DirectPlay

The Microsoft® DirectPlay® application programming interface (API) is the component of Microsoft DirectX® that enables you to write network applications such as multiplayer games.

Like other components of DirectX, DirectPlay can be used with C, C++, and Microsoft Visual Basic®.

For an overview of the organization of the DirectPlay Help, see Roadmap.

Roadmap

Information on Microsoft® DirectPlay® is presented in the following sections.

What's New in DirectPlay. New features and functionality of this component in Microsoft DirectX® 8. If you've used DirectPlay before, read this section first because this technology has been completely redesigned since DirectX 7.

Introduction to DirectPlay. An overview of what DirectPlay is and what it can do for your application, together with a first look at some key objects and the steps involved in creating a network application.

Understanding DirectPlay. A deeper look at the underlying mechanisms. This section won't teach you how to implement a DirectPlay session, but it will help you understand the application programming interface (API) when you get into the details.

Using DirectPlay. A guide to using the API. You'll probably want to familiarize yourself with the table of contents for this section and then refer to parts of it as you need specific information. Use it in conjunction with the reference section.

[C++]
DirectPlay C++ Samples. A guide to the C/C++ sample applications in the SDK.

[Visual Basic]
DirectPlay Visual Basic Samples. A guide to the Visual Basic sample applications in the SDK.

[C++]
DirectPlay C/C++ Reference. Detailed information for the DirectPlay C++ API.

[Visual Basic]
DirectPlay Visual Basic Reference. Detailed information the DirectPlay Visual Basic API.

What's New in DirectPlay

The networking component of Microsoft® DirectX® has undergone a major revision. Microsoft DirectPlay® introduces a new set of interfaces that enable games to have more direct access to the hardware, providing better performance.

The following list describes some of the new DirectPlay features.

[C++]

Interfaces have been completely rewritten.

The complexity of creating a networked application has been dramatically simplified by separating the interfaces for creating peer-to-peer and client/server sessions. The interfaces for creating DirectPlay transport sessions, which are defined in the Dplay8.h header file, are now:

IDirectPlay8Peer
Provides methods for creating peer-to-peer sessions.

IDirectPlay8Client
Provides methods for creating the client-side portion of a client/server application.

IDirectPlay8Server
Provides methods for creating the server-side portion of a client/server application.

Lobbying is now independent from the rest of DirectPlay.

DirectPlay has removed the requirement that a lobby client work only with a DirectPlay application. This will allow for either the lobby service provider or the application to implement DirectPlay without concern for the other. Lobby implementation is now separated into two simplified interfaces that are defined in the Dplobby8.h header file.

IDirectPlay8LobbyClient
This interface is used to manage a lobby client and for enumerating and launching lobby-aware applications.

IDirectPlay8LobbiedApplication
This interface is used to register a lobby launchable application with the system so it can be lobby launched. It also is used to get the connection information from the lobby to enable game launching without querying the user.

Voice transmission has been added.

DirectPlay Voice provides a set of interfaces to add real-time voice communication to an application. The following interfaces are defined in the Dvoice.h header file.

IDirectPlayVoiceClient
Provides methods to create and manage clients in a DirectPlay Voice session.

IDirectPlayVoiceServer
Provides methods to host and manage a DirectPlay Voice session.

IDirectPlayVoiceTest
Used to test DirectPlay Voice audio configurations.

Addressing information has moved from GUID-based data to URL-based data format.

Previous versions of DirectPlay used binary chunks of data with GUID addresses that were difficult to implement and that humans could not read. In DirectX 8.0, DirectPlay introduces the representation of addresses in URL format. A set of interfaces, defined in Dpaddr.h, is used to create and manipulate the new addressing format.

IDirectPlay8Address
Provides generic addressing methods used to create and manipulate DirectPlay addresses.

IDirectPlay8AddressIP
Provides IP provider-specific addressing services.

Higher scalability and better memory management have been added.

Increases in consumer bandwidth have dramatically affected network game design and implementation. Improved DirectPlay thread-pool management makes it easier to for the developer to design scalable, more robust applications that can support massive multiplayer online applications.

[Visual Basic]

Objects have been completely rewritten.

The complexity of creating a networked application has been dramatically simplified by separating the objects used for creating peer-to-peer and client/server sessions. The objects for creating DirectPlay transport sessions are now

DirectPlay8Peer
Provides methods for creating peer-to-peer sessions.

DirectPlay8Client
Provides methods for creating the client-side portion of a client/server application.

DirectPlay8Server
Provides methods for creating the server-side portion of a client/server application.

Lobbying is now independent from the rest of DirectPlay.

DirectPlay has removed the requirement that a lobby client work only with a DirectPlay application. This will allow for either the lobby service provider or the application to implement DirectPlay without concern for the other. Lobby implementation is now separated into two simplified objects.

DirectPlay8LobbyClient
This object is used to manage a lobby client and for enumerating and launching lobby-aware applications.

DirectPlay8LobbiedApplication
This object is used to register a lobby launchable application with the system so it can be lobby launched. It also is used to get the connection information from the lobby to enable game launching without querying the user.

Voice transmission has been added.

DirectPlay Voice provides a set of objects to add real-time voice communication to an application. The following objects are defined in the Dvoice.h header file.

DirectPlayVoiceClient
Provides methods to create and manage clients in a DirectPlay Voice session.

DirectPlayVoiceServer
Provides methods to host and manage a DirectPlay Voice session.

DirectPlayVoiceTest
Used to test DirectPlay Voice audio configurations.

Addressing information has moved from GUID-based data to URL-based data format.

Previous versions of DirectPlay used binary chunks of data with GUID addresses that were difficult to implement and that humans could not read. In DirectX 8.0, DirectPlay introduces the representation of addresses in URL format. DirectPlay provides an object that is used to create and manipulate the new addressing format.

DirectPlay8Address
Provides generic addressing methods used to create and manipulate DirectPlay addresses.

Higher scalability and better memory management have been added.

Increases in consumer bandwidth have dramatically affected network game design and implementation. Improved DirectPlay thread-pool management makes it easier to for the developer to design scalable, more robust applications that can support massive multiplayer online applications.

Better support for Firewalls and Network Address Translators has been added.

Writing network games that traverse Network Address Translators (NATs), Firewalls, and other Internet Connection Sharing (ICS) methods can be difficult, particularly for non-guaranteed (UDP) traffic. Because DirectPlay 8.0 has been developed with these issues in mind, it will support NAT solutions where possible. The DirectPlay 8 TCP/IP service provider uses a single, developer-selectable UDP port for game data, making it possible to configure firewalls and NATs appropriately. Additionally, DirectPlay makes use of UDP so that, for

client/server games, clients behind some NATs will be able to connect to games without additional configuration.

Introduction To DirectPlay

The Microsoft® DirectPlay® API provides developers with the tools to develop multiplayer applications such as games or chat clients. For simplicity, this documentation will refer to all such applications as "games". A multiplayer application has two basic characteristics:

- Two or more individual users, each with a game client on their computer.
- Network links that enable the users' computers to communicate with each other, perhaps through a centralized server.

DirectPlay provides a layer that largely isolates your application from the underlying network. For most purposes, your application can simply use the DirectPlay API, and enable DirectPlay to handle the details of network communication. DirectPlay provides many features that simplify the process of implementing many aspects of a multiplayer application, including:

- Creating and managing both peer-to-peer and client/server sessions
- Managing users and groups within a session
- Managing messaging between the members of a session over different network links and varying network conditions
- Enabling applications to interact with lobbies
- Enabling users to communicate with each other by voice

This documentation provides a high-level overview of the capabilities of DirectPlay. Subsequent sections will take you into the details of how to use DirectPlay in your multiplayer game.

- Creating and Managing Sessions
- DirectPlay Network Communication
- Communicating with DirectPlay Objects
- DirectPlay Lobby Support
- DirectPlay Voice Communication

Creating and Managing Sessions

A game *session* is an instance of a particular multiplayer game. A session has two or more users playing simultaneously, each with the same game client on his or her computer. A *player* is an entity in the game itself, and is defined by the particular game. Each user may have more than one *player* in a game. However, the game application must manage these players itself, using separate Microsoft® DirectPlay® interfaces or objects for each player.

The first step in creating a session is to collect a group of users. There are two basic approaches:

- Many game sessions are arranged by a *lobby* application running on a remote computer. This approach is used by most Internet-based games.
- It is also possible to arrange games by having the individual users' computers communicate with each other. This approach is typically limited such situations as a group of potential users that are all on the same LAN.

Once the session has been arranged, the game is launched and gameplay begins. As the session proceeds, players may be eliminated from the session, or new players added. The details are up to the individual game.

With a multiplayer game, each user's UI can be synchronized with that of all the other users in the session. Managing a multiplayer session thus requires a continual stream of messages to and from each user. For example, every time a player moves, a message must be sent to update that player's position on all the other game clients in the session. The core of DirectPlay is that part of the API that supports efficient and flexible messaging between all the computers in a session.

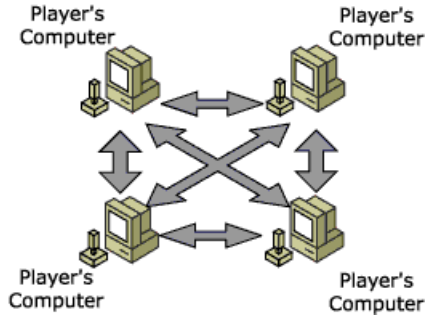
There are two basic ways to structure the messaging *topology* of a session: *peer-to-peer* and *client/server*. Both topologies have their advantages and limitations, so you will need to evaluate which is most appropriate for your game.

This section discusses

- Peer-to-Peer Topology
- Client/Server Topology

Peer-to-Peer Topology

A peer-to-peer game consists of the individual players' computers, connected by network links. Schematically, the topology of a four-player peer-to-peer game looks like:



Gameplay is handled by having each user's game client communicate directly with the other users' clients. For instance, when one user moves, the game client must send three update messages, one to each of the other users' computers.

A peer-to-peer game is normally arranged and launched through a *lobby client* application that resides on the user's computer. There are two basic ways the lobby client can arrange a session:

- The lobby client communicates directly with other potential users' lobby clients. This approach can be used, for instance, to arrange a game among users on the same LAN subnet.
- The lobby client acts as a link to lobby server application running on a remote computer. This is the way Internet-based games are normally arranged.

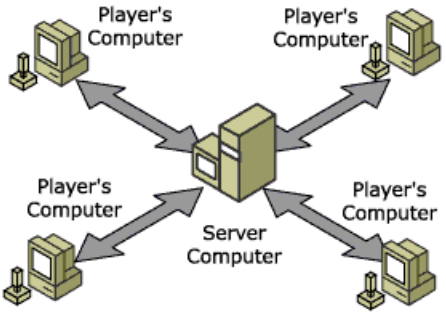
Once a session has been arranged and launched, most or all of the messaging will be user to user. If a lobby server is involved, it will only be handling such tasks as updating its list of session members when a player leaves the game, or enabling a new user to request entry to the session. Otherwise, the server stays in the background, and is typically not even aware of most of the messages that are being sent.

Because the server is either non-existent or at least not directly involved with the game play, one user is designated as the game *host*. They are responsible for handling logistical details such as bringing new players into an ongoing session.

Peer-to-peer games have the advantage of simplicity. All that is needed is a collection of players with game clients, and a way to organize a session. The primary drawback of the peer-to-peer topology is scalability. As the number of users increase, the number of messages needed to facilitate game play increases geometrically. The maximum number of users that can be accommodated depends on the game and the network bandwidth, but is typically no more than 20-30.

Client/Server Topology

A client/server game consists of the individual players' computers, connected to a central server computer. Schematically, the topology of a four-player peer-to-peer game looks like:



Gameplay is handled by having each user's game client communicate with the server. The server is responsible for passing information on to the other users. For instance, when one user moves, they send a message to the server. The server then sends messages to the other players to inform them of a change in game state. The server can have a number of responsibilities:

- Act as the session's messaging hub. Each computer only needs to send messages to the server. The server handles the logistics of synchronizing all the other users. This arrangement can substantially reduce message traffic, especially for large games.
- Host the game. The server normally takes care of the tasks that must be handled by the session host in a peer-to-peer game.
- Support many aspects of the game. The server often does much more than support game logistics. With many games, especially large ones, much of the processing that maintains the "game universe" takes place on the server. The game clients are primarily responsible for handling the user's UI.

A client/server game is normally arranged and launched through a *lobby client* application that resides on the user's computer. The lobby client acts as a link to a lobby server application that is normally running on a the same remote computer that is hosting the game. Once the game has been launched, the game server application becomes the host, and handles tasks such as admitting new users to the game.

There are a number of advantages to client/server games:

- They are more efficient, especially for large-scale games. In particular, the scale much better than peer-to-peer games, because additional players only cause a linear increase in the messaging traffic. The client/server topology is necessary for massively-multiplayer games.
- You not limited by the processing power of your users' computers. You can locate much of the processing required to maintain a large complex "game universe" on a single powerful computer, and let the users' computers handle the UI.
- You can control key aspects of your game at a central site. For instance, you can often update the game or fix bugs by simply modifying the server application, avoiding the need to update large numbers of game clients.

However, once you have developed and shipped a peer-to-peer game, you are essentially finished. The game clients are largely self-sufficient. With a client/server game, you have an ongoing commitment to your users that goes beyond providing normal support services. You must also provide and maintain a game server computer and the associated software, along with the network links to handle all the messaging, for the lifetime of the application. In the case of massively multiplayer games, you may need to operate your servers for extended periods with few or no breaks in service, or risk angering users by disrupting their gameplay.

DirectPlay Network Communication

The primary function of Microsoft® DirectPlay® is to provide you with efficient and flexible messaging support that largely isolates your application from the underlying network hardware and software. If you need to send a status update, you can simply call the relevant DirectPlay API, regardless of what kind of network link is involved.

DirectPlay network service providers support communication over TCP/IP, IPX, modem, and serial links.

This section discusses.

- DirectPlay Transport Protocol
- DirectPlay Addresses

Notes

DirectPlay does not support secure communications.

To use modems on Microsoft Windows® 95 systems, you must install version 2.0 of the Telephony API (TAPI). You can download TAPI 2.0 from http://www.microsoft.com.

DirectPlay Transport Protocol

The core of the Microsoft® DirectPlay® networking capabilities is the DirectPlay protocol. This transport-layer protocol has been completely overhauled for DirectPlay 8, and is now used for all messaging. The DirectPlay protocol is focused on making it simple for you to send data from the sending application to the target application, without needing to worry about what happens in between. The protocol offers a number of features that are tailored to the needs of multiplayer games, including:

- Reliable and unreliable delivery of messages. Reliable messages will be resent until the target application receives them. You can assign the delivery type on a message-by-message basis.
- Sequential and non-sequential delivery of messages. Sequential messages will be passed to the target application in the order they were sent.
- Message fragmentation and reassembly. If message size exceeds the capacity of a particular network, DirectPlay automatically fragments and reassembles the message.
- Congestion control. DirectPlay automatically *throttles* your outgoing messages to a level that can be handled by the target. This feature prevents you from flooding the target with more messages than it can process.
- Send prioritization. To ensure that the most important messages get sent first, DirectPlay enables you to designate messages as low, medium, or high priority. The high priority messages are sent to the front of the output queue, followed by medium and low priority messages.
- Message timeouts. To prevent the outgoing message queue from being clogged with messages that have been superseded by more recent messages, DirectPlay enables you to assign a timeout value to all messages. When a message times out, it is removed from the outgoing message queue, regardless of whether it has been sent or not.

DirectPlay Addresses

In order to deliver messages, each participant in a multiplayer game must have a unique address. Addresses can refer either to the computer that your application is running on (*device address*), or a computer that your application needs to communicate with (*host address*).

Microsoft® DirectPlay® addresses are in the form of URL strings. These strings consist of a scheme, scheme separator, and data string in the following general format.

x-directplay:/[*data string*]

The data string contains several elements that specify everything that is needed to enable communication to take place between sender and target, over a variety of different types of network link.

In use, the URL strings are embedded in a DirectPlay address object which is passed to or from DirectPlay API methods. You have the option of either manipulating the URL string directly, or using the methods exposed by the address object to handle each element of the data string separately.

Communicating with DirectPlay Objects

Microsoft® DirectPlay® essentially consists of a collection of COM objects. Each object exposes one or more interfaces that enable you to control various aspects of DirectPlay. For instance, the DirectPlay peer object (CLSID_DirectPlay8Peer) is used to manage peer-to-peer games.

[C++]
You communicate with a DirectPlay object by calling the methods exposed by its interfaces. For instance, to send some data to another user in a peer-to-peer game, you would send a message by calling the **IDirectPlay8Peer::SendTo** method. DirectPlay then takes care of getting the message to its target.

[Visual Basic]
You communicate with a DirectPlay object by calling the methods exposed by its interfaces. For instance, to send some data to another user in a peer-to-peer game, you would send a message by calling the **DirectPlay8Peer.SendTo** method. DirectPlay then takes care of getting the message to its target.

[C++]
DirectPlay communicates with your application through one or more *callback functions*. These functions are similar in principle to the familiar Window procedure. Your application implements the callback function and passes a pointer to the

function to DirectPlay during initialization. When DirectPlay needs to communicate with your application, it calls the callback function and passes in two key items of information:

- A message ID that identifies the message type
- A pointer to a block of data, typically a structure, that provides any needed details.

For instance, when the message sent in the above example arrives at its target, the target application's callback function will receive a message with a DPNMSGID_RECEIVE message id, indicating that a message has arrived from another user. The accompanying structure contains the data.

Because much of DirectPlay messaging is multithreaded, it is critical that callback functions be properly implemented.

[Visual Basic]
DirectPlay communicates with your application through one or more *message handlers*. A message handler is an object that DirectPlay calls to notify your application of various events. The documentation describes the methods that are exposed by the object, but you must implement all of the objects methods in your application. You then register the object during startup, and DirectPlay will call the object's methods to notify you when an event has occurred. Additional information about the event is passed through the method's parameters.

DirectPlay Lobby Support

A lobby is an application whose primary purpose is to enable players to meet and arrange games. It is typically located on a remote computer, and accessed over the Internet. Lobby servers often also perform a variety of other functions, such as hosting chat rooms, posting news and information, and selling merchandise. While lobby servers are convenient and commonly used to arrange multiplayer games, they aren't required. Multiplayer games can also be arranged by direct communication between lobby clients.

There are normally three components that are needed to enable a game to interact with a lobby:

- A *lobby server*
- A *lobby client*
- A *lobbyable* game.

Microsoft® DirectPlay® does not specify how you should implement a lobby server application. Instead, DirectPlay provides support for a lobby client. A lobby client is an application that is implemented by a lobby server vendor, and installed on each user's system. It serves as a link between the user and the lobby. While you could

handle such communication directly, you would have to know the specific implementation details of every lobby that might launch your game.

The lobby client application handles the details of communicating with its associated lobby server, using whatever protocols are appropriate. The lobby client communicates with the user and their game applications through a DirectPlay interface. DirectPlay then passes messages to the application. The application can also use a DirectPlay interface to pass messages to the lobby client.

A lobby can launch virtually any application. However, the application must have some specific lobby-aware components to take full advantage of lobby-launching. In particular, a lobbyable application can communicate with the lobby client throughout the course of the session. If an application is registered as lobbyable, the lobby client also automatically receives updates for various changes in game status, such as host migration.

DirectPlay Voice Communication

The current trend toward team-based multiplayer games makes player-to-player communication an essential part of gameplay. Historically this has been confined to text-based communication, where players type out the messages to their teammates. Although suitable for slower, turn-based games, text-based communication is at best an inconvenience for real-time games. Not only does it put slow typists at a disadvantage during gameplay but also it is a significant break in the reality that games attempt to create for the player. An obvious solution to the problem is the use of speech as a means for communication. It requires no training and increases the immersion of the game itself.

The windows platform provides all the tools required to provide real-time voice conferencing to video game developers, but it requires a significant amount of effort on the part of the game developer. This, combined with the cost and difficulty of obtaining the rights to compression technology capable of handling extremely low bandwidth situations, has prevented the wide-spread use of voice in games.

Microsoft® DirectPlay® 8.0 provides the game developer with a robust real-time voice conferencing system that requires a minimal amount of effort to use.

Understanding DirectPlay

This section of the Microsoft® DirectPlay® documentation provides the basic background you need to understand how to use the DirectPlay API in your application.

- Peer-to-Peer Sessions

- Client/Server Sessions
- DirectPlay Lobbies
- Basic Networking
- DirectPlay Callback Functions and Multithreading Issues
- Understanding DirectPlay Voice

Peer-to-Peer Sessions

A peer-to-peer session consists of a collection of users connected by a network. While a lobby server may be used to arrange and launch the game, the messaging needed to run the game is sent directly from one user's to another. Any communication with the lobby server is for such limited purposes as updating the list of participants.

With a peer-to-peer game, everything that is needed to run the game is part of the client software. With no server involved, all the processing needed to create and maintain the game universe must be handled by the client applications. This document discusses the basic principles of a lobbyable Microsoft® DirectPlay® peer-to-peer game. For a simple working example of a peer-to-peer application, see the SimplePeer application included with the SDK.

- Initiating a Peer-to-Peer Session
- Selecting a Service Provider for a Peer-to-Peer Session
- Selecting a Host for a Peer-to-Peer Session
- Connecting to a Peer-to-Peer Session
- Managing a Peer-to-Peer Session
- Host Migration
- Normal Peer-to-Peer Game Play
- Leaving a Peer-to-Peer Session
- Terminating a Peer-to-Peer Session

Initiating a Peer-to-Peer Session

A peer-to-peer game can be launched directly by the user, or *lobby-launched* by a *lobby client* application that resides on the user's computer. This documentation will assume that the game is lobbyable, and can communicate with the lobby client.

[C++]
One of the first steps you should take is to determine whether your game was lobby-launched. To do so, create and initialize a lobbied application object (CLSID_DirectPlay8LobbiedApplication). When you do so, you pass the object a pointer to your lobbied application message handler. This message handler receives messages directly from the lobbied application object, and indirectly from the lobby client and the lobby.

- If the application was lobby-launched, the **IDirectPlay8LobbiedApplication::Initialize** method returns a connection handle for the lobby client and a **DPL_MSGID_CONNECT** message is sent to your lobbied application message handler. The **pdplConnectionSettings** member of the associated structure points to a **DPL_CONNECTION_SETTINGS** structure that contains connection information such as address objects for the members of the session.
- If the application was not lobby launched, you will receive neither the connection handle, nor the message. However, if you call **IDirectPlay8LobbiedApplication::SetAppAvailable**, a lobby client can later connect your running application to a session by sending your lobbied application message handler a **DPL_MSGID_CONNECT** message.

