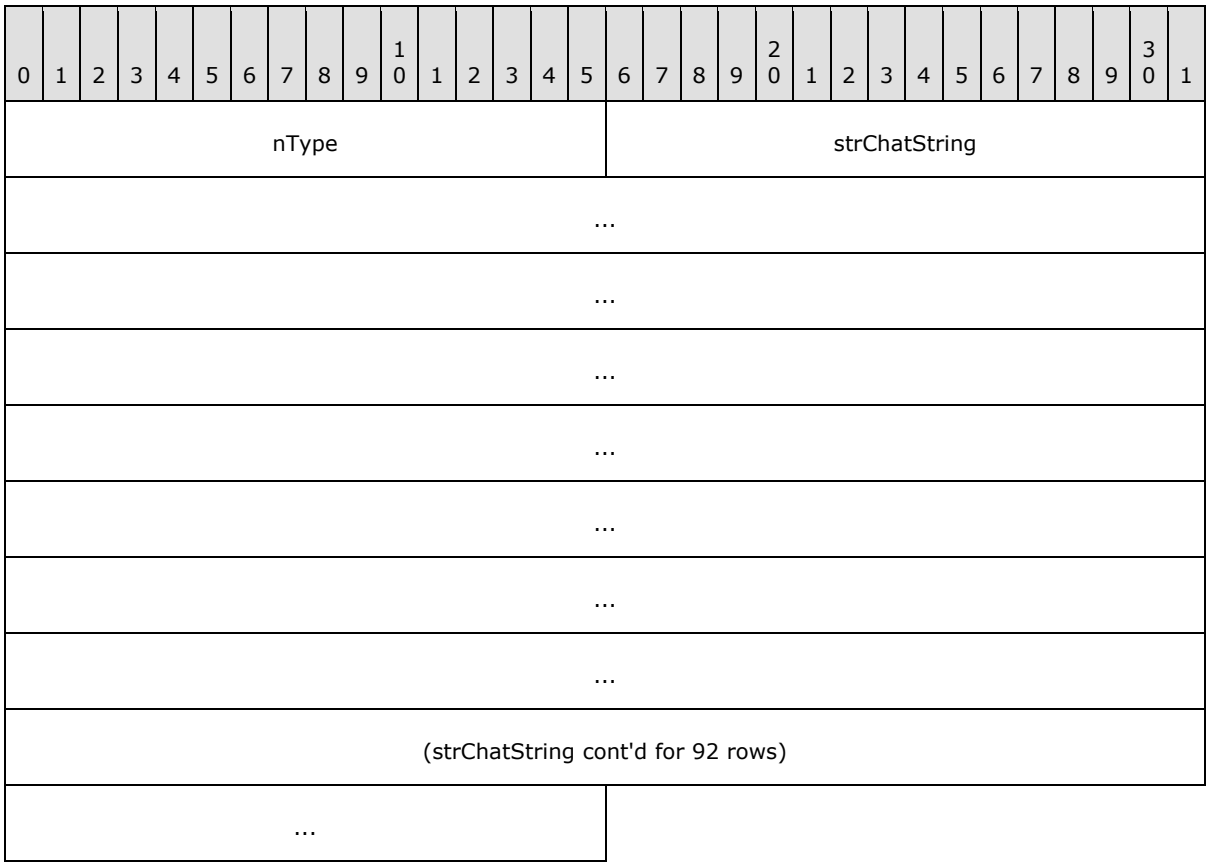
Value	Meaning
TRANS_USERDATA_RESYNC_VERSION 0x000000CA	Requests that the protocol version number be resynchronized to the current version number.

dwVersion (4 bytes): Current name table version number.

dwVersionNotUsed (4 bytes): Not used.

2.2.25 TRANS_USERDATA_SEND_MESSAGE

The TRANS_USERDATA_SEND_MESSAGE packet transmits a chat message to all other users in a chat session. This packet begins with a TRANS_USERDATA_HEADER and does not contain a **dwPacketType** identification field.



