[MC-DPLHP]:
DirectPlay 8 Protocol:
Host and Port Enumeration Specification

Revision Summary

Date	Revision History	Revision Class	Comments
08/10/2007	0.1	Major	Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.2.1	Editorial	Revised and edited the technical content.
11/30/2007	1.0	Major	Updated and revised the technical content.
01/25/2008	2.0	Major	Updated and revised the technical content.
03/14/2008	3.0	Major	Updated and revised the technical content.
05/16/2008	4.0	Major	Updated and revised the technical content.
06/20/2008	5.0	Major	Updated and revised the technical content.
07/25/2008	5.1	Minor	Updated the technical content.
08/29/2008	5.1.1	Editorial	Revised and edited the technical content.
10/24/2008	5.2	Minor	Updated the technical content.
12/05/2008	6.0	Major	Updated and revised the technical content.
01/16/2009	6.1	Minor	Updated the technical content.
02/27/2009	7.0	Major	Updated and revised the technical content.
04/10/2009	7.0.1	Editorial	Revised and edited the technical content.
05/22/2009	7.1	Minor	Updated the technical content.
07/02/2009	7.1.1	Editorial	Revised and edited the technical content.
08/14/2009	7.2	Minor	Updated the technical content.
09/25/2009	8.0	Major	Updated and revised the technical content.
11/06/2009	8.0.1	Editorial	Revised and edited the technical content.
12/18/2009	8.0.2	Editorial	Revised and edited the technical content.
01/29/2010	9.0	Major	Updated and revised the technical content.
03/12/2010	10.0	Major	Updated and revised the technical content.
04/23/2010	10.0.1	Editorial	Revised and edited the technical content.
06/04/2010	10.0.2	Editorial	Revised and edited the technical content.
07/16/2010	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
08/27/2010	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	10.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	10.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

This specification pertains to the DirectPlay 8 Protocol and describes the technology available for enumerating DirectPlay 8 hosts and ports. The enumeration functionality provided by the DirectPlay 8 Protocol allows a DirectPlay 8 Client/Peer to discover one or more DirectPlay 8 Servers/Hosts.

1.1 Glossary

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

client
client/server mode
DirectPlay
DirectPlay 8
DirectPlay 8 application
DirectPlay 8 protocol
DirectPlay 8 service provider
DirectPlay Client
DirectPlay Host
DirectPlay Peer
DirectPlay Server
game
globally unique identifier (GUID)
host
host Migration
Internetwork Packet Exchange (IPX)
peer
peer-to-peer mode
player
round-trip time (RTT)
server (3)
service provider
Unicode
User Datagram Protocol (UDP)

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

game session
group
payload
serial link

The following terms are specific to this document:

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

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

1.2.1 Normative References

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

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

[MC-DPL8CS] Microsoft Corporation, "DirectPlay 8 Protocol: Core and Service Providers Specification".

[MC-DPL8R] Microsoft Corporation, "DirectPlay 8 Protocol: Reliable Specification".

[MS-DPDX] Microsoft Corporation, "DirectPlay DXDiag Usage Protocol Specification".

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

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

[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

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

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

1.3 Overview

The DirectPlay 8 Protocol: Host and Port Enumeration enables a **DirectPlay Client**/**Peer** to discover **DirectPlay Servers**/**Hosts**.

The basic functionality is simple. A DirectPlay Client/Peer sends an EnumQuery message over a communications network. If the EnumQuery message is received by a DirectPlay Server/Host, and the DirectPlay Server/Host looks to enable the DirectPlay Client/Peer to connect to the **game session** that it is hosting, it replies to the DirectPlay Client/Peer with an EnumResponse message. The EnumResponse message contains the information required by the DirectPlay Client/Peer to attempt to connect to the game session being hosted by the DirectPlay Server/Host.

Note that it is possible for one EnumQuery message to be delivered to multiple DirectPlay Servers/Hosts, each of which may or may not reply with an EnumResponse message. Therefore, one EnumQuery message may generate zero, one, or more than one EnumResponse messages. The DirectPlay Client/Peer is not obligated to connect to any of the DirectPlay Servers/Hosts that reply with an EnumResponse message. On the contrary, one of the purposes of the DirectPlay 8 Protocol: Host and Port Enumeration process is to allow a DirectPlay Client/Peer to discover multiple game sessions and to choose which one to join based on application-specific preferences, such as **game** modes, latency, number of **players**, user preferences, and so on.

The EnumQuery and EnumResponse messages are delivered using the User Datagram Protocol (UDP) [RFC768] or a similar datagram-oriented, connectionless protocol. As a result, both EnumQuery and EnumResponse messages may be lost. It is therefore expected that a DirectPlay

Client/Peer will send multiple EnumQuery requests while searching for available DirectPlay Servers/Hosts.