You should also create and initialize a peer object (CLSID_DirectPlay8Peer). This object will be your primary means of communicating with Microsoft® DirectPlay®, and the other users in the session. If you want to have multiple players in the session, you must create a separate instance of this object for each player.

[Visual Basic]
One of the first steps you should take is to determine whether your game was lobby-launched. To do so, create a **DirectPlay8LobbiedApplication** object and a **DirectPlay8LobbyEvent** object. Register the **DirectPlay8LobbyEvent** object with Microsoft® DirectPlay® by calling **DirectPlay8LobbiedApplication.RegisterMessageHandler**. The **DirectPlay8LobbyEvent** object is essentially an event handler that receives notifications directly from the lobbied application object, and indirectly from the lobby client and the lobby. It is not provided by DirectPlay and must be implemented by your application. See the reference documentation for details.

If the application was lobby-launched, DirectPlay will call your **DirectPlay8LobbyEvent.Connect** method. The *dlNotify* parameter will contain a **DPL_MESSAGE_CONNECT** type with connection information. such as address objects for the members of the session.

You should also create a **DirectPlay8Peer** object and register a **DirectPlay8Event** notification handler object by calling **DirectPlay8Peer.RegisterMessageHandler**. These objects will be your primary means of communicating with DirectPlay and the other users in the session. If you want to have multiple players in the session, you must create a separate instance of these objects for each player.

Selecting a Service Provider for a Peer-to-Peer Session

The service provider is your network connection. Most games use either the TCP/IP or modem service provider, but Microsoft® DirectPlay® also provides support for serial and IPX connections.

[C++]
If your user was connected to the session by a lobby client, you can determine the appropriate service provider by examining the **DPL_CONNECTION_SETTINGS** structure that accompanies the **DPL_MSGID_CONNECT** message. Otherwise, you may need to determine which service provider to use, perhaps by querying the user. You can use the peer object's **IDirectPlay8Peer::EnumServiceProviders** method to enumerate the available service providers. See Using DirectPlay Enumerations for further discussion.

[Visual Basic]
If your user was connected to the session by a lobby client, you can determine the appropriate service provider by examining the **DPL_MESSAGE_CONNECT** structure that accompanies the **DirectPlay8LobbyEvent.Connect** method. Otherwise, you may need to determine which service provider to use, perhaps by querying the user. You can also use the peer object's **DirectPlay8Peer.GetServiceProvider** enumerate the available service providers.

Once you have selected a service provider, you can then create a DirectPlay address object for your user (a *device address)*. You will use this address to identify your device with a number of DirectPlay methods. See DirectPlay Addressing for a detailed discussion of DirectPlay addresses and address objects.

Selecting a Host for a Peer-to-Peer Session

[C++]
Although most aspects of peer-to-peer games can be handled by the various users' communicating directly with each other, there are some tasks that must have a single owner. These tasks are handled by the game host. To join a session, you must know the address of the session's host. A common way to select a host is through a lobby server. In that case, when a user's application is connected to the session, the connection settings that you receive with the **DPL_MSGID_CONNECT** message include the host's address object. To find out who the session host is:

- Check the **dwFlags** member of the **DPL_CONNECTION_SETTINGS** structure that is returned. If that member is set to DPLCONNECTSETTINGS_HOST, your system is the host.

- If the DPLCONNECTSETTINGS_HOST flag is not set, then you can get the address of the host from the **pdp8HostAddress** member.

You can also create a broadcast session, perhaps on a LAN subnet, by advertising yourself as a session host. To do so call **IDirectPlay8Peer::SetPeerInfo** to set the player's name and then call **IDirectPlay8Peer::Host** to advertise yourself as a potential host. You specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** structure that is passed through the *pdnAppDesc* parameter of **IDirectPlay8Peer::Host**.

To allow your user to examine the available sessions and hosts, you can enumerate the available hosts by calling **IDirectPlay8Peer::EnumHosts**. Once the user has selected a session, you can request a connection.

[Visual Basic]
Although most aspects of peer-to-peer games can be handled by the various users' communicating directly with each other, there are some tasks that must have a single owner. These tasks are handled by the game host. To join a session, you must know the address of the session's host. A common way to select a host is through a lobby server. In that case, when a user's application is connected to the session, the connection settings that you receive when Microsoft® DirectPlay® calls your **DirectPlay8LobbyEvent.Connect** method include the host's address object. To find out who the session host is:

- Check the **dwFlags** member of the **DPL_CONNECTION_SETTINGS** structure that is contained in the **DPL_MESSAGE_CONNECTION_SETTINGS** passed as the *dlNotify* parameter. If that member is set to DPLCONNECTSETTINGS_HOST, your system is the host.
- If the DPLCONNECTSETTINGS_HOST flag is not set, then you can get the address of the host from the **pdp8HostAddress** member.

You can also create a broadcast session, perhaps on a LAN subnet, by advertising yourself as a session host. To do so call **DirectPlay8Peer.SetPeerInfo** to set the player's name and then call **DirectPlay8Peer.Host** to advertise yourself as a potential host. You specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** type that is passed through the *pdnAppDesc* parameter of **IDirectPlay8Peer.Host**.

To allow your user to examine the available sessions and hosts, you can enumerate the available hosts by calling **DirectPlay8Peer.EnumHosts**. Once the user has selected a session, you can request a connection.

Connecting to a Peer-to-Peer Session

[C++]

Unless you are the session host, you will need to connect your player to the session. To do so, you must have the address of the session host. If your application was connected by a lobby client, you can obtain the host's address by calling **IDirectPlay8LobbiedApplication::GetConnectionSettings**. You can also obtain the address by enumerating the available hosts. The information returned by the enumeration includes each host's addresses, and a **DPN_APPLICATION_DESC** structure that describes the associated session.

To ask to join a session, call **IDirectPlay8Peer::SetPeerInfo** to set your player's name, and then call **IDirectPlay8Peer::Connect** with the selected host's address to connect to the session.

When a player attempts to join a session, the host receives a **DPN_MSGID_INDICATE_CONNECT** message. To accept the player into the session, return S_OK. Returning any other value rejects the request. In either case, the player will receive a **DPN_MSGID_CONNECT_COMPLETE** message that contains your response. If the host accepted the connection, the **hResultCode** member of the associated structure will be set to S_OK. If not, **hResultCode** will be set to DPNERR_HOSTREJECTEDCONNECTION.

The host can define a player context value when they receive the **DPN_MSGID_INDICATE_CONNECT** message, however the player ID will not yet be defined. The host can also wait to define a player context value until they receive a **DPN_MSGID_CREATE_PLAYER** message, which includes the player ID. Ordinary players to not receive a **DPN_MSGID_INDICATE_CONNECT** message.

Once the new player is connected, each member of the session, including the host, receives a **DPN_MSGID_CREATE_PLAYER** message announcing the new player. The structure associated with the message contains the player ID that you will use to send messages to that player. Peers that are not hosts must define the player context value when they handle this message. Once a peer or host has returned from handling this message, that player context value is set for the session, and cannot be changed. See Using Player Context Values for more discussion of player context values.

[Visual Basic]
Unless you are the session host, you will need to connect your player to the session. To do so, you must have the address of the session host. If your application was connected by a lobby client, you can obtain the host's address by calling **DirectPlay8LobbiedApplication.GetConnectionSettings**. You can also obtain the address by enumerating the available hosts. The information returned by the enumeration includes each host's addresses, and a **DPN_APPLICATION_DESC** structure that describes the associated session.

To ask to join a session, call **DirectPlay8Peer.SetPeerInfo** to set your player's name, and then call **DirectPlay8Peer.Connect** with the selected host's address to connect to the session.

When a player attempts to join a session, the Microsoft® DirectPlay® calls the host's **DirectPlay8Event.IndicateConnect** method. To accept the player into the session, set the method's *fRejectMsg* parameter to False before returning. Setting *fRejectMsg* to True rejects the request. In either case, the player's **DirectPlay8Event.ConnectComplete** method will be called with the response. If the host accepted the connection, the **hResultCode** member of the **DPNMSG_CONNECT_COMPLETE** type will be set to 0. If the request was rejected or failed for some other reason, **hResultCode** will be set to an error code.

Once the new player is connected, DirectPlay announces the new player by calling **DirectPlay8Event.CreatePlayer** for each member of the session, including the host. The *lPlayerID* parameter contains the player ID that you will use to send messages to that player.

Managing a Peer-to-Peer Session

The session host is responsible for managing the session, including:

- Managing the list of session members and their network addresses
- Deciding whether a new user is allowed to join the session.
- Notifying all members when a new user joins the session, and passing them the new user's address.
- Providing new users with the current game state
- Notifying all users when a user leaves the session

[C++]
When players attempt to join a session, the host will receive a **DPN_MSGID_INDICATE_CONNECT** message. To accept the player into the session return S_OK. Returning any other value rejects the request. In either case, the player will receive a **DPN_MSGID_CONNECT_COMPLETE** message that contains your response.

The host can remove a player from the session by calling **IDirectPlay8Peer::DestroyPeer**. Other members of the session cannot call this method successfully. If you want to allow players to request that another player be removed from the session, you must send the request to the host with normal Microsoft® DirectPlay® messaging, and have the host handle the request.

[Visual Basic]
When players attempt to join a session, Microsoft® DirectPlay® will call the host's **DirectPlay8Event.IndicateConnect** method. To accept the player into the session set *fRejectMsg* to False. Setting *fRejectMsg* to any other value rejects the request. In either case, DirectPlay calls the player's **DirectPlay8Event.ConnectComplete** method with the response to the request.

The host can remove a player from the session by calling **DirectPlay8Peer.DestroyPeer**. Other members of the session cannot call this method successfully. If you want to allow players to request that another player be removed from the session, you must send the request to the host, and have the host handle the request.

Host Migration

While the host must be one of the initial members of the session, they may choose to leave before session is finished. When the host leaves the session, there are two possible outcomes :

- The session terminates.
- The host *migrates*, and another user becomes host.

[C++]
Sessions may or may not permit host migration. To enable host migration, the session organizer must set the DPNSESSION_MIGRATE_HOST flag in the **dwFlags** member of the **DPN_APPLICATION_DESC** structure when they set up the game. If this flag is not set, the session terminates when the host leaves.

If the DPNSESSION_MIGRATE_HOST flag is set, the host can still force the session to terminate by calling **IDirectPlay8Peer::TerminateSession**.

If DPNSESSION_MIGRATE_HOST flag is set and the host leaves the session, Microsoft® DirectPlay® will select a new session host. All remaining session members will receive a **DPN_MSGID_HOST_MIGRATE** message that includes the ID of the new host.

[Visual Basic]
Sessions may or may not permit host migration. To enable host migration, the session organizer must set the DPNSESSION_MIGRATE_HOST flag in the **dwFlags** member of the **DPN_APPLICATION_DESC** type when they set up the game. If this flag is not set, the session terminates when the host leaves or loses their connection.

If the DPNSESSION_MIGRATE_HOST flag is set, the host can still force the session to terminate by calling **DirectPlay8Peer.TerminateSession**.

If DPNSESSION_MIGRATE_HOST flag is set and the host leaves the session, Microsoft® DirectPlay® will select a new session host. DirectPlay will call the **DirectPlay8Event.HostMigrate** method of all remaining session members with the ID of the new host.

Normal Peer-to-Peer Game Play

In Microsoft® DirectPlay®, a message is essentially a block of game-related data that you send to one or more members of the session. DirectPlay does not specify the contents or format of the data block, it just provides a mechanism to transmit the data from one user to another. Once the game is underway, each session member will normally send a constant stream of messages to all other members of the session for the duration of the game. The primary purpose of these messages is to keep the game state synchronized, so that each user's application displays the same UI. However, messages can also be used for a variety of other game-specific purposes.

For many games, especially rapidly changing ones, you may have to manage your messaging carefully. DirectPlay throttles outgoing messages to a level that can be handled by the target. You will have be careful that you do not send messages too rapidly, and ensure that the most important messages get through. See Basic Networking for a discussion of how to effectively handle DirectPlay messaging.

[C++]
To send a message to another session member, call **IDirectPlay8Peer::SendTo**. That member will receive a **DPN_MSGID_RECEIVE** message with the data. To send a message to a specific player, set the *dpnid* parameter to the player ID that you received with the associated **DPN_MSGID_CREATE_PLAYER** message. You can also send a message to every player in the session by setting *dpnid* to DPNID_ALL_PLAYERS_GROUP. You can also define groups of players, and use a single **SendTo** call to send a message to all members of a group.

Note

You can also use the **IDirectPlay8Peer::SetPeerInfo** method to send information to other users. They will receive the information with a **DPN_MSGID_PEER_INFO** message. However, this way of transmitting information is not very efficient, and should not be used for normal messaging.

[Visual Basic]
To send a message to another session member, call **DirectPlay8Peer.SendTo**. DirectPlay will call that members **DirectPlay8Event.Receive** method with the data. To send a message to a specific player, set the *idSend* parameter to the player ID that you when your **DirectPlay8Event.CreatePlayer** method was called. You can also send a message to every player in the session by setting *idSend* to DPNID_ALL_PLAYERS_GROUP. You can also define groups of players, and use a single **SendTo** call to send a message to all members of a group.

Note

You can also use the **DirectPlay8Peer.SetPeerInfo** method to send information to other users. DirectPlay will call their **DirectPlay8Event.InfoNotify** method with the information. However, this way of transmitting information is not very efficient, and should not be used for normal messaging.

Using Groups

[C++]
Many games allow players to be organized into groups. For instance, strategy games typically allow individual players to be organized into groups that can then be directed as a single entity. Microsoft® DirectPlay® also allows the formation of groups of players. DirectPlay groups are essentially a way to simplify your messaging. Once you have defined a group, you can send a message to every group member with a single call to **IDirectPlay8Peer::SendTo**. While DirectPlay groups normally correspond to the groups that are defined by the game, you are free to create a group for any reason.

To create a DirectPlay group, call **IDirectPlay8Peer::CreateGroup** All session members will then receive a **DPN_MSGID_CREATE_GROUP** message with the details. The message will include a group ID that is used to send messages to the group.

Once the group is created, you then add players by calling **IDirectPlay8Peer::AddPlayerToGroup**. Session members will then receive a **DPN_MSGID_ADD_PLAYER_TO_GROUP** message with the IDs of the group and the player that was just added.

Once the group is established, you can send data to the group by calling **IDirectPlay8Peer::SendTo**, with the *dpnid* parameter set to the group ID. All group members will then receive a **DPN_MSGID_RECEIVE** message with the data.

To remove a player from a group, call **IDirectPlay8Peer::RemovePlayerFromGroup**. The session members will receive a **DPN_MSGID_DESTROY_PLAYER** message with the player's ID.

Finally, when you no longer need the group, you can destroy it by calling **IDirectPlay8Peer::DestroyGroup** All session members will then receive a **DPN_MSGID_DESTROY_GROUP** message with the group ID.

[Visual Basic]
Many games allow players to be organized into groups. For instance, strategy games typically allow individual players to be organized into groups that can then be directed as a single entity. Microsoft® DirectPlay® also allows the formation of groups of players. DirectPlay groups are essentially a way to simplify your messaging. Once you have defined a group, you can send a message to every group member with a single **DirectPlay8Peer.SendTo.** While DirectPlay groups normally correspond to the groups that are defined by the game, you are free to create a group for any reason.

To create a DirectPlay group, call **DirectPlay8Peer.CreateGroup**. DirectPlay will call all session members' **DirectPlay8Event.CreateGroup** method with the details.

The method's *lGroupID* parameter will be set to the group ID that you can use to send messages to the group.

Once the group is created, you then add players by calling **DirectPlay8Peer.AddPlayerToGroup**. DirectPlay will then call all members' **DirectPlay8Event.AddRemovePlayerGroup** with the IDs of the group and the player that was just added.

Once the group is established, you can send data to the group by calling **DirectPlay8Peer.SendTo**, with the *idSend* parameter set to the group ID. DirectPlay will call all group members' **DirectPlay8Event.Receive** method with the data.

To remove a player from a group, call **DirectPlay8Peer.RemovePlayerFromGroup**. DirectPlay will call the session members' **DirectPlay8Event.AddRemovePlayerGroup** method with the player's ID.

Finally, when you no longer need the group, you can destroy it by calling **DirectPlay8Peer.DestroyGroup**. DirectPlay will call all session members' **DirectPlay8Event.DestroyGroup** method with the group ID.

Leaving a Peer-to-Peer Session

[C++]
To leave a session, terminate the connection by calling **IDirectPlay8Peer::Close**. The session members will be notified with a **DPN_MSGID_DESTROY_PLAYER** message.

[Visual Basic]
To leave a session, terminate the connection by calling **DirectPlay8Peer.Close**. Microsoft® DirectPlay® will call the session members **DirectPlay8Event.DestroyPlayer** method with the *lPlayerID* parameter set to the player's ID.

If you are the session host, leaving also terminates the session unless you configured the session to allow host migration. See Host Migration for details.

Terminating a Peer-to-Peer Session

[C++]
When the session is over, the host should terminate the session by calling **IDirectPlay8Peer::TerminateSession**. This method terminates the session even if host-migration is enabled. All session members will be notified by a **DPN_MSGID_TERMINATE_SESSION** message. You should then perform any

necessary cleanup. To start another session, you must first call **IDirectPlay8Peer::Close**, and then **IDirectPlay8Peer::Initialize**.

If you registered your application as available for connection by calling **IDirectPlay8LobbiedApplication::SetAppAvailable**, a lobby client can offer to connect you to a new session by sending your lobbied application message handler a **DPL_MSGID_CONNECT** message. You must have first called **IDirectPlay8Peer::Close** and **IDirectPlay8Peer::Initialize**.

[Visual Basic]
When the session is over, the host should terminate the session by calling **DirectPlay8Peer.TerminateSession**. This method terminates the session even if host-migration is enabled. Microsoft® DirectPlay® will notify all session members by calling their **DirectPlay8Event.TerminateSession** method. You should then perform any necessary cleanup. To start another session, you must first call **DirectPlay8Peer.Close**, and then **DirectPlay8Peer.RegisterMessageHandler**.

If you registered your application as available for connection by calling **DirectPlay8LobbiedApplication.SetAppAvailable** a lobby client can offer to connect you to a new session by calling your **DirectPlay8LobbyEvent.Connect** method. You must have first called **DirectPlay8Peer.Close** and **DirectPlay8.RegisterMessageHandler**.

Client/Server Sessions

A client/server session consists of a collection of players, or clients, connected to a central server. As far as Microsoft® DirectPlay® is concerned, a client has no knowledge of any other clients, only the server. The messaging needed to run the game is between the individual clients and the server. DirectPlay does not provide direct client-to-client messaging, as it does for peer-to-peer sessions.

A client/server session requires two distinctly different applications:.

- The server application runs on a remote server. At a minimum, it serves as a central messaging hub and game host. The server must receive and handle all incoming messages from the clients, and send appropriate messages back out. Any transfer of data from one client to another must be handled by the server application.
- A client application runs on each players' computer. The primary function of this application is to handle the UI, and keep the player's game state synchronized with the server.

There are certain aspects of the session that can be handled by only one of these applications. For instance, updating a player's video display can only be done by the client application. However, many aspects of the processing needed to maintain the game universe can, at least in principle, be done by either application. Writing an

effective client/server game requires some careful consideration of how to divide the processing chores between the two applications.

This document describes the basic principles of client server games, and outlines how to implement client and server applications.

- Initiating a Client/Server Session
- Selecting a Service Provider for a Client
- Selecting a Client/Server Host
- Connecting to a Client/Server Session
- Managing a Client/Server Session
- Normal Client/Server Game Play
- Leaving a Client/Server Session
- Terminating a Client/Server Session

Initiating a Client/Server Session

A client/server game can be launched through a lobby, or directly by the server application.

The Server Application

Client/server games are often arranged through lobbies. The most straightforward way to launch the server is to implement it as a *lobbyable* application. This approach provides a way to launch the server, and supports communication between server and lobby during the course of the session. See DirectPlay Lobbies for further discussion.

A server can also be directly launched, and then advertise itself as available and wait for clients to connect. See Selecting a Client/Server Host for details.

[C++]
Once the server application has been launched, it should initialize itself by calling **IDirectPlay8Server::Initialize**. As with other similar Microsoft® DirectPlay® methods, the primary purpose of initialization is to provide DirectPlay with a pointer to your callback message handler. You should also call **IDirectPlay8Server::SetServerInfo** to describe the current game. Clients cannot connect to a server until this method has been called.

[Visual Basic]
Once the server application has been launched, it should register it's **DirectPlay8Event** notification handler object. The **DirectPlay8Event** object is essentially an event handler that receives notifications from Microsoft® DirectPlay®. It is not provided by DirectPlay and must be implemented by your application. See the reference documentation for details. You should also call

DirectPlay8Server.SetServerInfo to describe the current game. Clients cannot connect to a server until this method has been called.

The Client Application

[C++]
One of the first steps you should take is to determine whether your game was lobby-launched. To do so, create and initialize a lobbied application object (CLSID_DirectPlay8LobbiedApplication). When you do so, you pass the object a pointer to your lobbied application message handler. This message handler receives messages directly from the lobbied application object, and indirectly from the lobby client and the lobby.

- If the application was lobby-launched, the **IDirectPlay8LobbiedApplication::Initialize** method returns a connection handle for the lobby client and a **DPL_MSGID_CONNECT** message is sent to your lobbied application message handler. The **pdplConnectionSettings** member of the associated structure points to a **DPL_CONNECTION_SETTINGS** structure that contains connection information such as an address object for the server.
- If the application was not lobby launched, you will receive neither the connection handle, nor the message. However, if you call **IDirectPlay8LobbiedApplication::SetAppAvailable**, a lobby client can later connect your running application to a session by sending your lobbied application message handler a **DPL_MSGID_CONNECT** message.

You should also create and initialize a client object (CLSID_DirectPlay8Client). This object will be your primary means of communicating with Microsoft® DirectPlay® and the server. If you want to have multiple players in the session, you must create a separate instance of this object for each player.