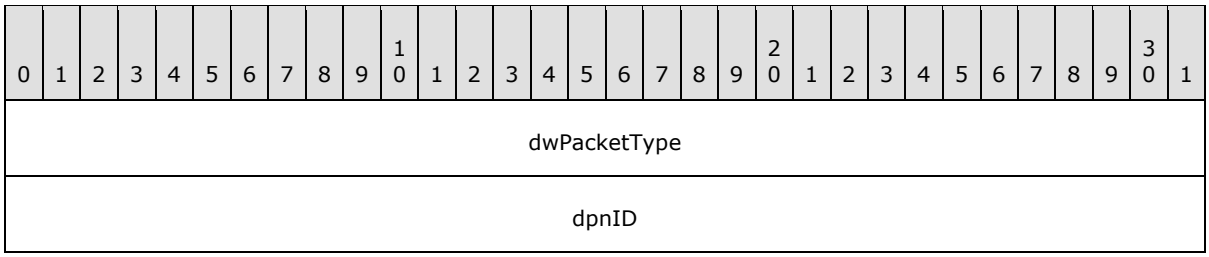
nType (2 bytes): Identifies the type of chat message being transmitted.

Value	Meaning
GAME_MSGID_CHAT 1	This is the only valid value for this field.

strChatString (400 bytes): Unicode-format chat message string. The application always sends 200 Unicode characters.

2.2.26 TRANS_USERDATA_SEND_PLAYER_DNID

The TRANS_USERDATA_SEND_PLAYER_DNID packet is used to send a user identification number to another client. This packet begins with a TRANS_USERDATA_HEADER.



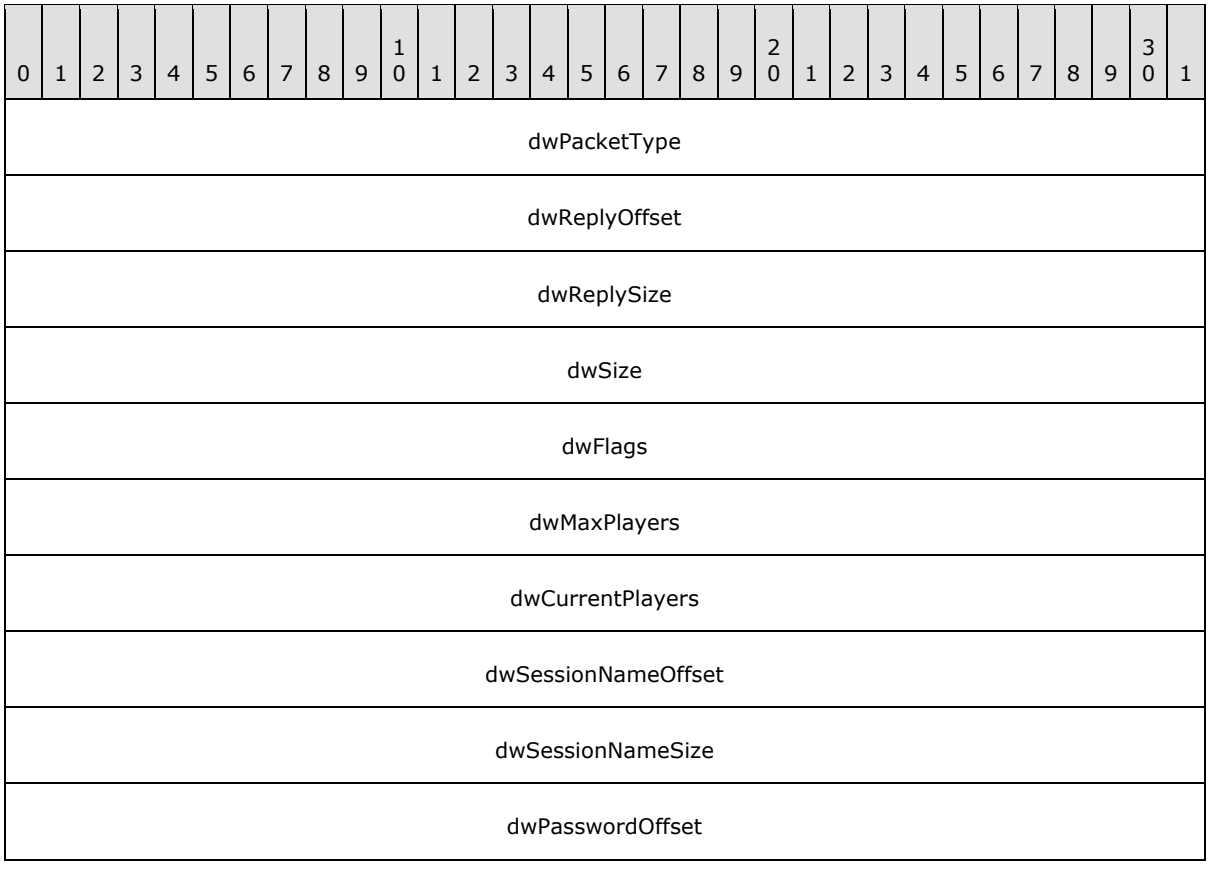
dwPacketType (4 bytes): Packet type field.