It is possible, although not required, for the DirectPlay Client/Peer to note the **round-trip** latency of each EnumQuery/EnumResponse pair, and the number of EnumQuery/EnumResponse pairs that are lost, and use that information to predict the future quality of the network service between itself and the responding DirectPlay Servers/Hosts.

1.4 Relationship to Other Protocols

How a DirectPlay Client/Peer connects to the game session being hosted by a DirectPlay Server/Host that chooses to send an EnumResponse message is specified in [MC-DPL8CS].

The first byte of a valid EnumQuery or EnumResponse message must be set to 0x00. This causes the entire message to be processed as described in this specification [MC-DPLHP]. When the lead byte is nonzero, the entire message including the lead byte is passed through for processing as described in [MC-DPL8R].

A **DirectPlay 8 Service Provider** allows **DirectPlay 8** messages to be layered on top of multiple different underlying network transport protocols, such as IPv4, IPv6, **IPX**, and **serial links**. The details of the DirectPlay 8 Service Provider are specified in [MC-DPL8CS].

1.5 Prerequisites/Preconditions

The DirectPlay Client/Peer and DirectPlay Server/Host must have already agreed upon the application **GUID** they will use to identify themselves as instances of the same **DirectPlay 8 application**.

The DirectPlay Server/Host must be hosting a game session before it can participate in the DirectPlay 8 Protocol: Host and Port Enumeration.

1.6 Applicability Statement

The DirectPlay 8 Protocol: Host and Port Enumeration is appropriate for use when a DirectPlay Client/Peer has to query multiple DirectPlay Servers/Hosts for their current status, to determine (possibly with the assistance of user input) which DirectPlay Server/Host to connect to, if any.

On IPv4 networks, it is also appropriate to use the DirectPlay 8 Protocol: Host and Port Enumeration when only the IPv4 address information of a DirectPlay Server/Host is known, and the DirectPlay Client/Peer has to discover which port the DirectPlay Server/Host is using. In this case, the DirectPlay Client/Peer should send the EnumQuery (section 2.2.1) message to the DirectPlay 8**server** well-known port, as specified in [IANAPORT]. Note that not all DirectPlay Servers/Hosts will respond to EnumQuery messages sent to this port. Nor do all implementations of this protocol support the use of the DirectPlay 8server well-known port.

1.7 Versioning and Capability Negotiation

The DirectPlay 8 Protocol: Host and Port Enumeration has no versioning or capability negotiation features. However, the application may use the application-specific fields of the protocol to perform application-level versioning or capability negotiation.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

DirectPlay 8 uses the following well-known **UDP** port assignment, as specified in [IANAPORT].

Parameter	Value	Used by
directplay8	6073/udp	DirectPlay 8

In addition to port 6073, a DirectPlay 8 application may also use any other arbitrary port for "in-game" communication. However, DirectPlay 8 does not mandate that these other ports be numbered within any particular range or selected according to any particular scheme. In fact, the DirectPlay 8 Host and Port Enumeration Protocol primarily uses port 6073 to allow for discovery of these other ports.

The sender of a query message may use any port for the source of the message. The server listening on port 6073 will reply to the address and port from which it receives a query. While a DirectPlay 8 application may find it convenient to send a query from the port that is being used for "in-game" communication, the sender is not required to use this port or any other particular port.

Although many DirectPlay 8 applications explicitly specify the port numbers to use for "in-game" communication, when the application has not specified particular port number(s), the DirectPlay 8 implementation chooses the first available port in the range 2302-2400.

2 Messages

2.1 Transport

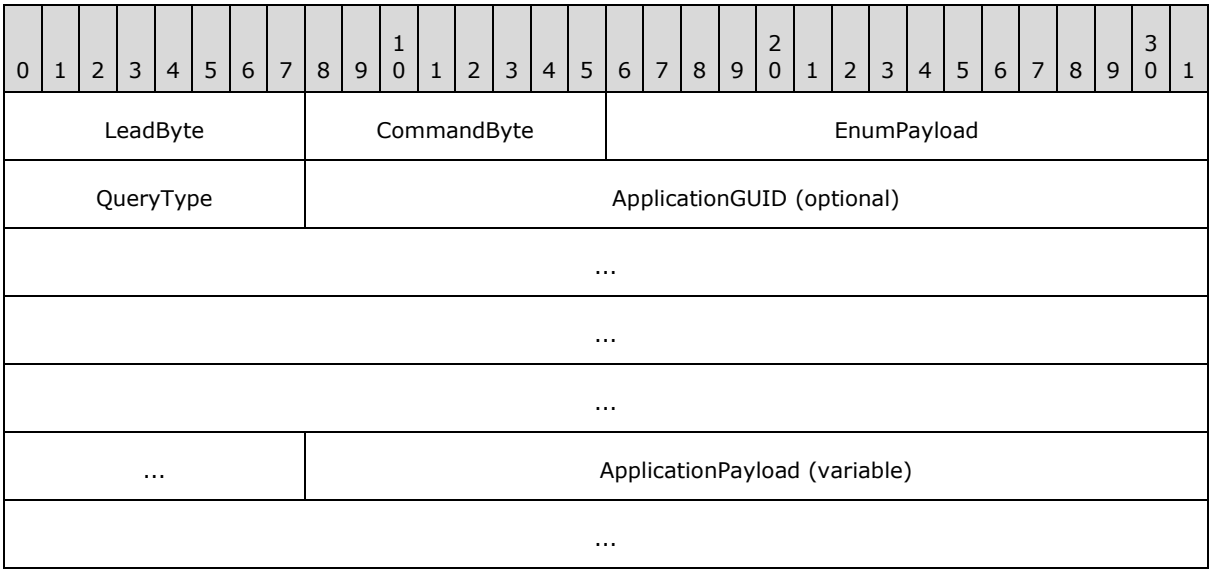
EnumQuery and EnumResponse messages are delivered using the same transport upon which the DirectPlay 8 Protocol: Reliable [MC-DPL8R] is built, which typically does not provide guaranteed message delivery. This means that both EnumQuery and EnumResponse messages might be lost.