[Visual Basic]
One of the first steps you should take is to determine whether your game was lobby-launched. To do so, create a **DirectPlay8LobbiedApplication** object and a **DirectPlay8LobbyEvent** object. Register the **DirectPlay8LobbyEvent** object with Microsoft® DirectPlay® by calling **DirectPlay8LobbiedApplication.RegisterMessageHandler**. The **DirectPlay8LobbyEvent** object is essentially an event handler that receives notifications directly from the lobbied application object, and indirectly from the lobby client and the lobby. It is not provided by DirectPlay and must be implemented by your application. See the reference documentation for details.

If the application was lobby-launched, DirectPlay will call your **DirectPlay8LobbyEvent.Connect** method. The *dlNotify* parameter will contain a

DPL_MESSAGE_CONNECT type with connection information. such as address objects for the members of the session.

You should also create a **DirectPlay8Client** object and register a **DirectPlay8Event** notification handler object by calling **DirectPlay8Client.RegisterMessageHandler**. These objects will be your primary means of communicating with the server.

Selecting a Service Provider for a Client

The service provider is your network connection. Most games use either the TCP/IP or modem service provider, but Microsoft® DirectPlay® also provides support for serial and IPX connections.

[C++]
If your user was connected to the session by a lobby client, you can determine the appropriate service provider by examining the **DPL_CONNECTION_SETTINGS** structure that accompanies the **DPL_MSGID_CONNECT** message. Otherwise, you may need to determine which service provider to use, perhaps by querying the user. You can use the client object's **IDirectPlay8Client::EnumServiceProviders** method to enumerate the available service providers. See Using DirectPlay Enumerations for further discussion.

[Visual Basic]
If your user was connected to the session by a lobby client, you can determine the appropriate service provider by examining the **DPL_CONNECTION_SETTINGS** type that you receive when DirectPlay calls your **DirectPlay8LobbyEvent.Connect**. Otherwise, you may need to determine which service provider to use, perhaps by querying the user. You can use the client object's **DirectPlay8Client.GetServiceProvider** method to enumerate the available service providers. See Using DirectPlay Enumerations for further discussion.

Once you have selected a service provider, you can then create a DirectPlay address object for your user (a *device address)*. You will use this address to identify your device with a number of DirectPlay methods. See DirectPlay Addressing for a detailed discussion of DirectPlay addresses and address objects.

Selecting a Client/Server Host

[C++]
By definition, the server application hosts the session. To join a session, a client application must determine the host server's address. A common way to select a host is through a lobby server. In that case, when a user's application is connected to the

session, the connection settings that you receive with the **DPL_MSGID_CONNECT** message include the host's address object. The **pdp8HostAddress** member of the associate structure points to an address object with the host's address.

Servers using an IP or IPX service provider can also create a *broadcast session,* perhaps on a LAN subnet, by advertising themselves as session hosts. To create a broadcast session, specify call **IDirectPlay8Server::SetServerInfo** specify the server settings. Then call **IDirectPlay8Server::Host** to advertise the server as a session host. You specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** structure that is passed through *pdnAppDesc* parameter of **IDirectPlay8Server::Host**.

To allow your user to look at the available sessions and hosts, a client application can query for available hosts by calling **IDirectPlay8Client::EnumHosts**. Once the user has selected a host, you can request a connection.

[Visual Basic]
By definition, the server application hosts the session. To join a session, a client application must determine the host server's address. A common way to select a host is through a lobby server. In that case, when a user's application is connected to the session, the connection settings that you receive when your **DirectPlay8LobbyEvent.Connect** method is called includes a connection ID and the host's address.

Servers using an IP or IPX service provider can also create a *broadcast session,* perhaps on a LAN subnet, by advertising themselves as session hosts. To create a broadcast session, specify call **DirectPlay8Server.SetServerInfo** specify the server settings. Then call **DirectPlay8Server.Host** to advertise the server as a session host. You specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** structure that is passed through *AppDesc* parameter of **DirectPlay8Server.Host**.

To allow your user to look at the available sessions and hosts, a client application can query for available hosts by calling **DirectPlay8Client.EnumHosts**. Once the user has selected a host, you can request a connection.

Connecting to a Client/Server Session

All clients must explicitly join the session by connecting to the host, even if the session has been arranged through a lobby. A connection establishes the client as a member of the session, and provides the host with the information it needs to communicate with the client. The host has the option of accepting or rejecting a connection request.

The Server Application

[C++]
When a client attempts to join a session, the host receives a **DPN_MSGID_INDICATE_CONNECT** message. To accept the player into the session, return S_OK. Returning any other value rejects the request. In either case, the client will receive a **DPN_MSGID_CONNECT_COMPLETE** message that contains your response. You can define a player context value at this time, or wait until you receive a **DPN_MSGID_CREATE_PLAYER** message. See Using Player Context Values for more discussion of player context values.

If the player is successfully added to the session, all clients and the server will receive a **DPN_MSGID_CREATE_PLAYER** message with the new player's ID (DPNID). If you want to define a player context value, and have not yet done so, you must define it before your message handler returns from handling this message. Once it has done so, you cannot change the player context value.

[Visual Basic]
When a client attempts to join a session, the Microsoft® DirectPlay® calls the server's **DirectPlay8Event.IndicateConnect** method. To accept the player into the session, set the method's *fRejectMsg* parameter to False before returning. Setting *fRejectMsg* to True rejects the request. In either case, the player's **DirectPlay8Event.ConnectComplete** method will be called with the response. If the host accepted the connection, **hResultCode** member of the **DPNMSG_CONNECT_COMPLETE** type will be set to 0. If the request was rejected or failed for some other reason, **hResultCode** will be set to an error code.

If the player is successfully added to the session, DirectPlay will call the **DirectPlay8Event.CreatePlayer** method for the server and all client's with the new player's ID (DPNID).

The Client Application

[C++]
To connect to a session, you must have the address of the session host. If your application was connected by a lobby client, you can obtain the host's address by calling **IDirectPlay8LobbiedApplication::GetConnectionSettings**.

If you not have the address of a session host and you are using either an IP or IPX service provider, you can look for broadcast sessions by calling **IDirectPlay8Client::EnumHosts** and enumerating the available hosts. You can also obtain the address by enumerating the available hosts. The information returned by the enumeration includes each host's address, the device use to reach the host, and a **DPN_APPLICATION_DESC** structure that describes the associated session.

To ask to join a session, call **IDirectPlay8Client::SetClientInfo** to set your player's name, and then call **IDirectPlay8Client::Connect** with the selected host's address to connect to the session.

Your message handler will receive a **DPN_MSGID_CONNECT_COMPLETE** message with the host's response. If the host accepted the connection, the **hResultCode** member of the associated structure will be set to S_OK. If not, **hResultCode** will be set to DPNERR_HOSTREJECTEDCONNECTION.

[Visual Basic]
To connect to a session, you must have the address of the session host. If your application was connected by a lobby client, you can obtain the host's address by calling **DirectPlay8LobbiedApplication.GetConnectionSettings**.

If you not have the address of a session host and you are using either an IP or IPX service provider, you can look for broadcast sessions by calling **DirectPlay8Client.EnumHosts** and enumerating the available hosts. You can also obtain the address by enumerating the available hosts. The information returned by the enumeration includes each host's address, the device use to reach the host, and a **DPN_APPLICATION_DESC** type that describes the associated session.

To ask to join a session, call **DirectPlay8Client.SetClientInfo** to set your player's name, and then call **DirectPlay8Client.Connect** with the selected host's address to connect to the session.

Microsoft® DirectPlay® will call your **DirectPlay8Event.ConnectComplete** method with the host's response. If the host accepted the connection, the **hResultCode** member of the **DPNMSG_CONNECT_COMPLETE** type will be set to 0. If the request was rejected or failed for some other reason, **hResultCode** will be set to an error code.

Managing a Client/Server Session

As host, the server is responsible for managing the course of the session. The details will depend on how the application is designed, but a session host's duties include, at a minimum,:

- Managing the list of session members and their network addresses. Microsoft® DirectPlay® handles some of this task, but server applications typically need to manage more player data than is provided by DirectPlay.
- Deciding whether a new user is allowed to join the session.
- Providing new users with the current game state.

[C++]

When a player attempts to join a session, the host receives a **DPN_MSGID_INDICATE_CONNECT** message. To accept the player into the session return S_OK. Returning any other value rejects the connection request. In either case, the player will receive a **DPN_MSGID_CONNECT_COMPLETE** message that contains your response.

The host can remove a player from the session by calling **IDirectPlay8Server::DestroyClient**.

[Visual Basic]
When a player attempts to join a session, DirectPlay calls the host's **DirectPlay8Event.IndicateConnect** method. To accept the player into the session set *fRejectMsg* to False. Setting *fRejectMsg* to any other value rejects the request. In either case, DirectPlay calls the player's **DirectPlay8Event.ConnectComplete** method with the response to the request.

The host can remove a player from the session by calling **DirectPlay8Server.DestroyClient**.

Normal Client/Server Game Play

In Microsoft® DirectPlay®, a message is essentially a block of game-related data that is sent from client to server or vice versa. DirectPlay does not specify the contents or format of the data block, it just provides a mechanism to transmit the data. Once the game is underway, each client will normally send a constant stream of messages to the server, and vice versa, for the duration of the game. The primary purpose of these messages is to keep the game state synchronized, so that each user's application displays the same UI. However, messages can also be used for a variety of other game-specific purposes.

For many games, especially rapidly changing ones, you may have to manage your messaging carefully. DirectPlay throttles outgoing messages to a level that can be handled by the target. You will have be careful that you do not send messages too rapidly, and ensure that the most important messages get through. See Basic Networking for a discussion of how to effectively handle DirectPlay messaging.

The Server Application

[C++]
To send a message to a client, call **IDirectPlay8Server::SendTo**. The client will receive a **DPN_MSGID_RECEIVE** message with the data.

[Visual Basic]

To send a message to a client, call **DirectPlay8Server.SendTo**. DirectPlay will call the client's **DirectPlay8Event.Receive** method with the data.

The Client Application

[C++]
To send a message to the server, call **IDirectPlay8Client::Send**. The server will receive a **DPN_MSGID_RECEIVE** message with the data.

[Visual Basic]
To send a message to the server, call **DirectPlay8Client.Send**. DirectPlay will call the server's **DirectPlay8Event.Receive** method with the data.

Note

DirectPlay does not provide a mechanism for clients to communicate with other clients, only with the server. Any client-client communication must be implemented by the server application.

Using Groups

[C++]
Many games allow players to be organized into groups. For example, in a squad-based game, every player in the squad could be a member of a group. DirectPlay allows servers in a client/server game to create groups of players. While DirectPlay groups typically correspond to the groups that defined by the game, you are free to create a group for any reason. DirectPlay groups are essentially a way to simplify your messaging. Once you have defined a group, you can send a message to every group member with a single **IDirectPlay8Server::SendTo** call.

To create a DirectPlay group, call **IDirectPlay8Server::CreateGroup**. Your message handler will then receive a **DPN_MSGID_CREATE_GROUP** message with the details. The message will include a group ID that is used to send messages to the group. Once the group is created, you then add players by calling **IDirectPlay8Server::AddPlayerToGroup**.

Once the group is established, you can send data to the group by calling **IDirectPlay8Server::SendTo**, with the *dpnid* parameter set to the group ID. All group members will then receive a **DPN_MSGID_RECEIVE** message with the data.

To remove a player from a group, call **IDirectPlay8Server::RemovePlayerFromGroup**. Finally, when you no longer need the group, you can destroy it by calling **IDirectPlay8Server::DestroyGroup**

[Visual Basic]
Many games allow players to be organized into groups. For example, in a squad-based game, every player in the squad could be a member of a group. DirectPlay allows servers in a client/server game to create groups of players. While DirectPlay groups typically correspond to the groups that defined by the game, you are free to create a group for any reason. DirectPlay groups are essentially a way to simplify your messaging. Once you have defined a group, you can send a message to every group member with a single **DirectPlay8Server.SendTo** call.

To create a DirectPlay group, call **DirectPlay8Server.CreateGroup**. DirectPlay will call your **DirectPlay8Event.CreateGroup** method with the details. The notification will include a group ID that is used to send messages to the group. Once the group is created, you then add players by calling **DirectPlay8Server.AddClientToGroup**.

Once the group is established, you can send data to the group by calling **DirectPlay8Server.SendTo**, with the *idSend* parameter set to the group ID. DirectPlay will then call the group members' **DirectPlay8Event.Receive** method with the data.

To remove a player from a group, call **DirectPlay8Server.RemoveClientFromGroup**. Finally, when you no longer need the group, you can destroy it by calling **DirectPlay8Server.DestroyGroup**.

Leaving a Client/Server Session

[C++]
A client can leave a session by calling **IDirectPlay8Client::Close**. The server is notified with a **DPN_MSGID_DESTROY_PLAYER** message.

[Visual Basic]
A client can leave a session by calling **DirectPlay8Client.Close**. Microsoft® DirectPlay® notifies the server by calling its **DirectPlay8Event.DestroyPlayer** method.

Terminating a Client/Server Session

[C++]
To terminate a client/server session, the server calls **IDirectPlay8Server::Close**. There is no host migration in a client/server session, so this method terminates all connections and closes the session. The clients are notified of the session end by a **DPN_MSGID_TERMINATE_SESSION** message.

The server will then receive a **DPN_MSGID_DESTROY_PLAYER** message for each player, including itself. **IDirectPlay8Server::Close** is synchronous, and will not return until all the **DPN_MSGID_DESTROY_PLAYER** messages have been processed. Once **IDirectPlay8Server::Close** has returned, you can safely shut down the server application.

[Visual Basic]
To terminate a client/server session, the server calls **DirectPlay8Server.Close**. There is no host migration in a client/server session, so this method terminates all connections and closes the session. Microsoft® DirectPlay® notifies the clients by calling their **DirectPlay8Event.TerminateSession** method.

DirectPlay then calls the server's **DirectPlay8Event.DestroyPlayer** method for each player, including itself. **DirectPlay8Server.Close** is synchronous, and will not return until all the **DirectPlay8Event.DestroyPlayer** method calls have been processed. Once **DirectPlay8Server.Close** has returned, you can safely shut down the server application.

DirectPlay Lobbies

A lobby is an application whose primary purpose is to help users arrange multiplayer games. The lobby is usually an application that is hosted on a remote server. The user visits the lobby, typically through the Internet, and either sets up a game session or joins a session started by someone else. The lobby application then launches the group's individual game applications, and the game is underway.

Because many multiplayer games are arranged through lobbies, most games based on Microsoft® DirectPlay® must be able to interact with lobby applications. Conversely, because most lobbies will want to support DirectPlay-based games, the lobby application must be able to interact with the game application. This document discusses how to enable a Microsoft DirectX® game to interact with a lobby, and vice versa.

- DirectPlay Lobby Architecture
- Lobby Servers
- Lobby Clients
- Lobbyable Applications

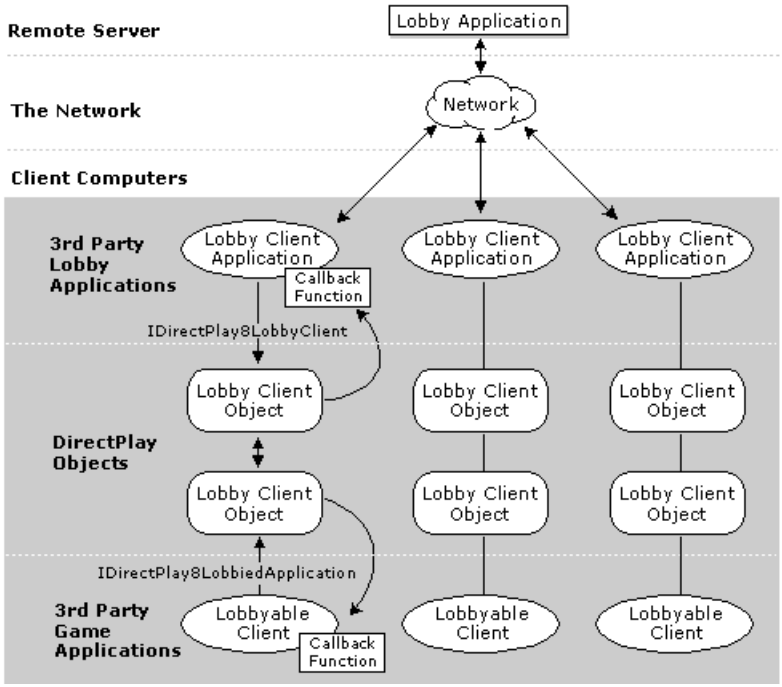
DirectPlay Lobby Architecture

The process of arranging and managing a session of a multiplayer game based on Microsoft® DirectPlay® involves five separate components. The lobby server application is a third-party application that typically resides on a remote server and is

accessed through the Internet. The remaining four components are installed on each player's computer.

- *Lobby client*. The lobby client is a third-party application that communicates with the lobby server. It also communicates with the user's game application through the DirectPlay lobby client object.
- *Lobbyable game application*. The lobbyable game application is a third-party application that uses the DirectPlay lobbied application object to communicate with the lobby client, and through the lobby client with the lobby server.
- *DirectPlay lobby client object*.
- *DirectPlay lobbied application object*.

The two DirectPlay objects act as links between the game application and the lobby client. They communicate with each other through private interfaces. The following graphic shows how these pieces are linked, and how they communicate.:



Lobby Servers

The lobby server is an application whose primary purpose is to enable players to meet and arrange games. It is typically located on a remote computer, and accessed over the Internet. Lobby servers often perform a variety of other functions such as hosting chat rooms, posting news and information, and selling merchandise.

To manage multiplayer games, a lobby server typically handles a variety of tasks, including:

- Managing the network addresses of the various game sessions and players.
- Launching a session by launching the associated game applications on the players' computers.
- Adding players to an ongoing session.
- Connecting the various computers in a session to the correct network addresses.
- Keeping track of changes in the session, such as players leaving the game or changes in the game's host.

The details of the lobby server application depend on what kind of services the vendor wants to offer. Microsoft® DirectPlay® does not specify how a lobby server should be implemented nor how it should communicate with its users' computers. However, lobby vendors must implement and distribute to their users a lobby client that is compatible with DirectPlay.

Lobby Clients

A lobby client is an application that is implemented by the lobby server vendor and installed on each player's computer. It handles communication between the players and their game applications, and the lobby server. A common way to install a lobby client is to have the user download it from the lobby server's Web site as part of the sign-up procedure.

The following is a typical scenario.

1. A new player goes to the Web site and signs up.
2. As part of the sign-up procedure, the lobby client is downloaded to the client's computer.
3. The player determines which to play and asks to join a session.
4. The Web site launches the lobby client on the player's computer. A typical launch mechanism is a URL that points to the lobby client's executable file.
5. The lobby client handles the mechanics of arranging the session, and then launches the user's game application.
6. If the game is a lobbyable application, the lobby client enables the game application to communicate with the lobby server. This connection enables the lobby server to keep track of events such as players entering and leaving the game and host migration.

Lobby clients do not necessarily have to be linked to a remote server. In another scenario the user launches the lobby client directly. The lobby client then lists the available games and sessions, perhaps among the people connected to the user's LAN subnet. Once the user chooses a game and session, the lobby client launches the game.

This section discusses some the general features of a lobby client. For more information on communicating between a lobby client and its associated lobby server, see Communicating with a Lobbyable Game.

For more information on implementation details, see Implementing a Lobby Client or the LobbyClient sample application included in the SDK.

Communicating with a Lobbyable Game

Communication between a lobby client and its associated lobby server can be handled in any way that is convenient. Microsoft® DirectPlay® specifies only how the lobby client must communicate with a lobbyable game application.

[C++]
Lobby clients do not communicate directly with game applications. Instead, they communicate with the DirectPlay lobby client object (CLSID_DirectPlay8LobbyClient) through its **IDirectPlay8LobbyClient** interface. If the game application is lobbyable, the lobbied application object then passes messages to the game. The **IDirectPlay8LobbyClient** interface enables the lobby client to do the following.

- Enumerate the lobbyable applications on the user's system.
- Launch the application, if it is not already running, and connect it to the session.
- Release the application from a session, and close the link with the lobby client
- Send a message to a lobbied application that was launched or connected by the lobby client.

[Visual Basic]
Lobby clients do not communicate directly with game applications. Instead, they communicate with the **DirectPlay8LobbyClient** object. If the game application is lobbyable, the lobbied application object then passes messages to the game. The **DirectPlay8LobbyClient** object enables the lobby client to do the following.

- Enumerate the lobbyable applications on the user's system.
- Launch the application, if it is not already running, and connect it to the session.
- Release the application from a session, and close the link with the lobby client
- Send a message to a lobbied application that was launched or connected by the lobby client.

Note

A DirectPlay lobby client can launch any application, whether or not it is lobbyable. However, only lobbyable applications can use DirectPlay to communicate back to the lobby client during the course of the game.

[C++]
The lobby client object communicates with the lobby client through a callback function that is implemented by the lobby client. A function pointer is passed to the

lobby client object during initialization. This callback function enables the lobby client object to send the lobby client information such as:

[Visual Basic]
The lobby client object communicates with the lobby client through a **DirectPlay8LobbyEvent** message handler object. The **DirectPlay8LobbyEvent** object is essentially an event handler that receives notifications directly from the lobby client object, and indirectly from the application. It is not provided by DirectPlay and must be implemented by your application. See the reference documentation for details. You must register this object with DirectPlay by calling **DirectPlay8LobbyClient.RegisterMessageHandler**.

The **DirectPlay8LobbyEvent** object enables the lobby client object to send the lobby client information such as:

- Connection information.
- Connection status.
- Session status, including connection, disconnection, and host migration.
- Messages from the application to the lobby client.

Launching an Application

When you launch an application, you can attempt to pass a block of game-specific information to the application. When a lobbyable application is launched by a lobby client, the application creates and initializes a lobbied application object. The information is passed to the game through the initialization method.

When the lobbied application object is initialized, the lobby client receives a message indicating that the user has been connected. One primary purpose of this message is to notify the lobby client that the application is lobbyable. If the lobby client has not received a connect message after a reasonable period of time following the launch, the game is not lobbyable and you can stop attempting to make this connection.

After Launching an Application

The lobby client has relatively little to do once a lobbyable game is in progress. Depending on the game topology, most of the user's messages are sent directly to the other players or to the game server,. However, DirectPlay sends the lobby client messages in response to events such as disconnection and host migration. These messages enable the lobby client to pass such status changes to the lobby server. For instance, if the host migrates, the lobby server can update its UI to indicate the new host.

The application can also pass messages to the lobby client. This message can contain virtually anything, and it can be used for any purpose. The lobby client typically passes the data to the lobby server for processing. For example, at the end of the

game, the application might send a message that enables the lobby server to update its high-score list.