Value	Meaning
TRANS_USERDATA_SEND_PLAYER_DNID 0x000000C4	Sends user identification to another client.

dpnID (4 bytes): Identifier of the client.

2.2.27 TRANS_USERDATA_SEND_SESSION_INFO

The TRANS_USERDATA_SEND_SESSION_INFO packet is used by the session server to relay session information to the client. This packet begins with a TRANS_USERDATA_HEADER.



dwPasswordSize
dwReservedDataOffset
dwReservedDataSize
dwApplicationReservedDataOffset
dwApplicationReservedDataSize
guidInstance
...
...
...
applicationGUID
...
...
...
dpnid
dwVersion
dwVersionNotUsed
dwEntryCount
dwMembershipCount
DN_NAMETABLE_ENTRY_INFO (variable)
...
DN_NAMETABLE_MEMBERSHIP_INFO (variable)

...
URL (variable)
...
Data (variable)
...
name (variable)
...
reply (variable)
...
SessionName (variable)
...
ReservedData (variable)
...
ApplicationReservedData (variable)
...

dwPacketType (4 bytes): A 32-bit integer that indicates the packet type.

Value	Meaning
TRANS_USERDATA_SEND_SESSION_INFO 0x000000C2	The server response to a client that contains session information.

dwReplyOffset (4 bytes): The offset, in bytes, from the packet header of the **reply[]** field. If **dwReplyOffset** is zero, the packet does not include a reply.

dwReplySize (4 bytes): The size, in bytes, of the **reply[]** field.

dwSize (4 bytes): The size, in bytes, of the application description information.

dwFlags (4 bytes): A 32-bit integer that specifies the application flags.

dwMaxPlayers (4 bytes): A 32-bit integer that specifies the maximum number of clients allowed in the session.

dwCurrentPlayers (4 bytes): A 32-bit integer that specifies the current number of clients in the session.

dwSessionNameOffset (4 bytes): The offset, in bytes, from the packet header to the **sessionName[]** field. If **dwSessionNameOffset** is zero, the packet does not include a session name.

dwSessionNameSize (4 bytes): The size, in bytes, of the **sessionName[]** field.

dwPasswordOffset (4 bytes): The offset, in bytes, from the packet header to the start of the password. When the packet does not include a password, this field MUST be set to zero.

dwPasswordSize (4 bytes): The size, in bytes, of the password. When the packet does not include a password, this field MUST be set to zero.

dwReservedDataOffset (4 bytes): The offset, in bytes, from the packet header to the **reservedData[]** field. If **dwReservedDataOffset** is zero, the packet does not include reserved data.

dwReservedDataSize (4 bytes): The size, in bytes, of the **reservedData[]** field.

dwApplicationReservedDataOffset (4 bytes): The offset, in bytes, from the packet header to the **applicationReservedData[]** field. If **dwApplicationReservedDataOffset** is zero, the packet does not include application reserved data.

dwApplicationReservedDataSize (4 bytes): The size, in bytes, of the **applicationReservedData[]** field.

guidInstance (16 bytes): The instance globally unique identifier (GUID) of the connecting client.

applicationGUID (16 bytes): The application GUID.

Value	Meaning
61EF80DA-691B-4247-9ADD-1C7BED2BC13E	DXDiag application GUID.

dpnid (4 bytes): A 32-bit integer that provides the identifier for the new client joining the session.

dwVersion (4 bytes): A 32-bit integer that specifies the current name table version.

dwVersionNotUsed (4 bytes): Not used.

dwEntryCount (4 bytes): A 32-bit integer that provides the number of entries in the name table.

dwMembershipCount (4 bytes): A 32-bit integer that provides the number of memberships in the name table.