2.2 Message Syntax

2.2.1 EnumQuery

The EnumQuery message is used to query for instances of a DirectPlay Server/Host that is hosting a game session.

The size of the variable length field in the EnumQuery message is limited by whatever limit is placed on the overall message size by the DirectPlay 8 Service Provider that is used to transmit the message.



LeadByte (1 byte): This field is 8 bits in length. It MUST be 0x00.

Note The first byte MUST be 0 for the message to be a valid EnumQuery message. When a message is received and the first byte is nonzero, the entire message MUST be passed through for processing as described in [MC-DPL8R].

CommandByte (1 byte): This field is 8 bits in length. It MUST be 0x02.

EnumPayload (2 bytes): This field is 16 bits in length. The **EnumPayload** is a value selected by the sender of the EnumQuery message that can be used to match EnumResponse messages to their corresponding EnumQuery.

QueryType (1 byte): This field is 8 bits in length.

Value	Meaning
0x02	Indicates that this EnumQuery message contains no **ApplicationGUID** field. All DirectPlay Servers/Hosts that receive this EnumQuery message in valid form SHOULD respond to it.
0x01	Indicates that this query contains an **ApplicationGUID** field. Only DirectPlay Servers/Hosts that are identified by the **ApplicationGUID** SHOULD respond to this EnumQuery message, if it is valid. For more information about the GUID type, see [MS-DTYP] section 2.3.2.

ApplicationGUID (16 bytes): The Application GUID. This optional field, when present, is 128 bits in length.

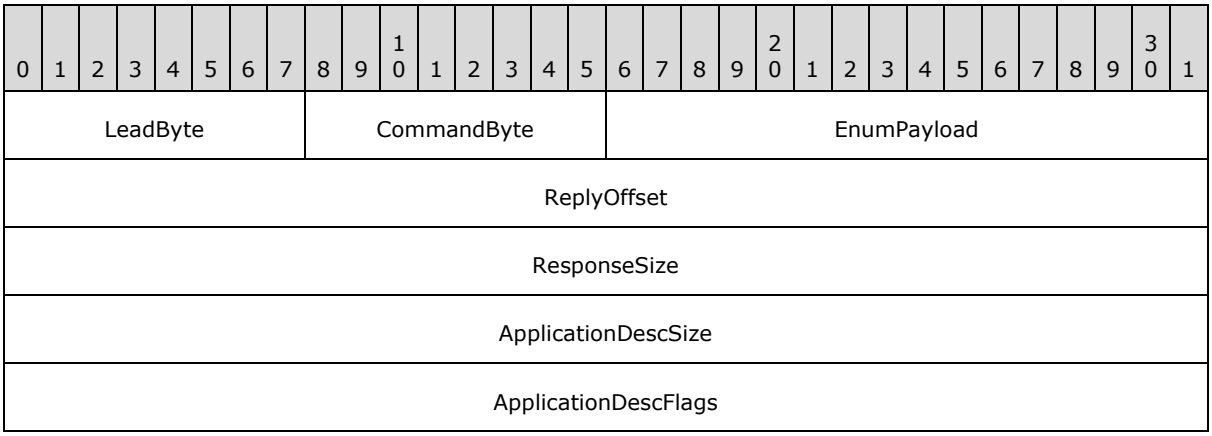
ApplicationPayload (variable): This variable-length optional field, when present, MUST be a multiple of 8 bits in length. Note that the receiver is expected to discover the size of the **ApplicationPayload** field by examining the total size of the message delivered by the underlying DirectPlay 8 Service Provider, because this is the only variable-length field in this message. No explicit-size field is provided. The contents of this field are application-specific and allow the DirectPlay Client/Peer to send additional application-specific information to the DirectPlay Server/Host. The **server application** can then use the information to decide if it will reply to the EnumQuery, and/or determine what additional information, if any, it SHOULD return in the EnumResponse message.