Lobbyable Applications

Lobbyable applications are designed to work with a lobby client based on Microsoft® DirectPlay®. While a lobby client can use DirectPlay to launch any application, lobbyable applications have a number of advantages.

- The lobby client receives automatic updates when game status changes.
- The lobby client can use a standard API to communicate with the application.
- The application can use a standard API to communicate with the lobby client.

In short, DirectPlay virtually eliminates the need for game-specific or lobby client-specific communication code. You can use a standard API for everything with little or no modification for the particular game or lobby client.

This section discusses some of the general features of a lobbyable application. For more information, see Launching a Lobbyable Application.

For a discussion of implementation details, see Implementing a Lobbyable Application or SDK samples such as SimplePeer, or StagedPeer.

Launching a Lobbyable Application

[C++]
One of the first things a lobbyable application should do after it is launched is create a lobbied application object. Among other things, this object enables your application to determine whether it was lobby-launched. A lobbied application must also implement a message-handler callback function to receive messages from the lobby client. The basic procedure is:

- Create a lobbied application object.
- Initialize the object.
- If the initialization method returns a valid connection handle, your application was lobby launched.
- Examine the user context value that is returned by the initialization method. This value might contain game-specific information from the lobby client.
- Examine the connection message received through the lobbied application message handler. This message carries with it a variety of information, including the ID that you will use to send messages to the lobby client.

Once an application has been successfully lobby launched, Microsoft® DirectPlay® can automatically send status updates to the lobby client when events such as host migration occur. To enable automatic status updates, call the **RegisterLobby** method of the **IDirectPlay8Peer**, **IDirectPlay8Client**, or **IDirectPlay8Server** interface. You can also use the lobbied application interface to send messages to the lobby client.

Be aware that your message handler function might receive messages from the lobby client before the initialization method returns. In addition to the connection message, the callback function receives messages when the lobby client changes connection settings, or it breaks the connection. The lobby client can also send messages directly to your message handler that contain game-specific information.

Note

It is possible to receive messages from more than one thread. To handle messaging properly, your lobbied application callback function should be re-entrant.

[Visual Basic]
One of the first things a lobbyable application should do after it is launched is create a **DirectPlay8LobbiedApplication** object. Among other things, this object enables your application to determine whether it was lobby-launched. A lobbied application must also implement a message-handler callback function to receive messages from the lobby client. The basic procedure is:

- Create a **DirectPlay8LobbiedApplication** object.
- Register a **DirectPlay8LobbyEvent** message handler object by calling **DirectPlay8LobbiedApplication.RegisterMessageHandler**.
- If the application was lobby-launched, DirectPlay will call your **DirectPlay8LobbyEvent.Connect** method. The *dlNotify* parameter will contain a **DPL_MESSAGE_CONNECT** type with connection information. such as address objects for the members of the session.

Note

The **DirectPlay8LobbyEvent** object is essentially an event handler that receives notifications directly from the lobbied application object, and indirectly from the lobby client and the lobby. It is not provided by DirectPlay and must be implemented by your application. See the reference documentation for details.

Once an application has been successfully lobby launched, Microsoft® DirectPlay® automatically sends status updates to the lobby client when events such as host migration occur. You can also use the lobbied application interface to send messages to the lobby client.

Be aware that your message handler object might be called by the lobby client before the initialization method returns. In addition to the connection message, the DirectPlay calls the object when the lobby client changes connection settings, or it breaks the connection. The lobby client can also send messages directly to your message handler that contain game-specific information.

Basic Networking

This section covers some basic networking technology topics that you need to understand to write Microsoft® DirectPlay® applications. For a general discussion of networking technology, see one of the standard texts on the subject, such as *Computer Networks* by Andrew Tannenbaum.

- DirectPlay Service Providers
- DirectPlay Addressing
- DirectPlay Protocol
- Optimizing Network Usage
- Using the DirectX Protocol in an Application

DirectPlay Service Providers

Microsoft® DirectPlay® provides your application with a virtual network connection that enables you to communicate with other computers in the same way, regardless of type of network involved. To provide this level of abstraction, network connections are made through a service provider. Once you have selected a service provider, your application uses the appropriate DirectPlay methods to communicate with other computers in a session. The service provider handles the details of communicating over the selected network hardware.

DirectPlay includes service providers for four types of network connections: TCP/IP, IPX, modem, and serial. See DirectPlay Addressing for a discussion of how to select a service provider.

[Visual Basic

Note

To use modems on Microsoft Windows® 95 systems, you must install version 2.0 of the Telephony API (TAPI). You can download TAPI 2.0 from http://www.microsoft.com.

[C++]

Note

DirectPlay uses the telephony API (TAPI) to handle modem communication. The use of this API means that the code that is used to answer the phone must be in the message loop's thread. To use modems on Microsoft Windows® 95 systems, you must install version 2.0 of the Telephony API (TAPI). You can download TAPI 2.0 from http://www.microsoft.com.

DirectPlay Addressing

To deliver messages, each participant in a multiplayer game must have a unique address. Addresses can refer either to the computer that your application is running on (*device address*), or a computer that your application needs to communicate with (*host address*).

[C++]
Microsoft® DirectPlay® 8.0 represents addresses in the form of a URL string. That address string is then encapsulated in a DirectPlay address object that is passed as a parameter in or out of methods such as **IDirectPlay8Peer::Connect**.

This section describes two ways to handle DirectPlay addresses.

- DirectPlay URLs discusses how to construct the address string directly.
- DirectPlay Address Objects discusses how to manipulate the address string using the methods exposed by the address object's IDirectPlay8Address interface.

[Visual Basic]
Microsoft® DirectPlay® 8.0 represents addresses in the form of a URL string. That address string is then encapsulated in a **DirectPlay8Address** object that is passed as a parameter in or out of methods such as **DirectPlay8Peer.Connect**.

This section describes two ways to handle DirectPlay addresses.

- DirectPlay URLs discusses how to construct the address string directly.
- DirectPlay Address Objects discusses how to manipulate the address string using the methods exposed by the address object's IDirectPlay8Address interface.

DirectPlay URLs

Microsoft® DirectPlay® represents addresses as URLs. In general, URLs are strings that consist of three basic components in the following order: scheme, scheme separator, and data string.

All DirectPlay addresses use "x-directplay" as the scheme, and ":/" (a colon followed by a forward slash) as the scheme separator. Using ":/" as a separator implies that the data that follows is *opaque*. In other words, the data string does not conform to any Internet standard and should be passed to the receiving application without modification. All DirectPlay URLs thus have the following general form.

```
x-directplay:/[data string]
```

Note

Do not use "://" (a colon followed by two forward slashes) as a scheme separator. That separator implies that the data that follows conforms to an Internet standard

and can be interpreted as such. To prevent confusion, DirectPlay flags any URL containing"://" as invalid.

This section discusses

- Data Strings
- Data Values
- Data Value Summary
- Sample URLs

Data Strings

The data string holds address information. The first part of a data string consists of a series of *keyname*=*value* elements separated by semicolons (;). You can include optional user data by putting a number sign (#) after the last value, followed by an application-defined string.

The key name is a lowercase string that identifies the data and implicitly indicates what type of data is contained in the value. For instance, the "provider" key name indicates that the value contains a Microsoft® DirectPlay® service provider GUID, in the form of a GUID string. The following characters are reserved and should not be used in value strings.

Ampersand (&)	Forward slash (/)
At sign (@)	Number sign (#)
Colon (:)	Question mark (?)
Equal sign (=)	Semicolon (;)

The first element in the data string must be the provider. Other elements can follow in any order. A generic URL looks something like this.

x-directplay:/provider=%*provider GUID%;*[*keyname1=value1*];[*keyname2=value2*][...]#[*user defined string*]

Data Values

The values that need to be included in the data string depend on the particular service provider. Modem providers, for instance, need a telephone number in their address, whereas LAN providers might need a port number. This section provides a detailed description of the standard data values. It also includes a key name that can be used in place of the literal string. These names are defined in Dpaddr.h.

- Application Instance
- Baud
- Device
- Flow Control
- Host Name
- Parity

- Phone Number
- Port
- Program
- Provider
- Stop Bits

Application Instance

An optional GUID that identifies an application instance. This value is used when specifying the game that is to be connected to.

Key Name: DPNA_KEY_APPLICATION_INSTANCE

Key String: "applicationinstance"

Data Type: GUID

Providers: All

Valid Values: Any valid application instance GUID

Baud

The baud rate

Key Name: DPNA_KEY_BAUD

Key String: "baud"

Data Type: DWORD

Providers: Modem and serial

Valid Values: Any valid baud rate. You can set this value to the appropriate integer, or you can use one of the following predefined values from Dpaddr.h.

DPNA_BAUD_RATE_9600
DPNA_BAUD_RATE_14400
DPNA_BAUD_RATE_19200
DPNA_BAUD_RATE_38400
DPNA_BAUD_RATE_56000

Device

A GUID that identifies the device on the local computer that will be used. If the service provider supports all adapters, you do not need to specify a device.

Key Name: DPNA_KEY_DEVICE

Key String: "device"

Data Type: GUID

Providers: All, but for device addresses only, not host addresses

Valid Values: Any valid device GUID.

Flow Control

The type of flow control to be used

Key Name: DPNA_KEY_FLOWCONTROL

Key String: "flowcontrol"

Data Type: String

Providers: Serial and modem

Valid Values: Any of the following predefined values from Dpaddr.h.

DPNA_FLOW_CONTROL_NONE DPNA_FLOW_CONTROL_DTR
DPNA_FLOW_CONTROL_XONXOFF DPNA_FLOW_CONTROL_RTSDTR
DPNA_FLOW_CONTROL_RTS

Host Name

The name of a remote host computer

Key Name: DPNA_KEY_HOSTNAME

Key String: "hostname"

Data Type: String

Providers: All, but for host addresses only, not device addresses

Valid Values: A fully-qualified host name, or a dotted address.

Parity

The parity of the connection

Key Name: DPNA_KEY_PARITY

Key String: "parity"

Data Type: String

Providers: Serial and modem

Valid Values: Any of the following predefined values from Dpaddr.h.

DPNA_PARITY_NONE DPNA_PARITY_MARK
DPNA_PARITY_EVEN DPNA_PARITY_SPACE
DPNA_PARITY_ODD

Phone Number

A phone number

Key Name: DPNA_KEY_PHONENUMBER

Key String: "phonenumber"

Data Type: String

Providers: Modem

Valid Values: Any valid phone number

Port

An optional port number

Key Name: DPNA_KEY_PORT

Key String: "port"

Data Type: DWORD

Providers: IP and IPX

Valid Values: Any 16-bit integer. Only the lower 16 bits of the value are valid. If you do not specify a port, DirectPlay will choose one for you.

Program

An optional application GUID

Key Name: DPNA_KEY_PROGRAM

Key String: "program"

Data Type: GUID

Providers: All

Valid Values: Any valid application GUID

Provider

A GUID that identifies the Microsoft® DirectPlay® service provider that will be used

Key Name: DPNA_KEY_PROVIDER

Key String: "provider"

Data Type: GUID

Providers: All.

Valid Values: Any valid service provider GUID

Stop Bits

The number of stop bits

Key Name: DPNA_KEY_STOPBITS

Key String: "stopbits"

Data Type: String

Providers: Serial and modem

Valid Values: Any of the following predefined values from Dpaddr.h.

DPNA_STOP_BITS_ONE DPNA_STOP_BITS_TWO
DPNA_STOP_BITS_ONE_FIVE

Data Value Summary

The following two tables outline the standard data values, and they indicate which values are used by each type service provider for both host and device addresses.

Host Addresses

	IP	IPX	Serial	Modem
Application Instance	Optional	Optional	Optional	Optional
Baud	Not used	Not used	Required	
Device	Not used	Optional	Required	Required
Flow Control	Not used	Not used	Required	
Host Name	Required	Required	Optional	Optional
Parity	Not used	Not used	Required	
Phone Number	Not used	Not used	Not used	Required
Port	Required	Required	Not used	Not used
Program	Optional	Optional	Optional	Optional
Provider	Required	Required	Required	Required
Stop Bits	Not used	Not used	Required	

Device Addresses

	IP	IPX	Serial	Modem
Application Instance	Optional	Optional	Optional	Optional
Baud	Not used	Not used	Required	Not used
Flow Control	Not used	Not used	Required	Not used
Host Name	Optional	Optional	Optional	Optional
Device	Optional	Required	Required	Required
Parity	Not used	Not used	Required	Not used
Phone Number	Not used	Not used	Not used	Not used
Port	Optional	Required	Not used	Not used
Program	Optional	Optional	Optional	Optional
Provider	Required	Required	Required	Required
Stop Bits	Not used	Not used	Required	Not used

Sample URLs

The following sample URLs illustrate what a Microsoft® DirectPlay® URL might look like for the four standard service providers.

Local IP Address

x-directplay:/provider=%7BEBFE7BA0-628D-11D2-AE0F-006097B01411%7D;device=%7BIP ADAPTER GUID%7D;port=0000230034#IPUserData

Local IPX Address

x-directplay:/provider=%7B53934290-628D-11D2-AE0F-006097B01411%7D;device=%7BIPX ADAPTER GUID%7D;port=00230#IPXUserData

Local Serial Address

x-directplay:/provider=%7B743B5D60-628D-11D2-AE0F-006097B01411%7D;device=%7BCOM PORT GUID%7D;baud=57600;stopbits=1;parity=NONE;flowcontrol=RTSDTR#SerialUserData

Remote Modem Address

x-directplay:/provider=%7B6D4A3650-628D-11D2-AE0F-006097B01411%7D;device=%7BMODEM DEVICE GUID%7D;phonenumber=555-1212#ModemUserData

Handling Addresses

If you call the **Host**, **EnumHosts**, or **Connect** methods exposed by the **IDirectPlay8Peer**, **IDirectPlay8Client**, or **IDirectPlay8Server** you must pass address objects as parameters. If Microsoft® DirectPlay® does not have sufficient address information, the method that you called will fail, and it will return DPNERR_ADDRESSING. However, it is not necessary to have all the information in the address object at the time you call the method.

All address objects must have the service provider GUID set. However, it is possible to omit other data values.

- You can omit the device if the service provider supports all adapters.
- You can omit the port number for IP and IPX service providers for the **Host**, **EnumHosts**, and **Connect** methods. DirectPlay will assign a port number. This number may vary.
- If you set the OKTOQUERYFORADDRESSING flag, the service provider can display a dialog box asking the user for the information needed to complete the address. If the user does not supply sufficient information, the method will fail. If the OKTOQUERYFORADDRESSING flag is not set, no dialog box will be displayed. If the address you pass to the method is insufficient, the method will fail. In the last two cases, the error value that is returned will be DPNERR_ADDRESSING.

There are two important issues for IP and IPX service providers that you need to be aware of. Failing to handle them properly may cause your application to fail.

- If you set the NOBROADCASTFALLBACK flag when you call an enumeration method, you must supply a hostname. If you do not do so, the method will fail and return DPNERR_ADDRESSING.
- If you do not specify a port, do not assume that DirectPlay will always choose the same port number. The only way to be certain of the port number is to specify it in your address. If you do not specify a port number, you must retrieve the actual value later, after the command is in progress.

DirectPlay Address Objects

[C++]
Microsoft® DirectPlay® does not handle URL strings directly. Instead, the string must be encapsulated in a DirectPlay address object (CLSID_DirectPlayAddress). This object exposes the **IDirectPlay8Address** interface that enables you to insert URL information into, or extract it from, the address object.

To create DirectPlay address, you must call **CoCreateInstance** to create a DirectPlay address object. You can then define the address in one of two ways:

- Create the URL string directly. Then use either **IDirectPlay8Address::BuildFromURLA** or **IDirectPlay8Address::BuildFromURLW** to insert the complete string.
- Use **IDirectPlay8Address** methods to insert the various pieces of data that make up the string directly into the object.

When you receive an address object, you have a similar pair of options.

- Extract the entire URL string with either **IDirectPlay8Address::GetURLA** or **IDirectPlay8Address::GetURLW**. Then parse the string and extract the needed information
- Use other **IDirectPlay8Address** methods to extract the particular data you are interested in from the address object.

[Visual Basic]
Microsoft® DirectPlay® does not handle URL strings directly. Instead, the string must be encapsulated in a **DirectPlay8Address** object. This object exposes a number of methods that enable you to insert URL information into, or extract it from, the object.

To create DirectPlay address, you must first create a **DirectPlay8Address** object. You can then define the address in one of two ways:

- Create the URL string directly. Then use **DirectPlay8Address.BuildFromURL** to insert the complete string.
- Use other **DirectPlay8Address** methods to insert the various pieces of data that make up the string directly into the object.

When you receive an address object, you have a similar pair of options.

- Extract the entire URL string with **DirectPlay8Address.GetURL**. Then parse the string and extract the needed information
- Use other **DirectPlay8Address** methods to extract the particular data you are interested in from the address object.

DirectPlay Protocol

Multiplayer games require efficient and flexible network messaging services for optimal performance. The Microsoft® DirectPlay® protocol is a transport-layer messaging protocol that is used for all DirectPlay messaging. The protocol has been substantially reworked for DirectPlay 8.0. It provides your application with the messaging support it needs to make everything run smoothly. The DirectPlay protocol includes the following messaging support.

- Reliable and unreliable delivery of messages
- Sequential and non-sequential delivery of messages
- Message fragmentation and reassembly
- Congestion control
- Send prioritization
- Message timeouts

Note

With previous versions of DirectPlay, the DirectPlay protocol was optional, and had to be specified explicitly. With DirectPlay 8.0, this protocol is used for all DirectPlay messaging.

This document provides a general description of how the DirectX protocol works, and how you can use it in your application.

- Basic Message Handling
- Message Categories
- Congestion Control
- Send Prioritization
- Monitoring Messaging Statistics
- Monitoring the Pending Message Queues

Basic Message Handling

A message, as the term is used in this document, is a block of data that needs to be sent to another computer. A network protocol creates a *packet* by adding some bits to the data block that hold information such as the target's network address. This packet is the basic unit of network data. When the target receives the packet, the target's network protocol removes the extra bits and passes the data block to the receiving application.

Although similar in usage, the terms *message* and *packet* are not strictly interchangeable. This document uses the term *message* to refer to the unit of information that is passed to and received from the Microsoft® DirectPlay® API. *Packet* refers to the unit of information handled by the network. DirectPlay handles packets internally. With rare exceptions, DirectPlay applications need to deal only with messages.

The primary reason for the distinction between *message* and *packet* is that networks generally limit the maximum size of the packets they handle. This size is referred to as a Maximum Transmission Unit (MTR). If a message is small, it is sent in a single packet and the two terms are effectively synonymous. However, large messages might need to be fragmented into two or more packets and then be reassembled by the receiver. The DirectPlay protocol automatically handles fragmentation and reassembly of messages as needed. As far as your application is concerned, you send a message, and the target receives it.

[C++]

Note

DirectPlay delivers messages of any size. However, the more packets that are required for a single message, the greater the odds that one or more packets will be lost and have to be retransmitted. Messages that are large enough to require fragmentation and reassembly thus typically have more network latency than single-packet messages. If you need to keep network latency to a minimum, avoid sending large messages, especially on lossy networks. You can determine the maximum size that your connection can accommodate in a single packet by calling the **GetSPCaps** method exposed by the **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server** interfaces.

[Visual Basic]

Note

DirectPlay delivers messages of any size. However, the more packets that are required for a single message, the greater the odds that one or more packets will be lost and have to be retransmitted. Messages that are large enough to require fragmentation and reassembly thus typically have more network latency than single-packet messages. If you need to keep network latency to a minimum, avoid sending large messages, especially on lossy networks. You can determine the maximum size that your connection can accommodate in a single packet by calling the **GetSPCaps** method exposed by the **DirectPlay8Peer**, **DirectPlay8Client**, and **DirectPlay8Server** interfaces.

Message Categories

The Microsoft® DirectPlay® protocol is designed to handle the following two basic types of network messaging.

- *Reliable* versus *unreliable* messaging determines whether messages are guaranteed to be delivered to the target application.
- *Non-sequential* versus *sequential* messaging determines whether messages are received by the target application in the same order they are sent.

Games use messaging for a variety of purposes, each with different demands. To support this range of messaging needs, the DirectPlay protocol enables you to designate a message as belonging to one of four categories:

- Reliable and sequential
- Unreliable and sequential
- Reliable and non-sequential
- Unreliable and non-sequential

The DirectPlay protocol enables you to optimize your messaging strategy by assigning categories on a message-by-message basis.

Reliable and Unreliable Messaging

Messages are sometimes lost in transit. *Reliable* messaging provides a guarantee that the target will receive every message. This type of messaging is required when data loss cannot be tolerated. Most reliable messaging schemes require the target to acknowledge receipt of each message. If the sender does not receive an acknowledgment within a specified timeout period, it resends the message. This process typically continues until the sender receives an acknowledgment, confirming that the message has arrived.

The DirectPlay protocol imposes a limit on the number of resend attempts. If no acknowledgment is received after a reasonable number of attempts, DirectPlay assumes that the connection has been lost, and closes it.

Unreliable messaging is the simplest form of network communication. It might be faster than reliable messaging because there is no guarantee that the message will be delivered to the target. The sender transmits the message. If the target does not receive the message, the sender will not transmit the message again, and the packet is lost.

Unreliable messaging is used primarily when speed or bandwidth is more important than an occasional lost message. For example, high-bandwidth streaming media applications often use unreliable messaging. They cannot afford to take up bandwidth with acknowledgments and retransmissions, nor can they wait for a lost message to be retransmitted. An occasional lost message normally has only a minor impact on quality, so it can be ignored.

Sequential and Non-Sequential Messaging

Messages leave the sender in a particular sequence. However, there is no guarantee that messages will arrive at the target's computer in the same order that they are sent. For example, if a message is lost and must be retransmitted, that message will typically arrive later than messages that followed it in the original sequence.

Sequential messaging uses sequencing information embedded in the message to ensure that the messages are presented to the target application in the correct order. This type of messaging is required when the target application must receive messages in the correct order. Out-of-order messages are buffered until the missing messages arrive.

Non-sequential messaging presents the received messages to the target as soon as they arrive at the target computer, regardless of the order in which they were sent. Because there is no need to wait for a missing packet, applications often use non-sequential messaging when speed is more important than an occasional out-of-order message. The out-of-order message is ignored.