DN_NAMETABLE_ENTRY_INFO (variable): A dwEntryCount size array of structures that provide information on a **name table entry**, as specified in section 2.2.27.1.

DN_NAMETABLE_MEMBERSHIP_INFO (variable): A dwMembershipCount size array of structures that provide information on a name table membership, as specified in section 2.2.27.2.

URL (variable): Zero-terminated character array that contains the URL of a user in the chat session. This field's position is determined by **dwURLOffset** and the size stated in **dwURLSize**; both are fields in the corresponding DN_NAMETABLE_ENTRY_INFO structure. There can be multiple instances of the **URL** field, with an upper limit specified by the **dwURLSize** field.

Data (variable): Zero-terminated character array that contains the user data. This field's position is determined by **dwDataOffset** and the size stated in **dwDataSize**; both are fields in the corresponding DN_NAMETABLE_ENTRY_INFO structure. There can be multiple instances of the **Data** field with an upper limit specified by the **dwEntryCount** field.

name (variable): Zero-terminated character array that contains the client name. This field's position is determined by **dwNameOffset** and the size stated in **dwNameSize**; both are fields in the corresponding DN_NAMETABLE_ENTRY_INFO structure. There can be multiple instances of the **name** field with an upper limit specified by the **dwEntryCount** field.

reply (variable): Zero-terminated character array that contains the reply. This field's position is determined by **dwReplyOffset** and the size stated in **dwReplySize**.

SessionName (variable): Zero-terminated character array that contains the session name. This field's position is determined by **dwSessionNameOffset** and the size stated in **dwSessionNameSize**.

ReservedData (variable): Zero-terminated character array that contains the reserved data. This field's position is determined by **dwReservedDataOffset** and the size stated in **dwReservedDataSize**.

ApplicationReservedData (variable): Zero-terminated character array that contains the application reserved data. This field's position is determined by **dwApplicationReservedDataOffset** and the size stated in **dwApplicationReservedDataSize**.

2.2.27.1 DN_NAMETABLE_ENTRY_INFO

Information on a name table entry. The number of DN_NAMETABLE_ENTRY_INFO structures in this packet is specified in the **dwEntryCount** field.

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
dpnid																															
dpnidOwner																															
dwFlags																															
dwVersion																															
dwVersionNotUsed																															
dwDNETVersion																															
dwNameOffset																															
dwNameSize																															
dwDataOffset																															
dwDataSize																															
dwURLOffset																															
dwURLSize																															

dpnid (4 bytes): A 32-bit integer that specifies the DirectPlay identifier.

dpnidOwner (4 bytes): A 32-bit integer that provides the DirectPlay identifier for the owner.

dwFlags (4 bytes): A 32-bit integer that specifies the session flags.

dwVersion (4 bytes): A 32-bit integer that specifies the version number of the name table.

dwVersionNotUsed (4 bytes): Not used.

dwDNETVersion (4 bytes): A 32-bit integer that provides the DirectPlay version.

Value	Meaning
0x00000001	DirectX version 8.0
0x00000002	DirectX version 8.1
0x00000003	Pocket PC

Value	Meaning
0x00000004	Not used.
0x00000005	Windows Server 2003
0x00000006	DirectX version 8.2
0x00000007	DirectX version 9.0

dwNameOffset (4 bytes): The offset, in bytes, from the TRANS_USERDATA_SEND_SESSION_INFO packet header to the **name[]** field.

dwNameSize (4 bytes): The size, in bytes, of the **name[]** field.

dwDataOffset (4 bytes): The offset, in bytes, from the TRANS_USERDATA_SEND_SESSION_INFO packet header to the **data[]** field.