2.2.2 EnumResponse

When a valid EnumQuery message is received by a DirectPlay Server/Host, it SHOULD reply with an EnumResponse message. The DirectPlay Server/Host SHOULD NOT respond to any EnumQuery messages where the **QueryType** field is 0x01 and the **ApplicationGUID** field does not match the DirectPlay Server/Host GUID.

The EnumResponse message MUST be sent to the address from which the EnumQuery message was sent. The form of this address will depend on the DirectPlay 8 Service Provider that is being used. For example, in an IPv4 **service provider**, the address would consist of an IPv4 style address:port pair. The response MUST be sent from the address to which the DirectPlay Client/Peer connects if it chooses to join the game session.

The sizes of the variable-length fields in the EnumQuery message are limited by whatever limit is placed on the overall message size by the DirectPlay 8 Service Provider that is used to transmit the message.



MaxPlayers
CurrentPlayers
SessionNameOffset
SessionNameSize
PasswordOffset
PasswordSize
ReservedDataOffset
ReservedDataSize
ApplicationReservedDataOffset
ApplicationReservedDataSize
ApplicationInstanceGUID
...
...
...
ApplicationGUID
...
...
...
SessionName (variable)
...
Password (variable)
...
ReservedData (variable)

...
ApplicationReservedData (variable)
...
ApplicationData (variable)
...

LeadByte (1 byte): This field is 8 bits in length. It MUST be 0x00.

Note The first byte MUST be 0 for the message to be a valid EnumResponse message. When a message is received and the first byte is nonzero, the entire message MUST be passed through for processing as described in [MC-DPL8R].

CommandByte (1 byte): This field is 8 bits in length. It MUST be 0x03.

EnumPayload (2 bytes): This field is 16 bits in length. The **EnumPayload** is a value selected by the sender of the EnumQuery message that can be used to match EnumResponse messages to their corresponding EnumQuery.

Each EnumResponse message is generated because an EnumQuery message was received. The **EnumPayload** field in the EnumResponse message MUST match the **EnumPayload** field in the EnumQuery message that generated this EnumResponse message, so the DirectPlay Client/Peer can track which EnumQuery message this EnumResponse message is responding to.

ReplyOffset (4 bytes): This field is 32 bits in length. This field contains the zero-based offset, in bytes, of the **ApplicationData** field within this EnumResponse message. The zero-based offset of the **ApplicationData** field is measured from the start of the **ReplyOffset** field, that is, the offset into the EnumResponse message not counting the first 4 bytes. This field will be 0 if no **ApplicationData** field is contained in this message.

ResponseSize (4 bytes): This field is 32 bits in length. This field indicates the size, in bytes, of the **ApplicationData** field within the EnumResponse message. This field will be 0 if no **ApplicationData** field is contained in this message.

ApplicationDescSize (4 bytes): This field is 32 bits in length. Its value MUST be 0x00000050. It represents the sum of the size of this field plus the **ApplicationDescFlags**, **MaxPlayers**, **CurrentPlayers**, **SessionNameOffset**, **SessionNameSize**, **PasswordOffset**, **PasswordSize**, **ReservedDataOffset**, **ReservedDataSize**, **ApplicationReservedDataOffset**, **ApplicationReservedDataSize**, **ApplicationInstanceGUID**, and **ApplicationGUID** fields.

ApplicationDescFlags (4 bytes): This field is 32 bits in length and provides the characteristics of the application game session. It is a combination of the following bit flags.

Value	Meaning
DPNSESSION_CLIENT_SERVER 0x00000001	A **client/server** game session. If clear, a **peer-to-peer** game session.

Value	Meaning
DPNSESSION_MIGRATE_HOST 0x00000004	**Host migration** is allowed.
DPNSESSION_NODPNSVR 0x00000040	Not using **DirectPlay Name Server (DPNSVR)** (game session is not enumerable via well-known port 6073).
DPNSESSION_REQUIREPASSWORD 0x00000080	Password required to join game session.
DPNSESSION_NOENUMS 0x00000100	Enumerations are not allowed. This flag will never be set in an EnumResponse message.
DPNSESSION_FAST_SIGNED 0x00000200	Fast message signing is in use. For details about fast message signing, see [MC-DPL8R].
DPNSESSION_FULL_SIGNED 0x00000400	Full message signing is in use. For details about full message signing, see [MC-DPL8R].

Note Flags 0x00000200 and 0x00000400 will never both be set, because a game session MUST never use both fast message signing and full message signing at the same time.