Choosing the Best Message Category

Choosing the best category for messages is a core issue for multiplayer game developers. While DirectPlay provides the tools to manage your messaging, the choice of a message category ultimately depends on the semantic content of the message and the nature of the game.

The following are general guidelines for choosing the best message category.

- Use non-guaranteed messaging whenever the content permits. For example, your game might send frequent player-location updates. Each update is independent, and it supersedes any previous updates. If an update is lost, the next update is sufficient to maintain the player's game state. A lost and retransmitted message might arrive later than the subsequent update message.
- Use guaranteed messaging when data loss cannot be tolerated. For example, a text-based chat feature depends on every character being delivered to its target.
- Use sequential messaging when the order of the messages is important. For example, streaming media typically uses sequential-unreliable messaging. An occasional dropped message can be tolerated, but an out-of-order message would cause problems.

Congestion Control

In an ideal world, your game can send messages as often as it needs to. They arrive at the target immediately and are processed instantaneously. If all of the computers in your game have ample processing power and are connected by a lightly used high-bandwidth network link, you might approach this ideal situation. You can then send messages as often as you like. However, a number of factors can create *congestion* and cause messaging to work more slowly than this ideal:

- Network latency. Even under ideal conditions, messages take a finite time to traverse the network from sender to target, especially over the Internet. There might be further delays for acknowledgments, retransmission of lost packets, or reassembly of out-of-order packets.
- Network bandwidth. The network bandwidth controls the rate at which a message can be sent or received by a computer. Network links have a wide range of bandwidths, and even high-bandwidth networks might be slowed by high traffic levels. If one or more of your players has a low-bandwidth connection, they will be able to send and receive messages only at a limited rate.
- Processing speed. Even if network bandwidth is high and latency low, the target application still needs some time to process a received message. If one or more of the players in a session is using a relatively slow computer, the rate at which they can process received messages might be below the rate at which messages can be sent.

Message Throttling

If there is no control over the rate at which messages are sent, a target can be flooded by more messages than it can handle. To prevent this situation, the Microsoft® DirectPlay® protocol *throttles* the rate at which messages are sent. The net effect of throttling is that the rate at which messages are sent is controlled by the rate at which the target can handle them.

Throttling is implemented with a *sliding window* mechanism. The sliding window is basically a queue with a limited number of slots that holds messages that have been sent but not yet received. All outgoing messages are placed in this queue, regardless of their category. Once the sent-message queue is full, it accepts no more outgoing messages until one of the messages in the queue has been received.

For optimal performance, the size of the sliding window must be matched to current network conditions. The DirectPlay protocol automatically monitors such factors as the number of messages and the total number of bytes in the sent-message queue. This information is then used to dynamically adjust the size of the sliding window to optimize messaging for the current network conditions.

Connection Checking

If there is no activity on a link, the DirectPlay protocol periodically tests the connection by sending an empty reliable packet. If no acknowledgment is received from the target after a reasonable number of attempts, DirectPlay concludes that the link has been disconnected.

Send Prioritization

Messages often vary widely in importance. Some are time-critical, and must be delivered as quickly as possible. Others can be delayed if necessary, or possibly not sent at all. One issue with congestion control algorithms is that an application might create messages faster than they can be sent. Unsent messages must then be held in a queue until an outgoing slot opens up. If all unsent messages are held in a single

pending-message queue, high priority messages might be blocked while waiting for lower priority messages to be sent first.

The Microsoft® DirectPlay® protocol solves this problem by having three pending message queues: low, medium, and high priority. When a slot opens up in the sent-message queue, the protocol selects the next message to be sent as follows:

1. Send the oldest message in the high-priority queue.
2. If there are no messages in the high-priority queue, send the oldest message in the medium-priority queue.
3. If there are no messages in the medium-priority queue, send the oldest message in the low-priority queue.

This priority mechanism enables you to get your time-critical messages out as quickly as possible, even though other less important messages have already been submitted.

Send Timeouts

One of the consequences of throttling is that messages might spend a relatively long amount of time in a pending-message queue, especially if they are low priority. Some messages might stay long enough to have been superseded by subsequent messages. These messages are no longer relevant. For example, your application might periodically send player-location update messages. Each update is independent of the others, and supersedes any previous updates. If you have two player-location updates in the pending message queue, only the most recent one needs to be sent.

The DirectPlay protocol enables you to handle the problem of outdated messages by adding an optional timeout value to the message. If the message is still in a pending-message queue when the timeout expires, the message will be canceled.

Disconnection

When an application sends a disconnect message, the message is placed at the end of the low-priority pending-message queue, and the protocol stops accepting outgoing messages. This practice guarantees that all pending messages are sent before the link is disconnected. The disconnect message is sent as a reliable sequential message to guarantee that it arrives, but not before all other messages in the queue have been delivered.

Monitoring Messaging Statistics

While the Microsoft® DirectPlay® protocol handles many aspects of messaging automatically, your application should still monitor messaging behavior. For example, if you are consistently sending messages faster than they can be delivered, you might need to modify your messaging scheme.

[C++]
Because network conditions change continuously, your application should periodically check the behavior of the network and adjust its messaging scheme accordingly. To do so, call the **GetConnectionInfo** method that is exposed by every

DirectPlay 8.0 interface that supports messaging. **GetConnectionInfo** returns a structure that contains a wide variety of statistical information that you can use to refine your messaging scheme, including the following:

[Visual Basic]
Because network conditions change continuously, your application should periodically check the behavior of the network and adjust its messaging scheme accordingly. To do so, call the **GetConnectionInfo** method that is exposed by every DirectPlay 8.0 object that supports messaging. **GetConnectionInfo** returns a type that contains a wide variety of statistical information that you can use to refine your messaging scheme, including the following:

- Round trip latency
- Throughput
- Packets sent
- Packets received
- Packets resent
- Packets dropped
- Messages transmitted at different priority levels

Note
The messaging statistics are obtained by monitoring the actual network traffic. If you call **GetConnectionInfo** immediately after you initialize the connection, there will have been little time to collect data and the statistics might be misleading.

Monitoring the Pending Message Queues

[C++]
You should monitor your pending message queues to ensure that they do not become too large. The **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server** interfaces all expose a **GetSendQueueInfo** method that can be used to check the number of messages and the number of bytes currently in the queue. By default, the method returns the total for all three queues, but you can also obtain values for each of the three priority levels.

[Visual Basic]
You should monitor your pending message queues to ensure that they do not become too large. The **DirectPlay8Peer**, **DirectPlay8Client**, and **DirectPlay8Server** objects all expose a **GetSendQueueInfo** method that can be used to check the number of messages and the number of bytes currently in the queue. By default, the method

returns the total for all three queues, but you can also obtain values for each of the three priority levels.

Optimizing Network Usage

Providing the best gaming experience normally means sending updates and other information as rapidly as possible without flooding the target with more messages than it can handle. The Microsoft® DirectPlay® protocol combined with asynchronous messaging enables you to dynamically optimize your messaging strategy to provide your users with the best possible game experience.

[C++]
The bulk of your messaging will use the **IDirectPlay8Peer::SendTo**, **IDirectPlay8Client::Send**, or **IDirectPlay8Server::SendTo** methods. These methods normally work asynchronously for all message categories. They return immediately, and your message handler receives a **DPNMSG_SEND_COMPLETE** message when the message is actually sent. You can choose to send messages synchronously by setting the DPNSEND_SYNC flag. If you do so, the method will block until the message is actually sent.

[Visual Basic]
The bulk of your messaging will use the **DirectPlay8Peer.SendTo**, **DirectPlay8Client.Send**, or **DirectPlay8Server.SendTo** methods. These methods normally work asynchronously for all message categories. They return immediately, and DirectPlay calls your message handler's **DirectPlay8Event.SendComplete** method when the message is actually sent.

You can choose to send messages synchronously by setting the DPNSEND_SYNC flag. If you do so, the **Send/SendTo** method will block until the message is actually sent.

The DirectPlay protocol's throttling mechanism guarantees that the client will not receive messages faster than they can be handled. However, the throttling protocol does not control how frequently you submit messages to the outgoing queue. You can easily end up with a large backlog of messages in your unsent message queues. You can avoid this situation by sending messages as infrequently as possible, but then you might unnecessarily degrade the user's experience. An optimal messaging strategy sends messages as fast as possible without exceeding the target's ability to handle them.

The following are tips for optimizing your messaging strategy.

- Send most if not all of your messages asynchronously. If you send a message synchronously, the method will block until the throttling mechanism allows the message to be sent.
- Monitor the pending message queues and the network statistics. If there are few or no messages in the queue, you can increase your transmission rate. If the queues are large or growing rapidly, decrease your transmission rate and perhaps cancel some messages. See the discussion of send timeouts in Send Prioritization for further discussion.
- Analyze the pending message queues on a player-by-player basis. Some players might be able to receive messages at a much higher rate than others. The bulk statistics might be misleading. Consider using directed sends rather than group sends.
- Choose the appropriate category for each message. Reserve the categories with the highest overhead for the most important messages.
- Prioritize your messages, so that the most important are assured of being sent promptly and not delayed by relatively unimportant messages.
- Do not let the pending message queues grow too large. In addition to delaying the transmission of your messages, a large number of pending messages might consume significant memory resources.
- Use the timeout feature of the **Send** and **SendTo** methods to automatically clear outdated messages from the pending message queue.
- Minimize the amount of data per message. It is usually better to send frequent small messages than a smaller number of large messages.
- Do not loop tightly when checking the pending message queue. Doing so wastes CPU cycles. Instead, use a sleep period based on how long it typically takes the queue to get down to the level that it will be ready for another send.

Using the DirectX Protocol in an Application

This section covers how you can use the features of the Microsoft® DirectPlay® protocol in your application.

[C++]
You can use five DirectPlay interfaces to send messages.

- **IDirectPlay8Peer**
- **IDirectPlay8Client**
- **IDirectPlay8Server**
- **IDirectPlay8LobbyClient**
- **IDirectPlay8LobbiedApplication**

Depending on which interface your application is using to send messages, you send a message by calling a method named either **Send**, or **SendTo**. While the usage of

these five methods is similar, they vary in detail. Refer to the appropriate reference pages for further discussion.

[Visual Basic]
You can use five DirectPlay objects to send messages.

- **DirectPlay8Peer**
- **DirectPlay8Client**
- **DirectPlay8Server**
- **DirectPlay8LobbyClient**
- **DirectPlay8LobbiedApplication**

Depending on which object your application is using to send messages, you send a message by calling a method named either **Send**, or **SendTo**. While the usage of these five methods is similar, they vary in detail. Refer to the appropriate reference pages for further discussion.

[C++]
The **Send/Sendto** method's parameters might allow you to control many of the DirectPlay protocol's features. For example, the *dwFlags* field of **IDirectPlay8Peer::SendTo** allows you to specify:

- The message's priority level.
- Whether the message is reliable or unreliable.
- Whether the message is sequential or non-sequential.

Refer to the appropriate method reference for further details.

When your application receives a message, your callback function will receive a DPN_RECEIVE message. The associated structure contains a pointer to the data block, along with information such as the source of the message.

[Visual Basic]
The **Send/Sendto** method's parameters might allow you to control many of the DirectPlay protocol's features. For example, the *lFlags* field of **DirectPlay8Peer.SendTo** allows you to specify:

- The message's priority level.
- Whether the message is reliable or unreliable.
- Whether the message is sequential or non-sequential.

Refer to the appropriate method reference for further details.

When your application receives a message, DirectPlay will call your message handler's **DirectPlay8Event.Receive** method. The associated type contains a pointer to the data block, along with information such as the source of the message.

DirectPlay Callback Functions and Multithreading Issues

Microsoft® DirectPlay® and DirectPlay Voice both require you to implement and register several callback functions to handle the events fired by DirectPlay. DirectPlay is multithreaded and will fire multiple events concurrently. It is possible that your application will receive multiple overlapping callbacks.

DirectPlay maintains a thread pool to service callback indications, and your callback is invoked on a thread from the pool of threads maintained by DirectPlay. The size of this thread pool is configurable on a per process basis in Microsoft Windows 2000. Also, DirectPlay will use I/O completion ports when running on Windows 2000. I/O completion ports is an advanced topic beyond the scope of this document, and it is recommended that you look in the Microsoft Developer Network documents or in one of the Win32 multithreading references currently available.

In order to correctly and reliably access data in DirectPlay callbacks, you are required to implement a method of multithreading synchronization. This is known as making your callback *re-entrant* or *threadsafe*.

The Microsoft Windows family of operating systems currently offers three methods of synchronizing data in multithreaded environments:

- Mutex Objects
- Semaphore Objects
- Critical Section Objects

The DirectPlay voice samples that ship with the DirectX 8.0 SDK demonstrate synchronization using Critical Section Objects. If you wish to implement a Mutex or Semaphore Object, these topics are discussed in the Microsoft Platform SDK as well as in many reference books. Implementing any of these synchronization methods requires an expert knowledge level in these areas due of the level of complexity and difficulty in debugging should any issues arise.

The DirectPlay threading model is optimized for maximum efficiency and there are no thread context switches during "send" operations and during "indication" messages, including receive messages.

This section discusses DirectPlay Networking Callbacks.

See Implementing a Callback Function in DirectPlay and DirectPlay Voice for more information.

DirectPlay Networking Callbacks

[Visual Basic]
This topic pertains only to applications written in C++.

[C++]
Microsoft® DirectPlay® networking callback functions are of type PFNDPNMESSAGEHANDLER. Depending on the type of networking session, you register the address of your callback function with IDirectPlay8Peer::Initialize, IDirectPlay8Client::Initialize, or IDirectPlay8Server::Initialize.

Synchronization Issues

You must employ one of the three thread synchronization objects in order to maintain the integrity of your game data during processing in a DirectPlay callback.

In order to understand how your game data could be corrupted, consider that your callback inserts a packet of game data into a structure. If another thread enters the callback reentrantly before the first callback has completed, it is possible that this thread could also attempt to access the structure at the same location in memory and change the data. Therefore, the data placed in the structure by the first thread is overwritten by the data placed in the structure by the second thread. Please note that this is an oversimplified example of multithreading and there are many other implications to not properly synchronizing multiple threads. Again, it is advisable to achieve an expert level of knowledge in implementing multithreaded callbacks before you attempt to create your own.

See Implementing a DirectPlay Networking Callback Using Critical Section Objects for an example of how to synchronize data in a DirectPlay networking session.

Worker Threads

You have the option of creating your own "worker threads". A worker thread is another multithreaded application defined callback that is created to process game data independently of the DirectPlay callbacks. The most common way of accomplishing this is to buffer data received during a DirectPlay networking callback thread. Then, a new thread is created and a message is sent to your worker thread callback to notify it to process the buffered data.

Multithreading Performance Issues and Asynchronous Operations

It is important to carefully consider how much time is spent processing messages in DirectPlay callbacks. If you process a lot of data within the DirectPlay callbacks and you employ a data locking mechanism to synchronize threads, you will run into blocking problems as other threads wait to enter the callback.

If you choose to implement a worker thread and offset the processing of game data to another callback, you run the risk of adding a lot of overhead processing time as the CPU switches context between the threads you create and the threads created by DirectPlay. This should be done only if the game data requires a large amount of processing time, and the data is not critical to the real time operation of the game. For example, it is not recommended to process player location data in a worker thread because this data is critical to positioning players in real time within the game.

You can also return **DPNSUCCESS_PENDING** from the callback, create a pointer to the data buffer, and make that pointer available the worker thread. When the worker thread is finished processing the game data, it calls the ReturnBuffer method of either IDirectPlay8Peer or IDirectPlay8Client, depending on the topology used.

Understanding DirectPlay Voice

Microsoft® DirectPlay® Voice is a full-voice communications API that is integrated with DirectPlay for network session management and network transport.

DirectPlay Voice is also integrated with DirectPlay Sound for voice recording and playback, and all DirectPlay Sound audio features are inherited including the ability to target voice data to different playback buffers and the use of special audio effects such as three-dimensional sound positioning.

DirectPlay Voice Networking

DirectPlay Voice Topologies

Voice Host Migration

Audio Device Testing

Voice Codecs

Automatic Gain Control

Transmission Control

Capture Focus

Jitter Buffers

Working Set Guidelines

DirectPlay Voice Networking

Microsoft® DirectPlay® Voice uses a DirectPlay session for media-independent network transport and player management. The DirectPlay Voice API does not duplicate session control features from DirectPlay. A DirectPlay network transport session must also be created before DirectPlay Voice can transmit and receive voice

communications. DirectPlay Voice can use either the IDirectPlay4 object or IDirectPlay8 object for network transport.

Note that if DirectPlay Voice is being used in-process with a multiplayer game, the game will most likely also use the transport session to exchange its game-specific data. This makes it possible to optimize the use of network resources between the game and voice data.

It is also acceptable to create and use a transport session specifically for the voice session, as would be the case for a standalone voice conferencing application.

DirectPlay Voice Topologies

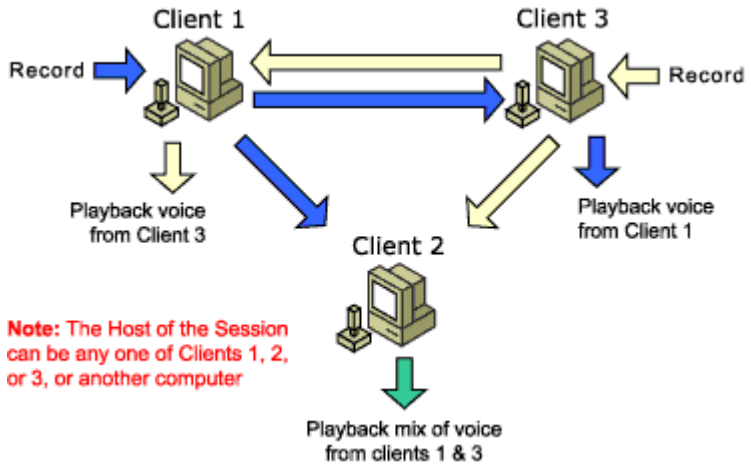
Microsoft® DirectPlay® Voice sessions require a DirectPlay network session to transport voice communication. Once the network session has been created, a DirectPlay Voice object can be created to use one of three topologies.

- Peer-to-Peer Voice Topology
- Forwarding Server Voice Topology
- Mixing Server Sessions

The choice of topology is dependent on several factors, and these factors are discussed in the individual DirectPlay Voice topology topics. Note that not all voice topologies can be transported over all types of DirectPlay networking sessions.

Peer-to-Peer Voice Topology

In a Microsoft® DirectPlay® Voice session using a peer-to-peer topology, each voice-session client streams its voice audio data directly to every other voice-session client. Each client receives all individual incoming voice audio streams, mixes the received streams, and plays the resulting mixed signal on the client's computer.



The advantage of using a peer-to-peer topology is that no computer in the voice session requires high bandwidth or processor power. However, the bandwidth and

processor usage on each client's computer varies according to the number of incoming and outgoing audio streams. The number of outgoing voice audio streams is equal to the number of targets participating in the voice session, unless the network provider is capable of true multicasting, as noted below. The number of incoming voice audio streams depends on how many other voice-session clients are targeting the client in question and also on how many of the other clients are speaking.

As a game design consideration, it is not useful for a voice-session client to be the target of more than about six to eight other clients. If all six to eight clients are speaking at once, the conversation can become confusing, and communication between clients can be difficult.

If the DirectPlay network session supports true multicasting, the number of outgoing voice audio streams can be reduced considerably. If all clients are part of a multicast network and the target of the voice stream is a DirectPlay group, there is only one outgoing stream.

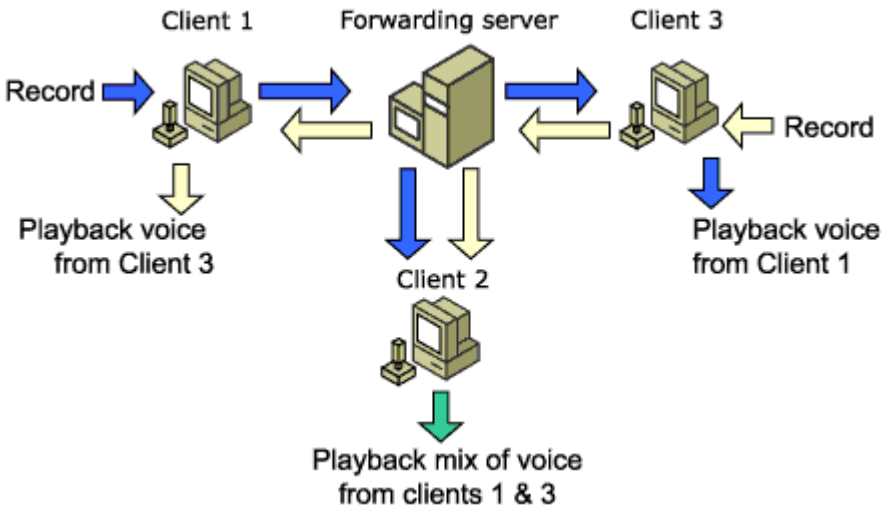
[C++]
A DirectPlay voice session using a peer-to-peer voice topology supports 3-D spatialization of the voice data using the **IDirectPlayVoiceClient::Create3DSoundBuffer** method.

[Visual Basic]
A DirectPlay voice session using a peer-to-peer voice topology supports 3-D spatialization of the voice data using the **DirectPlay8VoiceClient.Create3DSoundBuffer** method.

It is important to note that a voice session using a peer-to-peer voice topology cannot be used if the network transport is a client/server session.

Forwarding Server Voice Topology

In a Microsoft® DirectPlay® voice session using a forwarding server topology, one computer in the session acts as a forwarding server. Each client in the voice session streams voice data to the forwarding server, which then forwards the voice data to all other clients in the session. Each client receives all incoming audio streams forwarded from the forwarding server. Each client's computer then mixes the incoming streams and plays them back.