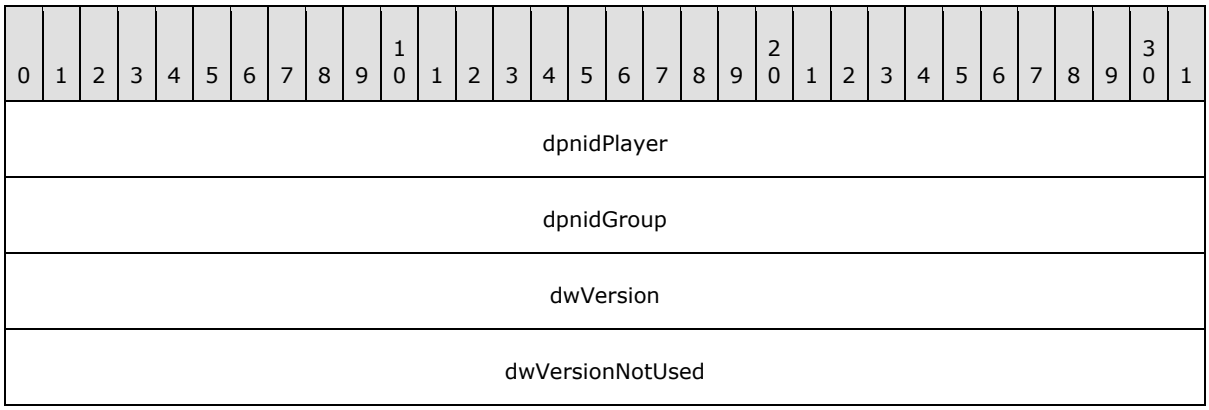
dwDataSize (4 bytes): The size, in bytes, of the **data[]** field.

dwURLOffset (4 bytes): The offset, in bytes, from the TRANS_USERDATA_SEND_SESSION_INFO packet header to the **url[]** field.

dwURLSize (4 bytes): The size, in bytes, of the **url[]** field.

2.2.27.2 DN_NAMETABLE_MEMBERSHIP_INFO

Information on a name table membership. The number of DN_NAMETABLE_MEMBERSHIP_INFO structures in this packet is specified in the **dwMembershipCount** field.



dpnidPlayer (4 bytes): A 32-bit integer that specifies the DirectPlay identifier for the user.

dpnidGroup (4 bytes): A 32-bit integer that provides the DirectPlay identifier for the group.

dwVersion (4 bytes): A 32-bit integer that specifies the name table version.

dwVersionNotUsed (4 bytes): Not used.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in the DirectPlay DXDiag Usage 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 what is described in this document.

Name Table: The list of systems participating in a DXDiag session used both for local use and for transmission to enable peer-to-peer connectivity when additional participants join. This could also be considered the player list. It has a version number that monotonically increases with every operation that changes the name table content such as adding or removing a player.

Name Table Entry: The DN_NAMETABLE_ENTRY_INFO structure along with associated strings and data buffers for an individual participant in the DXDiag session. These could be considered players.

3.1.2 Timers

3.1.2.1 Connect Retry Timer

Timer used to retry TRANS_COMMAND_CONNECT and TRANS_COMMAND_CONNECT_ACCEPT messages if no response is received. Implementations MAY retry as many times as needed at any frequency. Recommended values are for the first retry to be 200 ms, doubling every subsequent retry with a cap at 5 seconds and 14 retries.

3.1.2.2 Retry Timer

Timer tracking when a TRANS_USERDATA_HEADER prefixed message appears to have been dropped and needs to be retried or to have a send mask sent. Recommended values are for the first retry to be 2.5 RTT + the delayed acknowledgment time out (nominally 100 ms). It is also recommended that there be linear backoff for the second and third retries, exponential backoff for subsequent retries 4 through 8, and an overall cap at 5 seconds and 10 retries.

3.1.3 Initialization

There is no common initialization.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Sending a Chat Message

When a participant wants to send a chat message to others, it SHOULD send a TRANS_USERDATA_SEND_MESSAGE to the other participants. The TRANS_USERDATA_HEADER for the message SHOULD indicate that it is unreliable and sequential, that is, it SHOULD have the PACKET_COMMAND_SEQUENTIAL flag set, and SHOULD NOT have the PACKET_COMMAND_RELIABLE flag set in the bCommand field.

3.1.4.2 Disconnecting

When a participant wants to disconnect, the upper layer SHOULD initiate the sequence defined in section 3.1.5.3.

3.1.5 Message Processing Events and Sequencing Rules

The DXDiag application allows a client and server to create a chat session.