MaxPlayers (4 bytes): This field is 32 bits in length. It contains the maximum number of players that can join the game session identified by this EnumResponse message.

CurrentPlayers (4 bytes): This field is 32 bits in length. It contains the number of players currently in the game session at the time the EnumResponse message was sent by the DirectPlay Server/Host. Note that by the time the EnumResponse is received by the DirectPlay Client/Peer, the number of players in the game session may have changed.

SessionNameOffset (4 bytes): A 32-bit field that specifies the offset, in bytes, from the start of the **ReplyOffset** field to the **SessionName** field. If **SessionNameOffset** is 0, the packet does not include a game session name.

SessionNameSize (4 bytes): A 32-bit field that contains the size, in bytes, of the **SessionName** field within this EnumResponse message. The size includes the termination character. If **SessionNameSize** is 0, the packet does not include a game session name.

PasswordOffset (4 bytes): This field is 32 bits in length. A password is never used in the EnumResponse message; therefore, the **PasswordOffset** field will always be 0.

PasswordSize (4 bytes): This field is 32 bits in length. A password is never used in the EnumResponse message; therefore, the **PasswordSize** field will always be 0.

ReservedDataOffset (4 bytes): A 32-bit field that specifies the offset, in bytes, from the end of the **EnumPayload** field to the **ReservedData** field. If **ReservedDataOffset** is 0, the packet does not include any reserved data. The **ReservedDataOffset** field was intended to be used for future extensions to the **DirectPlay 8 Protocol**, but was never used. This field is not used in the EnumResponse message and will be 0.

ReservedDataSize (4 bytes): A 32-bit field that specifies the size, in bytes, of the **ReservedData** field. If the value of the **ReservedDataOffset** field is 0, then **ReservedDataSize** MUST be 0. If **ReservedDataOffset** is not 0, then **ReservedDataSize** MUST NOT be 0. This field is not used in the EnumResponse message and will be 0.

ApplicationReservedDataOffset (4 bytes): This field is 32 bits in length. It contains the zero-based offset, in bytes, of the **ApplicationReservedData** field within this EnumResponse message. The zero-based offset of the **ApplicationReservedData** field is measured from the start of the **ReplyOffset** field, that is, the offset into the EnumResponse message not counting the first 4 bytes. If no **ApplicationReservedData** is contained in the EnumResponse message, this field will be 0.

ApplicationReservedDataSize (4 bytes): This field is 32 bits in length. It contains the size, in bytes, of the **ApplicationReservedData** field within this EnumResponse message. If no **ApplicationReservedData** is contained in the EnumResponse message, this field will be 0.

ApplicationInstanceGUID (16 bytes): This field is 128 bits in length. It contains the GUID that identifies the particular instance of the DirectPlay Server/Host that generated this EnumResponse message. Each instance of a DirectPlay Server/Host generates a new GUID each time it chooses to **host** a new game session. Since GUIDs are by definition unique, each game session will have a unique **ApplicationInstanceGUID**. For more information about the GUID type, see [MS-DTYP] section 2.3.2.

ApplicationGUID (16 bytes): This field is 128 bits in length. It contains the GUID that identifies the DirectPlay Server/Host type that generated this EnumResponse. Each game MUST generate its own GUID to identify itself, and all DirectPlay Servers/Hosts for that game share the same **ApplicationGUID** that identifies the type of the DirectPlay Server/Host.

SessionName (variable): An optional, variable-length field that contains the human-readable name of the game session in 16-bit **Unicode** characters. The position of the field within the packet is determined by the **SessionNameOffset** field and the size specified in the **SessionNameSize** field, in bytes. The field is zero-terminated.

Password (variable): The EnumResponse message will never contain a password; therefore, this field is unused.

ReservedData (variable): The **ReservedData** field was intended to be used for future extensions to the DirectPlay 8 Protocol, but was never used. This field will never be used since **DirectPlay** has been deprecated.

ApplicationReservedData (variable): This optional field is of variable length. Its zero-based offset within the EnumResponse message is specified in the **ApplicationReservedDataOffset** field. Its size, in bytes, is specified in the **ApplicationReservedDataSize** field. The contents of this field are determined by the game. This field is intended to represent game-specific data that changes infrequently. For example, data that is set when the game session is created, but does not change thereafter, is appropriate for use in this field.

ApplicationData (variable): This optional field is of variable length. Its zero-based offset within the EnumResponse message is specified in the **ReplyOffset** field. Its size, in bytes, is specified in the **ResponseSize** field. The contents of this field are determined by the game. This field is intended to represent game-specific data that changes frequently. For example, data that changes as the game session is used, such as the current state of the game, is appropriate for use in this field.

3 Protocol Details

3.1 Server 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 specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this specification.