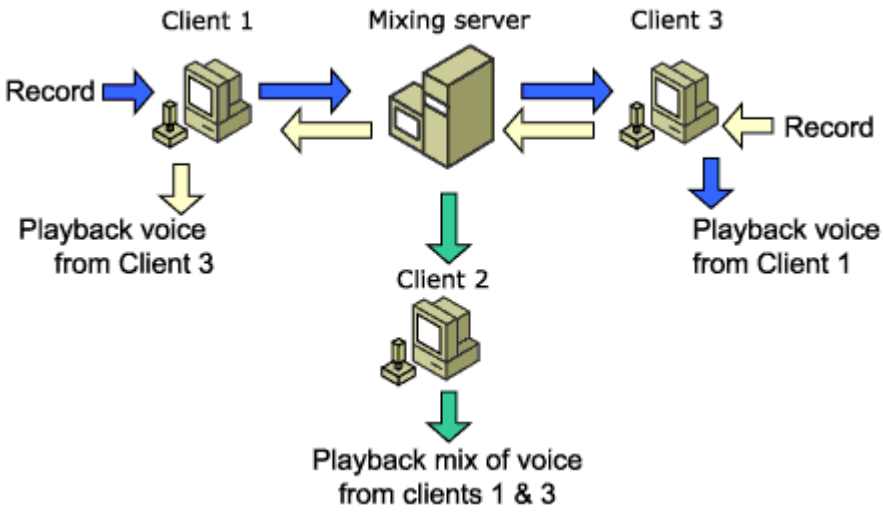
The outgoing bandwidth requirement on each client in a voice session using a forwarding server topology is constant because there is only one outgoing voice audio stream. The incoming bandwidth and processor requirements are identical to the requirements of a voice session using a peer-to-peer topology, but they vary depending on the number of incoming voice audio streams.

The server has much higher bandwidth requirements than the individual clients in a forwarding server DirectPlay voice session. However, the processor requirements are not high because no compression or decompression of voice data occurs on the server. This reduced load on the computer's processor also means that an individual client's computer with a high bandwidth connection can host the forwarding server without adversely affecting the performance of the individual client's computer or the performance of a game server and/or client program running on the same computer.

Note that in a voice session using a peer-to-peer topology, the outgoing bandwidth requirements on the individual clients are usually much higher than the incoming bandwidth requirements. Therefore, reducing the outgoing bandwidth requirement to a single stream of audio can result in a significant reduction in total bandwidth usage. For example, if a client is taking part in an eight-person voice session in which all clients can hear one another, the client has seven outgoing voice streams each time voice data is captured and transmitted on his or her computer. However, it is rare that all clients talk at once, so there are most likely fewer than two or three incoming voice streams at any one time.

Mixing Server Sessions

In mixing server sessions, one computer in the session acts as a mixing server. Each client streams its voice data to the mixing server. The mixing server examines the targets of each voice stream, performs decompression, mixing, and recompression as appropriate to generate a mixed stream of audio data for each client. Each client receives this single stream of pre-mixed audio data and plays it back.



The outgoing bandwidth, incoming bandwidth, and CPU requirement on the client in a mixing server session is easily predictable because each client has only one outgoing stream of audio to compress and send, and one incoming stream of audio to decompress and play back.

The mixing server has much higher bandwidth and CPU requirements than do the clients. Typically, the mixing server is either a completely dedicated computer, or it shares a computer with a dedicated game server.

[C++]
Mixing server voice sessions do not support 3-D spatialization of the voice data through the **IDirectPlayVoiceClient::Create3DSoundBuffer** method.

[Visual Basic]
Mixing server voice sessions do not support 3-D spatialization of the voice data through the DirectPlayVoiceClient8.Create3DSoundBuffer method.

You can run mixing server voice sessions using either a peer-to-peer or a client/server transport session.

Voice Host Migration

In a peer-to-peer Microsoft® DirectPlay® network session, one client of the networking session acts as host. If that host should exit the session or stop responding for any reason, another client in the session is elected as host.

[C++]

In a DirectPlay voice session, a similar process of host migration occurs in peer-to-peer voice sessions, except that the voice host migrates independently of the DirectPlay network session. The voice host migrates when the server calls **IDirectPlayVoiceServer::StopSession** or if the voice host stops responding. When the voice host migrates, each client in the voice session receives a DVMSG_HOSTMIGRATED callback. The structure passed to the new host has a valid **pdvServerInterface** pointer and can begin making **IDirectPlayVoiceServer** calls.

[Visual Basic]
In a DirectPlay voice session, a similar process of host migration occurs in peer-to-peer voice sessions, except that the voice host migrates independently of the DirectPlay network session. The voice host migrates when the server calls **DirectPlayVoiceServer8.StopSession** or if the voice host stops responding. When the voice host migrates, the **DirectPlayVoiceEvent8.HostMigrated** method is called on each of the clients. If the local client has become the new voice session host, the *NewServer* parameter will point to the newly created **DirectPlayVoiceServer8** object that can be used by the local client for providing host services. If the local client is not the new host, the NewServer parameter will be NULL.

Audio Device Testing

[C++]
Microphone setup is supported by the **IDirectPlayVoiceTest** interface. This interface has one method, **IDirectPlayVoiceTest::CheckAudioSetup**, that can be used to run the test wizard. This wizard confirms that your system properly supports full duplex operation and ensures your microphone and playback settings are correct. You need to run the wizard only once for each combination of playback and capture device you select. Each time your application starts, you should test the configuration by calling **IDirectPlayVoiceTest::CheckAudioSetup** with the *dwFlags* parameter set to DVFLAGS_QUERYONLY. This enables you to quickly test whether the device configuration has changed since you last tested them. If your devices have not been tested, you should run **IDirectPlayVoiceTest::CheckAudioSetup** again to invoke the wizard. If you do not do so, then **IDirectPlayVoiceClient::Connect** will return DVERR_RUNSETUP, and you will not be able to initialize DirectPlay Voice.

If the user's sound card does not have full duplex capability, it can only listen to voice communications. It cannot send voice communications because the game typically holds the audio card in playback mode. To prevent problems, DirectX 8.0 does not enable switching dynamically between playback and capturing. The DirectX audio setup wizard provides feedback to the user on the duplexing capabilities of the system.

[Visual Basic]
Microphone setup is supported by the **DirectPlayVoiceTest8.CheckAudioSetup** method.

Calling this method invokes the DirectX audio setup wizard, which runs tests and confirms that the system properly supports full duplex operation and ensures microphone and playback settings are correct. You need to run the wizard only once for each combination of playback and capture device you select. Each time your application starts, you should test the configuration by calling **DirectPlayVoiceTest8.CheckAudioSetup** with the *lFlags* parameter set to DVFLAGS_QUERYONLY. This enables you to quickly test whether the device configuration has changed since your devices were last tested. If your devices have not been tested, you should run **DirectPlayVoiceTest8.CheckAudioSetup** again to invoke the wizard. If the configuration has changed since the last test and you have not run the wizard again, **DirectPlayVoiceClient8.Connect** will return DVERR_RUNSETUP, and you will not be able to initialize DirectPlay Voice.

If the user's sound card does not have full duplex capability, it can only listen to voice communications. It cannot send voice communications because the game typically holds the audio card in playback mode. To prevent problems, DirectX 8.0 does not enable switching dynamically between playback and capturing. The DirectX audio setup wizard provides feedback to the user on the duplexing capabilities of the system.

Note that there are still many computer systems in active use that do not include a full duplex sound card. Full duplex sound cards came into popular use in 1998, although at the time few of the audio card drivers had full duplex operation enabled. Customers who purchased new systems in 1999 or upgraded drivers in 1999 are more likely to have full duplex capability.

Voice Codecs

The compression/decompression (codec) algorithms provided specifically with Microsoft® DirectPlay® are voice-quality codecs. They are all 8 kHz, 16-bit mono-format–based algorithms. Third-party codecs are not supported, and you cannot write proprietary codecs for use with DirectPlay Voice.

It is important to note that as the bandwidth requirements drop, the audio quality of the voice data also drops. The following table describes the sound quality of each codec.

Codecs cannot be dynamically switched during a game voice session, and all users must use the same codec in a voice session. However, a game could possibly create an appropriate user interface to force codec switching by dropping a current session and creating a new session using a different codec. This would be appropriate if the game application's users were jumping from a lobby chat to a game.

As with all other game setup parameters, the host should control the codec used. The voice-session host does not necessarily have to be the same as the game-data host.

As with any form of network communication, it is important to analyze the cost of the voice communication to ensure that adequate bandwidth is available to support communication of the game data and voice data. Analyzing the voice bandwidth consumption is straightforward. Estimate the number of simultaneous voice streams that you anticipate and multiply that number by the sum of the bandwidth required by the codec and the protocol overhead.

CPU consumption is another factor to consider when choosing a codec. Even low bandwidth codecs typically require about 8 percent of a 200 megahertz (MHz) Pentium processor's resources to encode a voice stream and an additional 4 percent to decode. As with network bandwidth, CPU resource consumption is additive per stream.

Automatic Gain Control

[C++]
Microsoft® DirectPlay® Voice offers functionality to adjust the hardware input volume on the sound card automatically to provide the best recording input level possible. To enable Automatic Gain Control, set the DVCLIENTCONFIG_AUTORECORDVOLUME flag in the *dwFlags* member of the **DVCLIENTCONFIG** structure when you set the client configuration. Automatic Gain Control can be activated or deactivated at any time during the voice session.

[Visual Basic]
Microsoft® DirectPlay® Voice offers functionality to adjust the hardware input volume on the sound card automatically to provide the best recording input level possible. To enable Automatic Gain Control, set the DVCLIENTCONFIG_AUTORECORDVOLUME flag in the *lFlags* member of the DVCLIENTCONFIG structure when you set the client configuration. Automatic Gain Control can be activated or deactivated at any time during the voice session.

Most game applications should use automatic gain control because it requires a negligible amount of game resources and prevents the need for an in-game volume recording control. Users are not required to set the level themselves, yet they experience the highest quality of voice transmission and reception possible.

Transmission Control

To keep the performance requirements of Microsoft® DirectPlay® Voice low, voice data should be transmitted only when the user is speaking. There are two methods to control voice data transmission. The choice of which method to use depends on your game's design considerations.

- Voice Activation
- Push to Talk

Voice Activation

[C++]
With voice activated transmission control, the microphone input is constantly analyzed to determine if the user is speaking. Voice activation has two benefits. It does not require the user to do anything more than speak into the microphone. Also, it is easily coded because it requires only setting the *dwFlags* parameter of the DVCLIENTCONFIG structure to DVCLIENTCONFIG_AUTOVOICEACTIVATED when the voice session is connected.

[Visual Basic]
With voice-activated transmission control, the microphone input is constantly analyzed to determine if the user is speaking. Voice activation has two benefits. It does not require the user to do anything more than speak into the microphone. Also, it is easily coded because it requires only setting the *lFlags* parameter of the DVCLIENTCONFIG structure to DVCLIENTCONFIG_AUTOVOICEACTIVATED when the voice session is connected.

However, one drawback of voice activation is that sounds such as the user breathing directly on the microphone, high levels of ambient sound caused by a noisy environment, or set of external speakers playing back the game's audio could cause unwanted voice activation. In addition, low-quality microphones exaggerate this possibility.

Push to Talk

Push-to-talk transmission control requires users to actively select when they want to transmit voice data. With this transmission control method, there is no danger that anything besides voice data will be transmitted. This method is analogous to pushing the Talk button on a two-way radio, and this functionality adds reality to certain game genres such as first-person shooters. Another benefit of using this method is that requiring users to actively select when they want to speak reduces the number of users speaking at once. This transmission control requires more design and development than voice activation because user-control functionality is required.

Capture Focus

[C++]
The concept of capturing focus is integral to creating lobbyable game applications and lobby applications with Microsoft® DirectPlay® Voice support. If your game

application does not properly implement focus capture, it is possible that voice communication will not function if your game was launched from a lobby application.

To illustrate this point, consider two players who meet in a lobby application that has DirectPlay Voice support. The two players agree to launch the game. After the game is launched, the lobby application loses focus on each player's computer, and each copy of the game application gains focus. If the game application does not properly gain focus from the lobby application, it is possible that the lobby application can still have focus while the game application is running.

For example, this will occur if the first player's lobby application retains focus while the game session is running while the second player's game session gains focus from the lobby application. From the second player's perspective, the first player's voice session has fallen back to half-duplex. The second player can hear the first player, but the first player cannot hear the second player. From the first player's perspective, the voice session has ended because the second player does not seem to be speaking. Also, the first player does not know that the second player can hear him or her.

Note that this behavior is by design. Consider the same scenario as above, but when the first player attempts to start the game session from the lobby application, there is a problem and the session fails to start. If the second player's session starts successfully, that player can hear the voice of the first player and the first player can inform the second player that their game session failed. Both players might then drop back to the lobby and attempt to start the session once again.

To handle capture focus properly, your game application must set the *hwndAppWindow* parameter in the DVSOUNDDEVICECONFIG structure to the window handle that will have focus when the game is running. The DirectPlay Voice session can then be created through a call to Connect. The game application must then handle the DVMSGID_LOSTFOCUS and DVMSGID_GAINFOCUS messages.

See **Implementing Capture Focus** for more information.

[Visual Basic]
This topic pertains only to applications written in C++.

Jitter Buffers

Microsoft® DirectPlay® Voice features a jitter buffer, an adaptive buffering algorithm that provides optimal voice quality with the least amount of latency.

On busy networks, individual packets of voice data information might arrive in a different sequence from that in which they were encoded on the host computer. Because voice data is sequential in nature, these incoming packets must be queued for a period of time so that delayed packets have an opportunity to arrive and be played back in order.

If the jitter buffer is set to maximize the quality voice communication, it takes longer for the required number of voice packets to arrive and be queued for play. The result is voice latency, and the effect is that voice communication is not heard in real time. Instead, the voice data might be heard anywhere from a fraction of a second to several seconds after it was recorded. This can introduce problems during cooperative gameplay because events can occur in the game but players will not be able to communicate information based on those events in real time. For example, if a player in a first-person shooter is about to be attacked from behind and a teammate attempts to warn the player, the voice communication might not be heard until after the player has been attacked.

If the jitter buffer is set to a reduce latency, the number of packets required to fill the queue is reduced. However, it is possible that not all sequential packets will arrive in time and, as a result, voice data will be missing from the buffer when it is played. The voice communication will be heard much closer to the actual time it was recorded. However, it will have a "broken-up" quality.

The DirectPlay jitter buffer uses two methods to determine how to provide the best quality of voice communication with the least amount of latency. First, network conditions are monitored to determine the amount of lag or network congestion. The size of the jitter buffer, or queue, is then dynamically sized to keep latency as low as possible while providing the least amount of voice break up.

[C++]
The default behavior of DirectPlay Voice jitter buffer is to automatically adjust to network conditions. You can manually adjust how closely the algorithm tracks network conditions using the *dwBufferAggressiveness* and *dwBufferQuality* members of the **DVCLIENTCONFIG** structure. The higher the level of "aggressiveness", the more closely the algorithm monitors network conditions. In general, the higher the quality value, the higher the quality of the voice but the higher the latency. The lower the quality value, the lower the latency but the lower the quality of the voice.

[Visual Basic]
The default behavior of DirectPlay Voice jitter buffer is to automatically adjust to network conditions. You can manually adjust how closely the algorithm tracks network conditions using the *lBufferAggressiveness* and *lBufferQuality* members of the DVCLIENTCONFIG structure. The higher the level of aggressiveness, the more closely the algorithm monitors network conditions. In general, the higher the quality value, the higher the quality of the voice but the higher the latency. The lower the quality value, the lower the latency but the lower the quality of the voice.

It is important to choose an appropriate level of aggressiveness for network conditions when your game application is running because selecting a high level of aggressiveness during times of steady network performance can cause the algorithm to misinterpret a transitory problem and overcompensate for a problem that might not exist.

Working Set Guidelines

Determining the best configuration of transport topology, voice topology, transmission control, and codec depends on the type or genre of game you are creating, the number of players that will participate in a single game session, and the type of connection that will be targeted.

It's important to note that the number of players participating in the voice session is not necessarily the number of players actually participating in the game session. For example, if your game is a first-person shooter, voice communication can be represented in the game as a radio or communicator that is offered as a time-limited powerup. Also, the radio metaphor can be used to limit communication to radios in either vehicles or stationary command stations.

A second example to consider is an online bridge game, which involves four players at one time. Because this is a small working set, it is appropriate to choose a peer-to-peer voice topology transported over a peer-to-peer network topology. This small working set also allows for the use of voice activation as the mode of transmission control. The peer-to-peer voice topology is easily implemented and does not require any player to act as a server. If all four players use the Voxware SC6 codec, the maximum resulting bandwidth is 4.2 Kbps per speech stream, including the codec protocol overhead. Further assuming that game data requires negligible bandwidth, the outgoing maximum bandwidth requirement for an individual speaker is three independent streams to the other three players, or 12.6 Kbps. The incoming stream to any client ranges from 0 if no other players are talking, to 12.6 Kbps if all three other players speak simultaneously. The CPU requirement is 8 percent for encoding and 0 to 12 percent for decoding. This results in a worst-case requirement of 25.2 Kbps. Therefore each player must have a minimum of a 14,400-baud modem.

Another example is a squad combat game that can involve up to 32 players split between 2 teams. Assume that the game data requires a 28,800 baud modem. In this example, there is a larger number of players and it is appropriate to choose a forwarding server voice topology. Again, if all players use the Voxware SC6 codec, the bandwidth requirements are the same as the bridge game above: 4.2 Kbps. In this example we see that there is 4.2 Kbps outgoing when speaking, and a maximum of 12.6 Kbps incoming from the squad. The maximum CPU requirement is 8 percent of a Pentium 200 for encode and 12 percent receiving. Therefore, each player requires 28.8 Kbps for game data, and the greater incoming bandwidth of 12.6 Kbps requires a minimum 41,400 baud rate from each player's modem.

The worst-case scenario for the forwarding server itself is if all 32 players talk at once, requiring 134.4 Kbps. The server CPU use is minimal because the server is not encoding or decoding the streams. It is merely redirecting them. More typically, there might be 16 players talking simultaneously for 67.2 Kbps.

To illustrate the difference between choosing a mixing server voice topology and a forwarding server voice topology, consider the same 32-player squad combat game discussed above. If a mixing server voice topology is used, each client requires 4.2 Kbps to send and 4.2 Kbps to receive. The worst-case bandwidth requirements drop

to 8.4 Kbps and 12 percent of the Pentium processor running at 200 MHz. This reduces the modem requirement to a 33,600 Kbps baud rate for the client.

For the server, the CPU burden changes. The server is now decoding and re-encoding all incoming streams and is also mixing the streams as required. The CPU burden mix the stream is relatively low and is considered negligible. The worst case is the decoding and encoding of 32 simultaneous streams. This results in a requirement of at least a Pentium II processor running at 400 MHz for the voice service alone.

Using DirectPlay

This section of the Microsoft® DirectPlay® documentation is designed to show you how to use the DirectPlay API to implement a number of important aspects of multiplayer applications.

- Using DirectPlay Enumerations

[C++]

- Using Player Context Values

- Implementing a Lobby Client
- Implementing a Lobbyable Application
- Monitoring DirectPlay Network Traffic with Netmon
- Implementing a Callback Function in DirectPlay and DirectPlay Voice
- Implementing a DirectPlay Networking Callback Using Critical Section Objects
- Using the DirectPlay DPNSVR Application

Using DirectPlay Enumerations

[C++]

Enumeration is one of the ways that Microsoft® DirectPlay® provides information to applications. Enumerations are used to collect data associated with a number of similar elements. For instance, you can use an enumeration to obtain information about each player in a session. To perform an enumeration, an application queries DirectPlay for a particular type of information, and DirectPlay returns a data structure for each related element, typically as an array of structures.

Most applications will need to perform at least one and often several enumeration operations. Each DirectPlay interface supports a different set of enumerations. Thus the details will vary from application to application. The following is a list of available enumerations, along with the interfaces that support each type of enumeration.

- Session hosts (**IDirectPlay8Peer**, **IDirectPlay8Client**)
- Service providers (**IDirectPlay8Peer**, **IDirectPlay8Client, IDirectPlay8Server**)
- Players (**IDirectPlay8Peer**, **IDirectPlay8Server**)
- Groups (**IDirectPlay8Peer**, **IDirectPlay8Server**)
- Group members (**IDirectPlay8Peer**, **IDirectPlay8Server**)
- Local programs (**IDirectPlay8LobbyClient**)

For more information, see Implementing a DirectPlay Enumeration.

[Visual Basic]
Enumeration is one of the ways that Microsoft® DirectPlay® provides information to applications. Enumerations are used to collect data associated with a number of similar elements. For instance, you can use an enumeration to obtain information about each player in a session. To perform an enumeration, an application queries DirectPlay for a particular type of information, and DirectPlay returns a data type for each related element, typically as an array of types.

Most applications will need to perform at least one and often several enumeration operations. Each DirectPlay object supports a different set of enumerations. Thus the details will vary from application to application. The following is a list of available enumerations, along with the objects that support each type of enumeration.

- Session hosts (**DirectPlay8Peer**, **DirectPlay8Client**)
- Service providers (**DirectPlay8Peer**, **DirectPlay8Client, DirectPlay8Server**)
- Players (**DirectPlay8Peer**, **DirectPlay8Server**)
- Groups (**DirectPlay8Peer**, **DirectPlay8Server**)
- Group members (**DirectPlay8Peer**, **DirectPlay8Server**)
- Local programs (**DirectPlay8LobbyClient**)

For more information, see Implementing a DirectPlay Enumeration.

Implementing a DirectPlay Enumeration

[C++]
Most Microsoft® DirectPlay® enumerations follow a standard pattern: call the appropriate enumeration method, and examine the returned data array. The exception to this pattern is host enumerations, which are discussed separately.

Before you can call the method to obtain the data, you must allocate enough memory to contain the returned data block. However, you typically do not know, in advance, how large that array will be. Allocating a large enough block of memory to hold any

conceivable array will work, but is inefficient. Instead DirectPlay is allows you query for the required array size, and then repeat the query to obtain the structure itself.

The following procedure outlines how to enumerate the members of a group in a peer-to-peer game. The same general procedure is followed by all other types of enumeration, except for host enumerations. Because enumerations are often used to obtain a snapshot of information that might be changing, you should perform enumerations in a loop until you are successful.

1. Call **IDirectPlay8Peer::EnumGroupMembers**. This method returns an integer array in the *prgdpnid* parameter that contains the ID of each player in the group. The *pcdpnid* parameter is used to indicate the number of elements in the array. Set the *pcdpnid* parameter to 0 to request the appropriate value. Set *prgdpnid* to NULL.
2. When the method returns, *pcdpnid* will point to the number of elements that will be in the array.
3. Allocate your array using the returned *pcdpnid* value, and assign the array to the *prgdpnid* parameter.
4. Set *pcdpnid* to the value that was returned in the first method call.
5. Call IDirectPlay8Peer::EnumGroupMembers again.
6. When the method returns the second time, check the return value. If successful, the method will return S_OK, and the array will contain the player's IDs.
7. If the method returns DPNERR_BUFFERTOOSMALL again, the number of players has increased since the previous method call. Return to step three and use the new *pcdpnid* value to increase the array size. Be careful not to leak memory.