3.1.5.1 Client Joins a DirectPlay Session with No Other Clients

- The client sends a SESS_ENUM_QUERY session packet in search of a chat session.
- The server responds to the client with a SESS_ENUM_RESPONSE session packet. These DirectPlay session packets are identifiable by a leading 0-byte tag.
- The client requests a connection using a TRANS_COMMAND_CONNECT user data packet.
- The server responds with a TRANS_COMMAND_CONNECT_ACCEPT user data packet.
- The client issues a TRANS_COMMAND_CONNECT_ACCEPT user data packet to acknowledge the connection.
- The server and client exchange TRANS_USERDATA_KEEPALIVE request packets.
- The client sends user information in a TRANS_USERDATA_PLAYER_CONNECT_INFO packet.
- The client sends a TRANS_USERDATA_ACK_SESSION_INFO acknowledgment to the server.
- The server sends a TRANS_USERDATA_INSTRUCT_CONNECT to the client to instruct it to form a connection.
- The client responds with a TRANS_USERDATA_NAMETABLE_VERSION packet.
- The server sends a TRANS_USERDATA_RESYNC_VERSION packet.
- The client acknowledges it by sending a TRANS_COMMAND_SACK packet.

3.1.5.2 Client Joins a DirectPlay Session with Multiple Other Clients

- The client sends a SESS_ENUM_QUERY session packet in search of a chat session.
- The server responds to the client with a SESS_ENUM_RESPONSE session packet. These DirectPlay session packets are identifiable by a leading 0-byte tag.
- The client requests a connection using a TRANS_COMMAND_CONNECT user data packet.
- The server responds with a TRANS_COMMAND_CONNECT_ACCEPT user data packet.
- The client issues a TRANS_COMMAND_CONNECT_ACCEPT user data packet to acknowledge the connection.
- The server and client exchange TRANS_USERDATA_KEEPALIVE request packets.
- The client sends user information in a TRANS_USERDATA_PLAYER_CONNECT_INFO packet.

- The server, after receiving the TRANS_USERDATA_PLAYER_CONNECT_INFO packet, alerts existing clients to the existence of the new client by sending a TRANS_USERDATA_ADD_PLAYER packet.
- The new client tests the network path to the existing clients with SESS_PATH_TEST packets.
- The new client sends a TRANS_USERDATA_ACK_SESSION_INFO to acknowledge the session information previously received from the server.
- The existing clients send the server a TRANS_USERDATA_NAMETABLE_VERSION packet.
- The server commands all clients to resynchronize their name tables with a TRANS_USERDATA_RESYNC_VERSION packet.
- The server sends a TRANS_USERDATA_INSTRUCT_CONNECT packet to each existing client to instruct them to connect to the new client. Each client initiates the connection with the new client using a TRANS_COMMAND_CONNECT packet.
- The new client responds with a TRANS_COMMAND_CONNECT_ACCEPT packet.
- The existing client acknowledges the connection with a TRANS_COMMAND_CONNECT_ACCEPT packet.
- The new client and the existing client exchange TRANS_USERDATA_KEEPALIVE packets, and the existing client transmits its user identifier in a TRANS_USERDATA_SEND_PLAYER_DNID packet to the new client.
- Finally, all clients acknowledge the session server and each other using TRANS_COMMAND_SACK packets.

3.1.5.3 Client Disconnects from Chat Session

When a client disconnects, the server makes an announcement to remaining clients, and then the departing client disconnects.

- The client issues a TRANS_USERDATA_END_OF_STREAM packet.
- The server responds with four TRANS_COMMAND_SACK packets and a TRANS_USERDATA_END_OF_STREAM packet (in any order).
- The client returns four TRANS_COMMAND_SACK packets before disconnecting.
- If other clients are present in the session, the server sends each one a TRANS_USERDATA_DESTROY_PLAYER packet to remove the departing client from their name table. This can happen before the server receives the final TRANS_COMMAND_SACK packet from the departing client.
- Each remaining client sends the server a TRANS_USERDATA_REQ_INTEGRITY_CHECK and TRANS_COMMAND_SACK packet to acknowledge the removal of the client.

3.1.5.4 Server Disconnects from Chat Session

A server can leave without destroying the chat session. The DirectPlay DXDiag Usage Protocol allows hosting to migrate to another member currently in the session. By default, the DXDiag utility transfers the hosting to the client that has been in the chat the longest.