A DirectPlay Server/Host MUST be hosting a game session to be eligible to reply to EnumQuery messages. A DirectPlay Server/Host that is hosting a game session SHOULD maintain the following DirectPlay 8 state information to be able to reply to an EnumQuery message:

- ApplicationDescFlags
- MaxPlayers
- CurrentPlayers
- SessionName (if any)
- Password (if any)

Additionally, a game can also maintain additional game specific state as follows:

- ApplicationReservedData
- ApplicationData

For detailed descriptions of each of these items, see the description of the EnumResponse message in section 2.2.2.

For more details on what it means to host a game session, see [MS-DPDX].

3.1.2 Timers

EnumQuery and EnumResponse messages are delivered by using the DirectPlay 8 service providers, which do not offer reliable message delivery. Therefore, to achieve a degree of reliability and to enable the collection of RTT and packet loss data, it is useful for the DirectPlay Client/Peer to send multiple EnumQuery messages that are spaced over a period of time.

It is appropriate to use a timer to manage the process of sending EnumQuery messages at regular intervals. The frequency of EnumQuery messages is implementation-defined. The DirectPlay 8 Protocol: Host and Port Enumeration places no restrictions on this frequency or on the number of EnumQuery messages that are sent.

3.1.3 Initialization

A DirectPlay Server/Host MUST be hosting a game session before it can respond to any EnumQuery messages.

Note that when using the IPv4 or IPv6 service provider, a DirectPlay Server/Host can specify which UDP port to use to listen for incoming messages. The DirectPlay Client/Peer MUST be aware of this

port selection in order to direct the EnumQuery message to the correct port. UDP port 6073 has been registered with IANA for use by DirectPlay 8. If the DirectPlay 8 application has no compelling reason to use a different port, this is a good port to choose on an IPv4 or IPv6 network. Because this port is registered with IANA (as specified in [IANAPORT]), and is used by multiple games, it increases the likelihood that some firewalls may be preconfigured to allow traffic on this port. However, a game can use any port it deems appropriate, according to the rules and customs of the IP networks that it is using.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

When a DirectPlay Server/Host is hosting a game session and it receives an EnumQuery message, it SHOULD respond to the address from which the EnumQuery message originated with an EnumResponse message. Note that the DirectPlay Server/Host can choose not to reply to any particular EnumQuery message for application-specific reasons, such as DirectPlay Server/Host load, current game state, or any other reason.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

3.2.1 Abstract Data Model

A DirectPlay Client/Peer can send EnumQuery messages at any time and to any destination. It is useful for the DirectPlay Client/Peer to keep a record of the EnumQuery messages it has sent in the recent past so that it can correlate any replies it receives with the original EnumQuery message. This enables the DirectPlay Client/Peer to measure the round-trip time (RTT) between itself and the responding DirectPlay Server/Host. It also enables the DirectPlay Client/Peer to notice any packet loss that may be occurring between itself and any responding DirectPlay Server/Host.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Processing Events and Sequencing Rules

None.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The following diagram shows an example use of the DirectPlay 8 Protocol: Host and Port Enumeration.