In some cases, the method returns an array of structures. In that case, you follow the same procedure, but the value returned from the first method call gives you the size of the array in bytes, instead of the number of elements in the array. Refer to the individual method references for details.

For more information, see Enumerating Hosts.

[Visual Basic]
Most DirectPlay enumerations follow a standard pattern: determine how many items are to be enumerated and call the appropriate enumeration method. The exception to this pattern is host enumerations, which are discussed separately.

The following procedure outlines how to enumerate the members of a group in a peer-to-peer game. The same general procedure is followed by all other types of enumeration, except for host enumerations.

1. Call **DirectPlay8Peer.GetCountGroupMembers**. This method returns the number of members in the group.
2. Call **DirectPlay8Peer.GetGroupMember** once for each member in the group. The method returns the ID of the specified player.

For more information, see Enumerating Hosts.

Enumerating Hosts

One way to arrange a session is to have session hosts advertise themselves as available. This type of session is referred to as a *broadcast session.* Peers or clients can look for a game to join by enumerating the available hosts, selecting one, and then join the game by sending a connection request. See Peer-to-Peer Sessions or Client/Server Sessions for a detailed discussion.

Unlike other enumerations, the information needed to respond to a request for available hosts is not stored on the local computer. Instead, a client or peer must broadcast a request, for instance on their local subnet, and wait for available hosts to respond. Hosts, on the other hand, must wait for these requests, and then respond appropriately. There are thus two slightly different procedures, depending on whether you are a potential session member, or a session host.

[C++]
The following procedure illustrates how enumerate the available hosts for a peer-to-peer session. The procedure for a client/server session is essentially the same. **IDirectPlay8Peer::EnumHosts** is the method that starts the enumeration. The key parameters to set are *pApplicationDesc*, *pdpaddrDeviceinfo,* and *pdpaddrHost.*

1. Assign the GUID of the game you are interested in playing to the **guidApplication** member of the **DPN_APPLICATION_DESC** structure and assign the structure pointer to the *pApplicationDesc* parameter.
2. Create an address object for your device and assign its pointer to *pdpaddrDeviceinfo.* This object contains the information needed to make a network connection.
3. To query a specific computer for available hosts, create a host address object for that computer and assign its pointer to *pdpaddrHost*. If you set this parameter to NULL, Microsoft® DirectPlay® will create an address object from the information contained in *pdpaddrDeviceinfo.* See DirectPlay Addressing for further discussion of address objects. If you are using an IP or IPX service provider, the query will then normally be broadcast to your local subnet. If you set the DPNENUMHOSTS_OKTOQUERYFORADDRESSING flag, the service provider may display a dialog box to the user to request address information.
4. Call **IDirectPlay8Peer::EnumHosts**.
5. Your callback message handler will then receive a series of **DPN_MSGID_ENUM_HOSTS_RESPONSE** messages, one for each host that responds.
6. Examine the information returned to your message handler, select a session, and ask to join it by calling **IDirectPlay8Peer::Connect**.

If you want to be the host of a broadcast session, advertise yourself as available, and wait for queries or connection requests. The following procedure applies to peer-to-peer hosts, but is essentially similar to the procedure for client/server hosts.

1. Call **IDirectPlay8Peer::SetPeerInfo** to specify the static settings for your player.
2. Specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** structure.
3. Call **IDirectPlay8Peer::Host** to advertise yourself as a potential host. Set the *pdnAppDesc* parameter to the **DPN_APPLICATION_DESC** structure defined in the previous step.
4. Wait for enumeration requests. They will take the form of a **DPN_MSGID_ENUM_HOSTS_QUERY** message sent to your callback message handler. If you wish to respond to the enumeration request, fill in the DPN_APPLICATION_DESC and return S_OK. The peer will receive a **DPN_MSGID_ENUM_HOSTS_RESPONSE** message with the information.
5. If the peer decides that they would like to join your session, you will receive a **DPN_MSGID_INDICATE_CONNECT** message.

See the Peer-to-Peer Sessions and Client/Server Sessions sections for further discussion of how to arrange and launch a game.

[Visual Basic]
The following procedure illustrates how enumerate the available hosts for a peer-to-peer session. The procedure for a client/server session is essentially the same. **DirectPlay8Peer.EnumHosts** is the method that starts the enumeration. The key parameters to set are *ApplicationDesc*, *Deviceinfo,* and Addr*Host.*

1. Assign the GUID of the game you are interested in playing to the **guidApplication** member of the **DPN_APPLICATION_DESC** type and assign assign it to the *ApplicationDesc* parameter.
2. Create a **DirectPlay8Address** object for your device and assign it to *Deviceinfo.* This object contains the information needed to make a network connection.
3. To query a specific computer for available hosts, create a **DirectPlay8Address** object for the host computer and assign it to *pdpaddrHost*. If you set leave the object empty, DirectPlay will create an address from the information contained in *pdpaddrDeviceinfo.* See DirectPlay Addressing for further discussion of address objects. If you are using an IP or IPX service provider, the query will then normally be broadcast to your local subnet.
4. Call **DirectPlay8Peer.EnumHosts**.
5. DirectPlay will then make a series of calls to your message handler's **DirectPlay8Event.EnumHostsResponse** method, once for each host that responds.
6. Examine the information returned to your message handler, select a session, and ask to join it by calling **DirectPlay8Peer.Connect**.

If you want to be the host of a broadcast session, advertise yourself as available, and wait for queries or connection requests. The following procedure applies to peer-to-peer hosts, but is essentially similar to the procedure for client/server hosts.

1. Call **DirectPlay8Peer.SetPeerInfo** to specify the static settings for your player.
2. Specify the configuration of the game by assigning values the **DPN_APPLICATION_DESC** type.
3. Call **DirectPlay8Peer.Host** to advertise yourself as a potential host. Set the *AppDesc* parameter to the **DPN_APPLICATION_DESC** type defined in the previous step.
4. Wait for enumeration requests. They will take the form of a call to your message handlers **DirectPlay8Peer.EnumHostsQuery** method. If you wish to respond to the enumeration request, fill in the **DPN_APPLICATION_DESC** type and set *fRejectMsg* to False.
5. If the peer decides that they would like to join your session, DirectPlay will call your message handler's **DirectPlay8Event.IndicateConnect** method.

See the Peer-to-Peer Sessions and Client/Server Sessions sections for further discussion of how to arrange and launch a game.

Using Player Context Values

[C++]
Most applications will want to associate some data with each player. However, when you receive a message that is associated to a player, you need some way to access that data quickly. Player context values are designed to provide you with an efficient way to access your player data.

Note

Only the **IDirectPlay8Peer** and **IDirectPlay8Server** interfaces use player context values. They are not needed for the **IDirectPlay8Client** interface because clients use this interface to communicate only with the server, not other clients.

- Defining a Player Context Value
- Managing Player Context Data

[Visual Basic]
This topic pertains only to applications written in C++.

Defining a Player Context Value

[C++]
To user player context values, you need to have a block of data on your system for each player, typically in the form of a structure. A player context value is normally an index into an array of pointers to the various players' data blocks. When you receive a message from a player, there is no need for time-consuming operations such as searching for the player's ID in a table. The index contained in the player context value allows you to quickly obtain the necessary pointer.

You define a player context value when you handle the **DPN_MSGID_CREATE_PLAYER** message that notifies you that a player has been added to the game. Host's can also define a player context value when they handle the **DPN_MSGID_INDICATE_CONNECT** message. That player context value will be set in the subsequent **DPN_MSGID_CREATE_PLAYER** message. When the host processes that message, it has the option of changing the player context value. To create a player context value:

- Allocate a structure to hold the player's data.
- Add the structure pointer to your player data array.
- Assign the index of that pointer to the **pvPlayerContext** member of the message's **DPNMSG_CREATE_PLAYER** structure.

Microsoft® DirectPlay® does not specify how you should obtain the data to populate the structure. Each game is responsible for handling that issue in its own way.

Note

The only place you can define a player context value is in a **DPN_MSGID_CREATE_PLAYER** or **DPN_MSGID_INDICATE_CONNECT** message handler. Once the **DPN_MSGID_CREATE_PLAYER** message handler returns, the player context value is set. For each subsequent message associated with that player, the player context value will be the same value that was set by the **DPN_MSGID_CREATE_PLAYER** message handler. You can modify the contents of the associated data structure, but you cannot change the player context value itself.

[Visual Basic]
This topic pertains only to applications written in C++.

Managing Player Context Data

[C++]

While player context values are fairly straightforward to handle, there are a couple of issues that you need to be careful with.

The player context value provides you with a quick way to obtain a valid memory address that will presumably be accessed each time a message arrives. However, you must be careful that different parts of your application do not access the data at the same time. Microsoft® DirectPlay® serializes messages associated with a particular player, which guarantees that you will never be handling two messages from the same player at the same time. As long as you only access the data structure from your callback message handler, you can safely access the structure. However, most applications will need to access player data outside the message handler.

If your application accesses the data outside the callback message handler, you must prevent concurrent access by providing some sort of global mechanism to lock the structure. Even if your application does not require such locking in the early stages of development, you should assume that locking will eventually be required, and build it in from the beginning. If your player context values that are indexes into an array, you should also make sure that you read and update that array safely.

Don't deallocate a player's data structure prematurely. When a player leaves the game, you will normally want to deallocate their data structure and free the associated memory. However, be careful about deallocating the structure as soon as you receive a **DPN_MSGID_DESTROY_PLAYER** message. If your application accesses that structure outside the callback message handler, that data may still be in use when the message arrives. If you simply deallocate the structure as soon as the message arrives, you may cause other parts of your application to fail.

To avoid prematurely deallocating the structure, you should not only provide an application-level locking mechanism, you should also implement some sort of reference counting. Increment this reference count when you create the structure, and every time you use it. Decrement the reference count every time you have finished with the structure, including in your **DPN_MSGID_DESTROY_PLAYER** message handler. As long as the reference count is non-zero, some part of your application is accessing the structure. Do not deallocate the structure until the reference count drops to zero.

[Visual Basic]
This topic pertains only to applications written in C++.

Implementing a Lobby Client

A lobby client is an application that resides on a user's computer. It typically serves as a link between a game application on the user's computer and a lobby server on a remote computer. However, lobby clients can also function as stand-alone

applications. For instance, they can be used to arrange a game session among the users of a particular LAN subnet.

Lobby clients typically consist of three primary components that handle the following tasks:

- Communicating with the outside world, either a lobby server or other lobby clients.
- Communicating with the user, typically through a graphical user interface (GUI).
- Communicating with Microsoft® DirectPlay®.

DirectPlay does not specify how the first two items should be implemented. Lobby client vendors should use whatever approach is suitable to their product. What DirectPlay provides is a standard API that a lobby client can use to communicate with DirectPlay, and through DirectPlay with the user's lobbyable game applications.

This section discusses the following essential details of lobby client implementation.

- Initializing a Lobby Client
- Launching a Lobbied Application
- Implementing a Lobby Client Message Handler
- Communicating with a Lobbied Application
- Closing Down a Lobby Client

See the LobbyClient sample application for a fully implemented example of a simple lobby client.

Initializing a Lobby Client

[C++]
Lobby clients are either launched by a lobby server or directly by a user. Once a lobby client is launched, it must be initialized before it can launch an application. Initialization involves the following tasks.

- Call **CoCreateInstance** to create a lobby client object (CLSID_DirectPlay8LobbyClient). Use the *riid* parameter to request a **IDirectPlay8LobbyClient** interface (**IID_IDirectPlay8LobbyClient)**.
- Call the lobby client's **IDirectPlay8LobbyClient::Initialize** method. Pass the method a pointer to your lobby client's callback message handler.
- Use the **IDirectPlay8LobbyClient::EnumLocalPrograms** method to enumerate the lobbyable applications on the user's system.

The first two steps create the lobby client object, and set up a communication link between that object and your lobby client. The final step determines what lobbyable applications are available on the user's system. You need this information in order to launch the selected application.

The following code sample illustrates how to enumerate local applications. It is a simplified version of the **EnumRegisteredApplications** function in the SDK's LobbyClient sample. Error handling and dialog-box-related code has been deleted for clarity. See the LobbyClient sample in the SDK for the complete code.

```
HRESULT EnumRegisteredApplications()
{
    HRESULT hr;
    DWORD   dwSize     = 0;
    DWORD   dwPrograms = 0;
    DWORD   iProgram;
    BYTE*   pData   = NULL;

  //g_pLobbyClient is a pointer to an IDirectPlay8LobbyClient interface
  //Start with a NULL data buffer. The required buffer size is
  //returned through dwSize.
   hr = g_pLobbyClient->EnumLocalPrograms( NULL, pData, &dwSize, &dwPrograms, 0 );

   if( dwSize == 0 )
   {
  //No registered applications.
   }
  //Set the data buffer to the appropriate size
   pData = new BYTE[dwSize];
   hr = g_pLobbyClient->EnumLocalPrograms( NULL, pData, &dwSize, &dwPrograms, 0 )

  //Cast the returned data to the appropriate structure type
   DPL_APPLICATION_INFO* pAppInfo = (DPL_APPLICATION_INFO*) pData;

  //Enumerate the names of the registered applications
   for( iProgram=0; iProgram<dwPrograms; iProgram++ )
   {
     TCHAR strAppName[MAX_PATH];
     DXUtil_ConvertWideStringToGeneric( strAppName, pAppInfo->pwszApplicationName );

   }
   SAFE_DELETE_ARRAY( pData );
   return S_OK;
}
```

[Visual Basic]

Lobby clients are either launched by a lobby server or directly by a user. Once a lobby client is launched, it must be initialized before it can launch an application. Initialization involves the following tasks.

- Create a **DirectPlay8LobbyClient** object.

Create a **DirectPlay8LobbyEvent** object. The **DirectPlay8LobbyEvent** object is essentially a message handler that receives messages directly from the lobby client object, and indirectly from the application. It is not provided by Microsoft® DirectPlay® and must be implemented by your application.

- You must register this object with DirectPlay by calling **DirectPlay8LobbyClient.RegisterMessageHandler**.
- Use the **DirectPlay8LobbyClient.GetLocalProgram** method to enumerate the lobbyable applications on the user's system.

The first two steps create the lobby client object, and set up a communication link between that object and your lobby client. The final step determines what lobbyable applications are available on the user's system. You need this information in order to launch the selected application.

You should store the application GUIDs of the registered applications because you need them to launch the application.

Depending on the design of your lobby client, you may also want to do one or more of the following tasks.

- Perform any initialization that is not related to Microsoft® DirectPlay®, such as establishing a link with the lobby server.

[C++]

- Create a DirectPlay peer or DirectPlay client object (CLSID_DirectPlay8Peer or CLSID_DirectPlay8Client, respectively).

[Visual Basic]

- Create a **DirectPlay8Peer** or **DirectPlay8Client** object.

- Use the peer or client object's enumeration methods to enumerate available service providers, adapters, and so on.

Launching a Lobbied Application

[C++]

Once the user has selected an application, and your lobby client has verified that it has been registered on the user's system, you can launch the application. To do so, call **IDirectPlay8LobbyClient::ConnectApplication**. The first parameter is a **DPL_CONNECT_INFO** structure that contains a variety of information needed to launch the application including the following:

- The GUID that identifies the application.

- The connection settings, including the user's Microsoft® DirectPlay® address. See DirectPlay Addressing for a discussion of DirectPlay addresses.
- Whether the application will be a host.

The **IDirectPlay8LobbyClient::ConnectApplication** method returns an application handle that is used to identify the application in all further communication. Once the application has launched and the connection successfully established, your message handler receives a DPL_MESSAGE_CONNECT message.

Note

Your message handler may receive the DPL_MESSAGE_CONNECT before the **IDirectPlay8LobbyClient::ConnectApplication** method has confirmed the connection by returning a success code. Your message handler should be prepared to handle the message even if the method has not yet returned.

[Visual Basic]
Once the user has selected an application, and your lobby client has verified that it has been registered on the user's system, you can launch the application. To do so, call **DirectPlay8LobbyClient.ConnectApplication**. The first parameter is a **DPL_CONNECT_INFO** structure that contains a variety of information needed to launch the application including the following:

- The GUID that identifies the application.
- The connection settings, including the user's Microsoft® DirectPlay® address. See DirectPlay Addressing for a discussion of DirectPlay addresses.
- Whether the application will be a host.

The **DirectPlay8LobbyClient.ConnectApplication** method returns an application handle that is used to identify the application in all further communication. Once the application has launched and the connection successfully established, DirectPlay will call your message handlers **DirectPlay8LobbyEvent.Connect** method.

Note

Your message handler's **DirectPlay8LobbyEvent.Connect** method may be called before the **DirectPlay8LobbyClient.ConnectApplication** method has confirmed the connection by returning a success code. Your message handler should be prepared to handle the message even if the method has not yet returned.

Implementing a Lobby Client Message Handler

[C++]

The message handler is a callback function that is used by the lobby client object to communicate with the lobby client. The lobby client message handler has three parameters that pass in the following information.

- A message ID that indicates the message type.
- A pointer to a message data block. You must cast this parameter to the structure that is used by the particular message.
- A pointer to an optional application-defined user-context data block.

The user context value is defined by the lobby client when it calls **IDirectPlay8LobbyClient::Initialize**. It can be used for such purposes as differentiating between messages that are sent from different objects. See **PFNDPNMESSAGEHANDLER** for a complete description the message handler function.

Your message handler must be able to handle the following five lobby client-specific messages.

- **DPL_MESSAGE_CONNECT**
- **DPL_MESSAGE_CONNECTION_SETTINGS**
- **DPL_MESSAGE_DISCONNECT**
- **DPL_MESSAGE_RECEIVE**
- **DPL_MESSAGE_SESSION_STATUS**

Most of these messages are generated by the lobby client object in response to changes in the game status, or when the lobby client requests information. The exception is **DPL_MESSAGE_RECEIVE**. This message is used to pass data directly from the game application to the lobby client.

Note

Microsoft® DirectPlay® message handlers must be written to work properly in a multithreaded environment, or your application may not function well.

DPL_MSGID_CONNECT

This message is sent by the lobby client following the launch of a lobbyable application. The message indicates that the application has been successfully connected. The associated **DPL_MESSAGE_CONNECT** structure holds a variety of information, including:

- A connection ID. Use this ID when your lobby client needs to send data to the application with **IDirectPlay8LobbyClient::Send**, or release the connection with **IDirectPlay8LobbyClient::ReleaseApplication**.
- Lobby connection data.
- An optional connection context value.

DPL_MSGID_CONNECTION_SETTINGS

This message is sent by DirectPlay whenever an associated lobbyable application calls its **IDirectPlay8LobbiedApplication::SetConnectionSettings** method to modify the session connections. The associated **DPL_MESSAGE_CONNECTION_SETTINGS** structure contains the updated connection information.

DPL_MSGID_DISCONNECT

This message is sent when the lobbyable application disconnects from the session by calling **IDirectPlay8LobbiedApplication::Close**. Your lobby client application should delete the connection from its list and free any data that is associated with the application.

DPL_MSGID_RECEIVE

This message enables an application to pass data to the lobby client. DirectPlay passes the data block from the application to the lobby client in a **DPL_MESSAGE_RECEIVE** structure. It is up to the lobby client to process the data.

DPL_MSGID_SESSION_STATUS

This message is sent by DirectPlay whenever one of the following six changes in the session's status occurs.

- The session is connected.
- The session could not connect.
- The session has been disconnected.
- The session has been terminated.
- The session host has migrated.
- This computer has become the session host.

The type of status change is indicated by the value of the dwStatus field in the associated **DPL_MESSAGE_SESSION_STATUS** structure.

For more information, see A Sample Lobby Client Message Handler.

[Visual Basic]
The message handler is a **DirectPlay8LobbyEvent** object, that receives messages directly from the lobby client object, and indirectly from the application. It is not provided by DirectPlay and must be implemented by your application.

Your message handler must implement all of the following methods:

- **DirectPlay8LobbyEvent.Connect**

- **DirectPlay8LobbyEvent.ConnectionSettings**
- **DirectPlay8LobbyEvent.Disconnect**
- **DirectPlay8LobbyEvent.Receive**
- **DirectPlay8LobbyEvent.SessionStatus**

Most of these methods are called by DirectPlay in response to changes in the game status, or when the lobby client requests information. The exception is **DirectPlay8LobbyEvent.Receive**. This method is called to pass data directly from the game application to the lobby client.

DirectPlay8LobbyEvent.Connect

This method is called by DirectPlay following the launch of a lobbyable application. The message indicates that the application has been successfully connected. The associated **DPL_MESSAGE_CONNECT** type holds a variety of information, including:

- A connection ID. Use this ID when your lobby client needs to send data to the application with **DirectPlay8LobbyClient.Send**, or release the connection with **DirectPlay8LobbyClient.ReleaseApplication**.
- Lobby connection data.
- An optional connection context value.

DirectPlay8LobbyEvent.ConnectionSettings

This message is called by DirectPlay whenever an associated lobbyable application calls its **DirectPlay8LobbiedApplication.SetConnectionSettings** method to modify the session connections. The associated **DPL_MESSAGE_CONNECTION_SETTINGS** type contains the updated connection information.

DirectPlay8LobbyEvent.Disconnect

This message is sent when the lobbyable application disconnects from the session by calling **DirectPlay8LobbiedApplication.Close**. Your lobby client application should delete the connection from its list and free any data that is associated with the application.

DirectPlay8LobbyEvent.Receive

This message enables an application to pass data to the lobby client by calling **DirectPlay8LobbiedApplication.Send**. DirectPlay passes the data block from the application to the lobby client in a **DPL_MESSAGE_RECEIVE** type. It is up to the lobby client to process the data.

DirectPlay8LobbyEvent.SessionStatus

This message is sent by DirectPlay whenever one of the following six changes in the session's status occurs.

- The session is connected.
- The session could not connect.
- The session has been disconnected.
- The session has been terminated.
- The session host has migrated.
- This computer has become the session host.

The type of status change is indicated by the value of the method's *status* parameter.

For more information, see A Sample Lobby Client Message Handler.