- The hosting migration begins with the server sending a TRANS_USERDATA_END_OF_STREAM packet to all clients in the session.
- The server responds with four TRANS_COMMAND_SACK packets and a TRANS_USERDATA_END_OF_STREAM packet (in any order).
- The server sends four TRANS_COMMAND_SACK packets to each client, and disconnects.
- The client that has been in the session the longest becomes the new server and contacts each client with a TRANS_USERDATA_HOST_MIGRATE user data packet.
- Each client sends a TRANS_USERDATA_NAMETABLE_VERSION user data packet to the new server.
- The new server ends the migration sequence with TRANS_USERDATA_HOST_MIGRATE_COMPLETE and TRANS_USERDATA_DESTROY_PLAYER packets. The TRANS_USERDATA_DESTROY_PLAYER packet identifies the previous host to remove from the client session list.

3.1.5.5 Participant Receives Chat Message

When a participant receives a chat message, it SHOULD display the chat message to the user. It SHOULD also send a TRANS_COMMAND_SACK message to acknowledge the packet sequence number in which the chat message was delivered.

3.1.5.6 Send Sequence ID (bNSeq) Validation and Processing

The TRANS_USERDATA_HEADER bNSeq field MUST either be the next sequence ID expected by the receiver or be within 63 packets beyond the expected ID. If the sequence ID is outside this range, a SACK packet SHOULD be sent, indicating the current expected state. If the PACKET_COMMAND_POLL flag is set in bCommand, this packet SHOULD be sent immediately; otherwise, the dedicated ACK timer SHOULD be set using the short time out (nominally 20 ms), and the SACK SHOULD be sent at that time. The payload of the packet SHOULD then be ignored.

3.1.5.7 Acknowledged Sequence ID (bNRcv) Processing

If the TRANS_USERDATA_HEADER bNSeq sequence ID is valid, the bNRcv field SHOULD be inspected. All previously sent TRANS_USERDATA_HEADER packets that are covered by the bNRcv sequence ID, that is, those packets that had been sent with bNSeq values less than bNRcv (accounting for 8-bit counter wrapping) are acknowledged. They no longer need to be remembered, and their retry timers can be canceled.

3.1.5.8 SACK Mask Processing

When one or both of the optional SACK mask 32-bit fields is present, and one or more bits are set, the sender is indicating that it received a packet or packets with sequence IDs higher than bNRcv out of order, presumably due to packet loss. The receiver of a SACK mask SHOULD shorten the retry timer for the first frame of the window to 10 ms to speed recovery from the packet loss. The receiver MAY also choose to remove the selectively acknowledged packets from its list to retry.

3.1.5.9 Send Mask Processing

When one or both of the optional send mask 32-bit fields are present, and one or more bits are set, the sender is indicating that it sent an unreliable packet or packets for which it did not receive acknowledgements yet. The receiver of a send mask SHOULD loop through each bit of the combined

64-bit value from the least significant bit to the most significant in little-endian byte order. Each bit corresponds to a sequence ID prior to bNSnd, and, if that is the bit that is set, it indicates that the corresponding packet had been sent unreliably and will not be retried. If the recipient of the send mask had not received the packet and had not already processed a send mask identifying the sequence ID, it SHOULD consider the packet as dropped and release its placeholder in the sequence. That is, any sequential messages that could not be indicated because of the gap in the sequence (where the unreliable packet had been) can now be processed.

3.1.6 Timer Events

3.1.6.1 Connect Retry Timer

When the connect retry timer expires, a new TRANS_COMMAND_CONNECT or TRANS_COMMAND_CONNECT_ACCEPT message SHOULD be sent, depending on the current state. The connect retry timer SHOULD then be rescheduled for the next period. It is recommended that the retry period start at 200 ms, doubling every time with a maximum of 5 seconds and 14 retries.

If the maximum number of retries has already been attempted when the timer expires, the connection attempt SHOULD be considered as failed. If the connection was initiated from an inbound TRANS_COMMAND_CONNECT packet arriving on a listening system, the listener MAY optionally choose to go back to listening if it did not allow additional connection attempts while the failed attempt was in progress.

3.1.6.2 Retry Timer

When the retry timer elapses without having been cancelled, and the associated packet was reliable, the TRANS_USERDATA_HEADER prefixed message SHOULD be resent, and the retry timer SHOULD then be scheduled for the next period. It is recommended that the retry period start at 2.5 RTT + 100 ms and that there be linear backoff for the second and third retries, exponential backoff for subsequent retries 4 through 8, and an overall cap at 5 seconds and 10 retries.