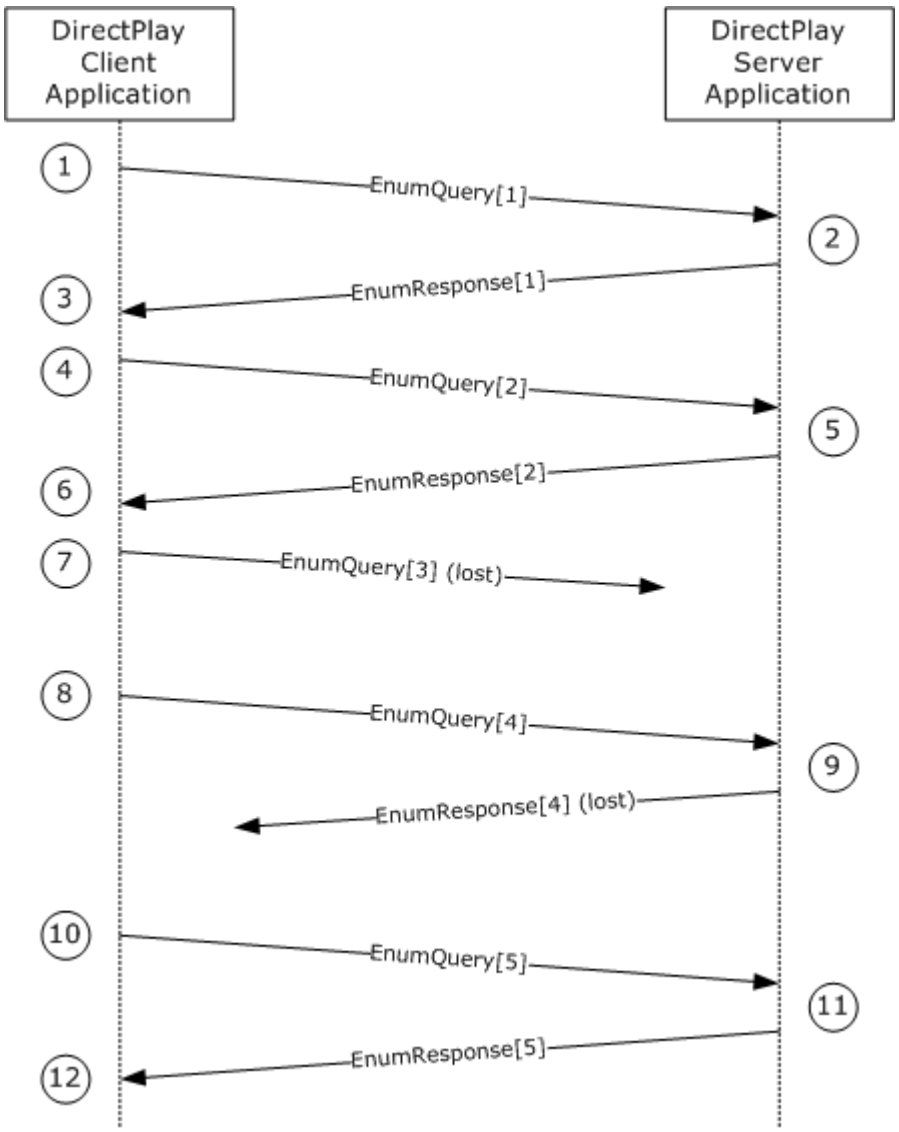


Figure 1: DirectPlay 8 Protocol: Host and Port Enumeration

The steps depicted in the diagram example are as follows:

1. The DirectPlay Client/Peer sends an EnumQuery message to the DirectPlay Server/Host. This EnumQuery message contains an **EnumPayload** value of 1. The EnumQuery message is sent to the DirectPlay Server/Host directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumQuery message is at risk of being lost in transit. In this example step, the EnumQuery message is successfully received by the DirectPlay Server/Host.

2. The DirectPlay Server/Host receives the EnumQuery message. The DirectPlay Server/Host is hosting a game session. Based on the content of the EnumQuery message and its own internal state, it responds to the EnumQuery message with an EnumResponse message. It copies the **EnumPayload** value of 1 from the EnumQuery message to the EnumResponse message and sends the EnumResponse message back to the address that the EnumQuery message came from. The EnumResponse message is sent directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumResponse message is at risk of being lost in transit. In this example step, the EnumResponse message is successfully received by the DirectPlay Client/Peer.

3. The DirectPlay Client/Peer receives the EnumResponse message. Based on the content of EnumResponse message, the DirectPlay Client/Peer has the information it requires to connect to the game session that is being hosted by the responding DirectPlay Server/Host. By measuring the elapsed time between sending the EnumQuery message with **EnumPayload** of 1, and receiving the EnumResponse message with **EnumPayload** of 1, the DirectPlay Client/Peer can also estimate the round-trip message latency between itself and the responding DirectPlay Server/Host. In this example, the DirectPlay Client/Peer does not immediately connect to the game session identified in the EnumResponse message. Instead, it continues sending EnumQuery messages at regular intervals to the DirectPlay Server/Host.

4. After some reasonable time period following step 1, the DirectPlay Client/Peer sends another EnumQuery message to the DirectPlay Server/Host. This EnumQuery message contains an **EnumPayload** value of 2. The EnumQuery message is sent to the DirectPlay Server/Host directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumQuery message is at risk of being lost in transit. In this example step, the EnumQuery message is successfully received by the DirectPlay Server/Host.

5. The DirectPlay Server/Host receives the EnumQuery message. The DirectPlay Server/Host is hosting a game session. Based on the content of the EnumQuery message and its own internal state, it responds to the EnumQuery message with an EnumResponse message. It copies the **EnumPayload** value of 2 from the EnumQuery message to the EnumResponse message and sends the EnumResponse message back to the address that the EnumQuery message came from. The EnumResponse message is sent directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore the EnumResponse message is at risk of being lost in transit. In this example step, the EnumResponse message is successfully received by the DirectPlay Client/Peer.

6. The DirectPlay Client/Peer receives the EnumResponse message. Based on the content of EnumResponse message, the DirectPlay Client/Peer has the information it requires to connect to the game session that is being hosted by the responding DirectPlay Server/Host. By measuring the elapsed time between sending the EnumQuery message with **EnumPayload** of 2, and receiving the EnumResponse message with **EnumPayload** of 2, the DirectPlay Client/Peer can also estimate the round-trip message latency between itself and the responding DirectPlay Server/Host. The DirectPlay Client/Peer now has two measurements of this round-trip message latency, and therefore can make a more accurate prediction of future message latency than it could after receiving only one EnumResponse message from this DirectPlay Server/Host. This is one of the benefits of sending multiple EnumQuery messages to the same DirectPlay Server/Host. In this example, the DirectPlay Client/Peer does not immediately connect to the game session identified in the EnumResponse message. Instead, it continues sending EnumQuery messages at regular intervals to the DirectPlay Server/Host.