If the maximum number of retries has already been attempted when the timer expires, the connection SHOULD be considered as lost. All other in-progress sends SHOULD be discarded, and the upper layer SHOULD be informed of the disconnection.

When the retry timer elapses without having been cancelled, and the associated packet was unreliable, the packet's sequence ID SHOULD be remembered as requiring a send mask to be used in future transmissions.

3.1.7 Other Local Events

There are no other local events that affect DXDiag operation.

3.2 Server Details

3.2.1 Abstract Data Model

There is no recommended server-specific abstract data model.

3.2.2 Timers

There are no server-specific timers.

3.2.3 Initialization

When a DXDiag-compatible application initializes as a host, it begins listening for enumeration requests on port 6073 and a user-specified port. It also begins accepting connections on the user-specified port.

3.2.4 Higher-Layer Triggered Events

There are no server-specific higher-layer triggered events.

3.2.5 Message Processing Events and Sequencing Rules

When a server receives a SESS_ENUM_QUERY request, the server SHOULD respond with a SESS_ENUM_RESPONSE.

When a server receives a TRANS_COMMAND_CONNECT packet from a new client, the server SHOULD respond with a TRANS_COMMAND_CONNECT_ACCEPT packet and begin the sequence specified in section 3.1.5.1 or 3.1.5.2.

3.2.6 Timer Events

There are no server-specific timer events.

3.2.7 Other Local Events

There are no server-specific local events.

3.3 Client Details

3.3.1 Abstract Data Model

There is no recommended client-specific abstract data model.

3.3.2 Timers

There are no client-specific timers.

3.3.3 Initialization

When a DXDiag-compatible application initializes as a client, it begins enumerating for hosts on port 6073.

3.3.4 Higher-Layer Triggered Events

When a higher layer initiates session discovery, the client SHOULD begin sending SESS_ENUM_QUERY messages to the address specified by the higher layer. This MAY be the broadcast address to discover local area network DXDiag sessions.

A higher layer presents a list of discovered sessions to the user for selection, or it MAY automatically select a discovered session. The higher layer will then initiate a connection attempt to the specified session. The client SHOULD initiate the sequence identified in section 3.1.5.1 or 3.1.5.2.

3.3.5 Message Processing Events and Sequencing Rules

When a client receives a SESS_ENUM_RESPONSE to a previously sent query, the client SHOULD include the responder in a list of available sessions in the user interface.

When the server sends a TRANS_USERDATA_ADD_PLAYER message indicating that another client is joining the session, the client SHOULD install the new player's information in the name table. Once the client receives the subsequent TRANS_USERDATA_INSTRUCT_CONNECT message, the client SHOULD then begin connecting to the new participant using the previously specified addressing information.

3.3.6 Timer Events

There are no client-specific timer events.

3.3.7 Other Local Events

There are no client-specific local events.

4 Protocol Examples

The following sections describe operations as used in common scenarios to illustrate the function of the DirectPlay DXDiag Usage Protocol.

4.1 User Joins a DXDiag Chat Session Example

The following example describes how clients connect to a DXDiag chat session.

- In the DXDiag application, the user selects the DirectPlay test option on the Network tab.
- The user selects the Network Provider, and then clicks Join Session.
- The DXDiag application goes through the steps listed in section 3.1.5.1.

4.2 Client Disconnects from a DXDiag Chat Session Example

The following example describes how clients and servers disconnect from a DXDiag chat session.

- User selects CLOSE on the Chat window. This is the same action regardless of whether the user is the server or the client in the session. However, the sequence of events that results differs according to the current role of the participant, as specified in sections 3.1.5.3 and 3.1.5.4.

5 Security

The following sections specify security considerations for implementers of the DirectPlay DXDiag Usage Protocol.

5.1 Security Considerations for Implementers

The DirectPlay DXDiag Usage Protocol is not intended for applications that require robust security, which cannot be implemented using other layers such as IPsec.

5.2 Index of Security Parameters

There are no security parameters for the DirectPlay DXDiag Usage Protocol.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows XP
- Windows Server 2003
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification that is prescribed by 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 1.7: After the release of DirectX 6.0, DirectPlay versions (1) through 3A were modified to resolve to DirectPlay4. However, DirectPlay versions (1) through 3A were last released in Windows 98 Second Edition.

7 Index