7. After some reasonable time period following step 4, the DirectPlay Client/Peer sends another EnumQuery message to the DirectPlay Server/Host. This EnumQuery message contains an **EnumPayload** value of 3. The EnumQuery message is sent to the DirectPlay Server/Host directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery.

Therefore the EnumQuery message is at risk of being lost in transit. In this example step, the EnumQuery message is lost in transit and is not received by the DirectPlay Server/Host.

8. After some reasonable time period following step 7, the DirectPlay Client/Peer sends another EnumQuery message to the DirectPlay Server/Host. This EnumQuery message contains an **EnumPayload** value of 4. The EnumQuery message is sent to the DirectPlay Server/Host directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumQuery message is at risk of being lost in transit. In this example step, the EnumQuery message is successfully received by the DirectPlay Server/Host.

9. The DirectPlay Server/Host receives the EnumQuery message. The DirectPlay Server/Host is hosting a game session. Based on the content of the EnumQuery message and its own internal state, it responds to the EnumQuery message with an EnumResponse message. It copies the **EnumPayload** value of 4 from the EnumQuery message to the EnumResponse message and sends the EnumResponse message back to the address that the EnumQuery message came from. The EnumResponse message is sent directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumResponse message is at risk of being lost in transit. In this example step, the EnumResponse message is lost in transit and is not received by the DirectPlay Client/Peer.

10. After some reasonable time period following step 8, the DirectPlay Client/Peer sends another EnumQuery message to the DirectPlay Server/Host. This EnumQuery message contains an **EnumPayload** value of 5. The EnumQuery message is sent to the DirectPlay Server/Host directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumQuery message is at risk of being lost in transit. In this example step, the EnumQuery message is successfully received by the DirectPlay Server/Host.

11. The DirectPlay Server/Host receives the EnumQuery message. The DirectPlay Server/Host is hosting a game session. Based on the content of the EnumQuery message and its own internal state, it responds to the EnumQuery message with an EnumResponse message. It copies the **EnumPayload** value of 5 from the EnumQuery message to the EnumResponse message and sends the EnumResponse message back to the address that the EnumQuery message came from. The EnumResponse message is sent directly via the selected DirectPlay 8 Service Provider, which does not offer reliable message delivery. Therefore, the EnumResponse message is at risk of being lost in transit. In this example step, the EnumResponse message is successfully received by the DirectPlay Client/Peer.

12. The DirectPlay Client/Peer receives the EnumResponse message. Based on the content of EnumResponse message, the DirectPlay Client/Peer has the information it requires to connect to the game session that is being hosted by the responding DirectPlay Server/Host. By measuring the elapsed time between sending the EnumQuery message with **EnumPayload** of 5, and receiving the EnumResponse message with **EnumPayload** of 5, the DirectPlay Client/Peer can also estimate the round-trip message latency between itself and the responding DirectPlay Server/Host. The DirectPlay Client/Peer now has three measurements of this round-trip message latency, and therefore can make a more accurate prediction of future message latency than it could after receiving only two EnumResponse messages from this DirectPlay Server/Host.

This is one of the benefits of sending multiple EnumQuery messages to the same DirectPlay Server/Host. Depending on the time that has elapsed since sending the EnumQuery messages with **EnumPayload** 3 and **EnumPayload** 4, the DirectPlay Client/Peer may also reasonably conclude that these EnumQuery messages, or the EnumResponse messages they may have generated, have been lost in transit. With that information, the DirectPlay Client/Peer can also generate an estimate of the possible future message delivery reliability. At this time, the DirectPlay Client/Peer now has a reasonable estimate of the future round-trip message latency and reliability. It can also decide to not connect to the game session identified in the EnumResponse messages, attempt to connect to the game session identified in the EnumResponse messages, or continue to send additional periodic

EnumQuery messages to obtain more information regarding the message latency and reliability between itself and the DirectPlay Server/Host.

The DirectPlay Client/Peer may also have been sending EnumQuery messages to other DirectPlay Server/Host in parallel, and may find one of those other game sessions is better in some application-specific way. The point at which a DirectPlay Client/Peer stops sending EnumQuery messages to a particular DirectPlay Server/Host is application-specific. The method that a DirectPlay Client/Peer uses to determine which game session to attempt to join is application-specific.

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® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 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.

7 Change Tracking

This section identifies changes that were made to the [MC-DPLHP] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

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

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

8 Index

V