A Sample Lobby Client Message Handler

[C++]
The following code is a simplified version of the message handler from the SDK's LobbyClient sample. Error handling code has been removed for clarity. See the sample for a complete version.

```
HRESULT WINAPI DirectPlayLobbyMessageHandler( PVOID pvUserContext,
                                              DWORD dwMessageId,
                                              PVOID pMsgBuffer )
{
  switch( dwMessageId )
  {
    case DPL_MSGID_DISCONNECT:
    {
      PDPL_MESSAGE_DISCONNECT pDisconnectMsg;
      pDisconnectMsg = (PDPL_MESSAGE_DISCONNECT)pMsgBuffer;

      // Free any data associated with the application and
      // Remove the connection from the list
      break;
    }
    case DPL_MSGID_RECEIVE:
    {
      PDPL_MESSAGE_RECEIVE pReceiveMsg;
      pReceiveMsg = (PDPL_MESSAGE_RECEIVE)pMsgBuffer;

      // The lobby application sent data. Process the data and
      // Respond appropriately
```

```
    break;
  }
  case DPL_MSGID_SESSION_STATUS:
  {
    PDPL_MESSAGE_SESSION_STATUS pStatusMsg;
    pStatusMsg = (PDPL_MESSAGE_SESSION_STATUS)pMsgBuffer;

    switch( pStatusMsg->dwStatus )
    {
      case DPLSESSION_CONNECTED: //Session connected
        break;
      case DPLSESSION_COULDNOTCONNECT: //Session could not connect
        break;
      case DPLSESSION_DISCONNECTED: //Session disconnected
        break;
      case DPLSESSION_TERMINATED: //Session terminated
        break;
      case DPLSESSION_HOSTMIGRATED: //Host migrated
        break;
      case DPLSESSION_HOSTMIGRATEDHERE: //Host migrated here
        break;
    }
    case DPL_MSGID_CONNECTION_SETTINGS:
    {
      PDPL_MESSAGE_CONNECTION_SETTINGS pConnectionStatusMsg;
      pConnectionStatusMsg = (PDPL_MESSAGE_CONNECTION_SETTINGS)pMsgBuffer;
      // The application has changed the connection settings.
      break;
    }
  }
  return S_OK;
}
```

[Visual Basic]
This topic pertains only to applications written in C++.

Communicating with a Lobbied Application

[C++]
Much of the lobby client's interaction with an associated application is indirect. The lobby client does something that affects the application, and Microsoft® DirectPlay® sends an appropriate message, and vice versa. For instance, if the lobby client

changes the connection settings, DirectPlay notifies the application and provides the new settings. However, the **IDirectPlay8LobbyClient::Send** method enables the lobby client to send a message directly to the application. DirectPlay passes the data to the application without modification. It is the responsibility of the application to process that data.

The lobbied application can also send data directly to the lobby client. The data is passed to the lobby client's message handler with a DPL_MSGID_RECEIVE message. DirectPlay passes the data to the lobby client without modification. It is up to the lobby client to process the data.

[Visual Basic]
Much of the lobby client's interaction with an associated application is indirect. The lobby client does something that affects the application, and Microsoft® DirectPlay® sends an appropriate message, and vice versa. For instance, if the lobby client changes the connection settings, DirectPlay notifies the application and provides the new settings. However, the **DirectPlay8LobbyClient.Send** method enables the lobby client to send a message directly to the application. DirectPlay passes the data to the application without modification. It is the responsibility of the application to process that data.

The lobbied application can also send data directly to the lobby client. DirectPlay transmits the data by calling the lobby client message handler's **DirectPlay8LobbyEvent.Receive** method. DirectPlay passes the data to the lobby client without modification. It is up to the lobby client to process the data.

Closing Down a Lobby Client

[C++]
When you are ready to close the session, disconnect the application by calling **IDirectPlay8LobbyClient::ReleaseApplication**. The application receives a **DPL_MSGID_DISCONNECT** message.

After releasing the application, perform any cleanup that is necessary, and close the session by calling **IDirectPlay8LobbyClient::Close**. You should then free the lobby client object by calling **IDirectPlay8LobbyClient::Release**.

[Visual Basic]
When you are ready to close the session, disconnect the application by calling **DirectPlay8LobbyClient.ReleaseApplication**. Microsoft® DirectPlay® will call notification by calling its message handler's **DirectPlay8LobbyEvent.Disconnect** method.

After releasing the application, perform any cleanup that is necessary, and close the session by calling **DirectPlay8LobbyClient.Close**.

Implementing a Lobbyable Application

A lobbyable application must be designed to work properly with a lobby client. While a lobby client can launch any application, only *lobby-launched* applications can receive messages from Microsoft® DirectPlay® and from the lobby client. To be lobby launched, an application must be appropriately registered, and it must be able to use a DirectPlay lobbied application object to communicate with DirectPlay and the lobby client.

This section discusses how to implement lobbyable applications.

- Registering a Lobbyable Application
- Handling Lobby Launching
- Implementing a Lobbied Application Callback Message Handler
- Communicating with a Lobby Client
- Closing Down a Lobbied Application

Registering a Lobbyable Application

[C++]
A lobbyable application must be registered before it can be lobby launched. This registration needs to be done only once and is typically handled by the application's setup procedure. Do not attempt to modify the registry directly to register an application as lobbyable. Instead, do the following:

- Call **CoCreateInstance** to create a lobbied application object (CLSID_DirectPlay8LobbiedApplication). Set the *riid* parameter to IID_IDirectPlay8LobbiedApplication to request an **IDirectPlay8LobbiedApplication** interface.
- Call **IDirectPlay8LobbiedApplication::RegisterProgram**.This method takes the information and creates appropriate registry keys and values.

You must provide **IDirectPlay8LobbiedApplication::RegisterProgram** with a variety of information, including the following:

[Visual Basic]
A lobbyable application must be registered before it can be lobby launched. This registration needs to be done only once and is typically handled by the application's setup procedure. Do not attempt to modify the registry directly to register an application as lobbyable. Instead, do the following:

- Create a DirectPlay8LobbiedApplication object.
- Call **DirectPlay8LobbiedApplication.RegisterProgram**.This method takes the information and creates appropriate registry keys and values.

You must provide **DirectPlay8LobbiedApplication.RegisterProgram** with a variety of information, including the following:

- A GUID that is used to identify the application.
- A friendly name for the application.
- The location and name of the application's executable file.
- The location and name of an optional *launcher* application.
- Any command-line arguments that need to be passed to the executable file when it is launched.

Instead of launching the game application, Microsoft® DirectPlay® launches a launcher application. The launcher application then launches the game. Launcher applications can be used, for example, as an anti-piracy measure.

[C++]
To unregister a registered program, call **IDirectPlay8LobbiedApplication::UnregisterProgram**. This method removes the registry entries created by **IDirectPlay8LobbiedApplication::RegisterProgram**.

[Visual Basic]
To unregister a registered program, call **DirectPlay8LobbiedApplication.UnregisterProgram**. This method removes the registry entries created by **DirectPlay8LobbiedApplication.RegisterProgram**.

Handling Lobby Launching

[C++]
The first thing your lobbyable application should do when it is launched is to create and initialize a Microsoft® DirectPlay® lobbied application object. To do this, perform the following tasks.

- Call **CoCreateInstance** to create a lobbied application object (CLSID_DirectPlay8LobbiedApplication). Set the *riid* parameter to IID_IDirectPlay8LobbiedApplication to request an **IDirectPlay8LobbiedApplication** interface.
- Call **IDirectPlay8LobbiedApplication::Initialize** to initialize the lobbied application object. Pass the object a pointer to your lobbied application callback message handler.

Next, determine whether your application was lobby launched. If so, your application needs to set up a communication channel with DirectPlay so that you can effectively manage the session. Do the following to detect whether your application was lobby launched.

- When the **IDirectPlay8LobbiedApplication::Initialize** method returns, examine the *pdpnhConnection* parameter. If this parameter is set to a valid connection handle, the game was lobby launched.
- Examine the DPL_MSGID_CONNECT message you receive through your message handler. This message carries with it a variety of information, including the ID that you use to send messages to the lobby client.

Note

Your message handler may receive the DPL_MSGID_CONNECT message before the **IDirectPlay8LobbiedApplication::Initialize** method returns. Your message handler should be prepared to handle the message appropriately.

If your application was not lobby launched, you can indicate that your application is available to lobby clients for connection by calling **IDirectPlay8LobbiedApplication::SetAppAvailable**. This method is typically called when the application has been launched by the user. However, it can also be used if the user has closed one session but the application is still running and available for another session. In either case, your message handler receives a DPL_MSGID_CONNECT message when the lobby client connects your application to a session.

[Visual Basic]
The first thing your lobbyable application should do when it is launched is to create and initialize a Microsoft® DirectPlay® lobbied application object. To do this, perform the following tasks.

- Create a **DirectPlay8LobbiedApplication** object.
- Create a **DirectPlay8LobbyEvent** message handler object
- Register the message handler object by calling **DirectPlay8LobbiedApplication.RegisterMessageHandler**.

If the application was lobby-launched, DirectPlay will call your **DirectPlay8LobbyEvent.Connect** method. The *dlNotify* parameter will contain a **DPL_MESSAGE_CONNECT** type with connection information. such as address objects for the members of the session.

If your application was not lobby launched, you can indicate that your application is available to lobby clients for connection by calling **DirectPlay8LobbiedApplication.SetAppAvailable**. This method is typically called when the application has been launched by the user. However, it can also be used if the user has closed one session but the application is still running and available for another session. In either case, DirectPlay will call your message handler's

DirectPlay8LobbyEvent.Connect method when the lobby client connects your application to a session.

[C++]
The following sample code illustrates how to initialize a lobbied application, and how to detect whether an application was lobby launched. It is a simplified version of the **InitDirectPlay** function used by the SDK's SimplePeer application. Refer to that sample application for the complete code. In particular, error-handling code has been deleted for clarity. The *g_bWasLobbyLaunched* variable is a global variable that is set to TRUE if the application was lobby launched.

```
HRESULT InitDirectPlay()
{
  DPNHANDLE hLobbyLaunchedConnection = NULL;
  HRESULT hr;

  // Create IDirectPlay8LobbiedApplication
  hr = CoCreateInstance( CLSID_DirectPlay8LobbiedApplication, NULL,
                         CLSCTX_INPROC_SERVER,
                         IID_IDirectPlay8LobbiedApplication,
                         (LPVOID*) &g_pLobbiedApp );

  // Initialize IDirectPlay8LobbiedApplication
  hr = g_pLobbiedApp->Initialize( NULL,
                              DirectPlayLobbyMessageHandler,
                              &hLobbyLaunchedConnection,
                              0 );
  //Check for a valid connection handle. If it is non-NULL
  //the application was lobby launched.
  g_bWasLobbyLaunched = ( hLobbyLaunchedConnection != NULL );

  return S_OK;
}
```

Implementing a Lobbied Application Callback Message Handler

[C++]
The message handler is a callback function that is used by the lobbied application object to communicate with a lobbied application. The lobbied application message handler has three parameters that pass in the following information.

- A message ID that indicates the message type.

- A pointer to a message data block. You must cast this parameter to the structure that is used by the particular message.
- A pointer to an optional application-defined user-context data block.

The user context value is defined by the lobby client when it calls **IDirectPlay8LobbyClient::Initialize**. It can be used for such purposes as differentiating between messages that are sent from different objects. See **PFNDPNMESSAGEHANDLER** for a complete description the message handler function.

Your message handler must to be able to handle the following four lobbied application-specific messages.

- **DPL_MESSAGE_CONNECT**
- **DPL_MESSAGE_CONNECTION_SETTINGS**
- **DPL_MESSAGE_DISCONNECT**
- **DPL_MESSAGE_RECEIVE**

Most of these messages are generated by the lobbied application object in response to changes in the connection, or when the lobbied application requests connection information. The exception is **DPL_MESSAGE_RECEIVE**. This message is used to pass data directly from the lobby client to the game application.

Note

Microsoft® DirectPlay® message handlers must be written to work properly in a multithreaded environment, or your application may not function well.

DPL_MSGID_CONNECT

This message is sent by the lobbied application object when the lobby client calls **IDirectPlay8LobbyClient::ConnectApplication** to connect an application to a session. The associated **DPL_MESSAGE_CONNECT** structure includes the following information.

- A connection ID. Use this ID when your application needs to send data to the lobby client with **IDirectPlay8LobbiedApplication::Send**, or update the session status with **IDirectPlay8LobbiedApplication::UpdateStatus**.
- Lobby connection data.
- An optional connection context value.

DPL_MSGID_CONNECTION_SETTINGS

DirectPlay sends this message whenever an associated lobby client calls its **IDirectPlay8LobbyClient::SetConnectionSettings** method to modify the session connections. The associated **DPL_MESSAGE_CONNECTION_SETTINGS** structure contains the updated connection information.

DPL_MSGID_DISCONNECT

This message is sent when the lobby client disconnects the application from the session by calling **IDirectPlay8LobbyClient::ReleaseApplication**. Your application should delete the connection from its list, and free any data that is associated with the session.

DPL_MSGID_RECEIVE

This message enables a lobby client to pass data to an application. DirectPlay passes the data block from the lobby client to the application in a **DPL_MESSAGE_RECEIVE** structure. It is up to the application to process the data.

For more information, see A Sample Lobbied Application Message Handler.

[Visual Basic]
The message handler is a **DirectPlay8LobbyEvent** object, that receives messages directly from the lobby client object, and indirectly from the application. It is not provided by Microsoft® DirectPlay® and must be implemented by your application.

Your message handler must implement all of the following methods:

- **DirectPlay8LobbyEvent.Connect**
- **DirectPlay8LobbyEvent.ConnectionSettings**
- **DirectPlay8LobbyEvent.Disconnect**
- **DirectPlay8LobbyEvent.Receive**
- **DirectPlay8LobbyEvent.SessionStatus**

Most of these methods are called by DirectPlay in response to changes in the game status, or when the lobbied application requests information. The exception is **DirectPlay8LobbyEvent.Receive**. This method is called to pass data directly from the lobby client to the game application.

DirectPlay8LobbyEvent.Connect

This method is called when the lobby client calls **DirectPlay8LobbyClient.ConnectApplication** to connect an application to a session. The associated **DPL_MESSAGE_CONNECT** type holds a variety of information, including:

- A connection ID. Use this ID when your lobby client needs to send data to the application with **DirectPlay8LobbyClient.Send**, or release the connection with **DirectPlay8LobbyClient.ReleaseApplication**.
- Lobby connection data.
- An optional connection context value.

DirectPlay8LobbyEvent.ConnectionSettings

DirectPlay calls this method whenever an associated lobby client calls its **DirectPlay8LobbyClient.SetConnectionSettings** method to modify the session connections. The associated **DPL_MESSAGE_CONNECTION_SETTINGS** type contains the updated connection information.

DirectPlay8LobbyEvent.Disconnect

This message is sent when the lobby client disconnects the application from the session by calling **DirectPlay8LobbyClient.ReleaseApplication**. Your application should delete the connection from its list, and free any data that is associated with the session.

DirectPlay8LobbyEvent.Receive

This message enables a lobby client to pass data to an application. When the lobby client calls **DirectPlay8LobbyClient.Send**, DirectPlay passes the data to the application by calling **DirectPlay8LobbyEvent.Receive**. It is up to the application to process the data.

DirectPlay8LobbyEvent.SessionStatus

DirectPlay does not call this method for lobbyable application message handlers. You must implement this method, but it can simply return 0.

A Sample Lobbied Application Message Handler

[C++]
The following code is a simplified version of the message handler from the SDK's SimplePeer sample. Error handling code has been removed for clarity. See the sample for a complete version.

```
HRESULT WINAPI DirectPlayLobbyMessageHandler( PVOID pvUserContext,
                                              DWORD dwMessageId,
                                              PVOID pMsgBuffer )
{
  switch( dwMessageId )
  {
    case DPL_MSGID_CONNECT:
    {
      PDPL_MESSAGE_CONNECT pConnectMsg;
      pConnectMsg = (PDPL_MESSAGE_CONNECT)pMsgBuffer;
```

```
    // Connected. Start the session.
    break;
   }
   case DPL_MSGID_DISCONNECT:
   {
    PDPL_MESSAGE_DISCONNECT pDisconnectMsg;
    pDisconnectMsg = (PDPL_MESSAGE_DISCONNECT)pMsgBuffer;

    // Disconnected. Free any data associated with
    // the lobby client.
    break;
   }
   case DPL_MSGID_RECEIVE:
   {
    PDPL_MESSAGE_RECEIVE pReceiveMsg;
    pReceiveMsg = (PDPL_MESSAGE_RECEIVE)pMsgBuffer;

    // The lobby client sent data. Process the data and
    // respond appropriately.
    break;
   }

   case DPL_MSGID_CONNECTION_SETTINGS:
   {
    PDPL_MESSAGE_CONNECTION_SETTINGS pConnectionStatusMsg;
    pConnectionStatusMsg = (PDPL_MESSAGE_CONNECTION_SETTINGS)pMsgBuffer;

    // The lobby client has changed the connection settings.
    break;
   }
  }
  return S_OK;
 }
```

[Visual Basic]
This topic pertains only to applications written in C++.

Communicating with a Lobby Client

[C++]
Much of the lobbied application's interaction with an associated lobby client is indirect. The application does something that affects the lobby client, Microsoft®

DirectPlay® sends an appropriate message, and vice versa. For instance, if the application changes the connection settings, DirectPlay notifies the lobby client, and provides the new settings. However, there are two methods that provide information directly to the lobby client: **IDirectPlay8LobbiedApplication::UpdateStatus** and **IDirectPlay8LobbiedApplication::Send**.

[Visual Basic]
Much of the lobbied application's interaction with an associated lobby client is indirect. The application does something that affects the lobby client, Microsoft® DirectPlay® sends an appropriate message, and vice versa. For instance, if the application changes the connection settings, DirectPlay notifies the lobby client, and provides the new settings. However, there are two methods that provide information directly to the lobby client: **DirectPlay8LobbiedApplication.UpdateStatus** and **DirectPlay8LobbiedApplication.Send**.

You must notify the lobby client when any of the following changes in the game status take place.

- The session is connected.
- The session could not connect.
- The session has been disconnected.
- The session has been terminated.
- The session host has migrated.
- This computer has become the session host.

[C++]
To notify the lobby client of one of these status changes, call **IDirectPlayLobbiedApplication::UpdateStatus**, and set the *dwStatus* parameter to the appropriate value. The lobby client receives a DPL_MSGID_SESSION_STATUS message to notify it of the status change.

The **IDirectPlay8LobbiedApplication::Send** method enables the application to send a message directly to the lobby client. DirectPlay passes the data to the lobby client without modification. It is the responsibility of the lobby client to process that data.

The lobby client can also send data directly to the application. The data is passed to the lobby client's message handler with a DPL_MSGID_RECEIVE message. DirectPlay passes the data to the application without modification. The lobby client must process the data.

[Visual Basic]
To notify the lobby client of one of these status changes, call **DirectPlay8LobbiedApplication.UpdateStatus**, and set the *lStatus* parameter to the

appropriate value. DirectPlay will call the lobby client message handler's **DirectPlay8LobbyEvent.SessionStatus** method to notify it of the status change.

The **DirectPlay8LobbiedApplication.Send** method enables the application to send a message directly to the lobby client. DirectPlay passes the data to the lobby client without modification. It is the responsibility of the lobby client to process that data.

The lobby client can also send data directly to the application. DirectPlay passes the data to the lobby client's message handler by calling its **DirectPlay8LobbyEvent.Receive** method. DirectPlay passes the data to the application without modification. The lobby client must process the data.

Closing Down a Lobbied Application

[C++]
To close a session, call **IDirectPlay8LobbiedApplication::Close**. The lobby client receives a DPL_MSGID_DISCONNECT message to notify it of the disconnection. If you want to keep the application running and connect to another session, indicate that your application is available by calling **IDirectPlay8LobbiedApplication::SetAppAvailable**. Otherwise, call **IDirectPlay8LobbiedApplication::Release** to free the lobbied application object, and shut the application down.

[Visual Basic]
To close a session, call **DirectPlay8LobbiedApplication.Close**. Microsoft® DirectPlay® calls the lobby client message handler's **DirectPlay8LobbyEvent.Disconnect** method to notify it of the disconnection. If you want to keep the application running and connect to another session, indicate that your application is available by calling **DirectPlay8LobbiedApplication.SetAppAvailable**.

Monitoring DirectPlay Network Traffic with Netmon

During game development, you may find it useful to monitor Microsoft® DirectPlay® network traffic, especially when trying understand bugs. The network monitor (netmon) is a standard utility for analyzing network traffic. DirectPlay includes a set of parsers, that allow you to use netmon to analyze four components of DirectPlay messaging: the service provider layer, the transport layer, the session layer, and the voice layer.

- How Netmon Works With DirectPlay

- Configuring Netmon for DirectPlay
- Capturing DirectPlay Network Traffic
- Tips for Using Netmon with DirectPlay

How Netmon Works With DirectPlay

The Microsoft® DirectPlay® protocol stack has three basic layers.

- The voice and session layers share the top level of the stack. Normal messaging passes through the session layer, and voice-related messaging passes through the voice layer.
- The transport layer is the middle of the stack. Both voice and session traffic passes through this layer, which is responsible for such tasks as fragmentation and reassembly of messages and retransmission of lost packets.
- The service provider layer is at the bottom of the stack. All messaging is handled by this layer, which is responsible for communicating with the network. For example, for TCP/IP networking, the service provider uses the Winsock API to communicate with the network stack. Netmon can only parse network traffic that is carried on an IP or IPX service provider.

By installing the DirectPlay parsers, you can use Netmon to analyze the network traffic as it passes through any of these four layers. You can see all DirectPlay traffic by selecting the service provider parser. However, by selecting one of the higher level parsers, you can filter out traffic that may not be of interest.

With the transport layer parser, you see all voice and session traffic, but not low-level traffic such as connection handshaking. Be aware that the transport layer breaks messages that are longer than the network's Maximum Transfer Unit (MTU) into one or more fragments.

The session and voice layer parsers allow you to analyze session and voice-related traffic separately. Both of these parsers are aware of fragmentation, and notify the user, but cannot parse fragmented packets.

Configuring Netmon for DirectPlay

If you have a Windows 2000 Server system, Netmon is already installed. For Windows 2000 Professional, you must purchase a copy of Systems Management Server (SMS). See the Netmon documentation for a general discussion of how to use Netmon. To configure Netmon to handle Microsoft® DirectPlay® traffic:

1. Copy Dp8parser.dll to the appropriate folder. The Netmon root folder is normally installed in the \Winnt\System32 folder. If you have installed SMS, the root folder will be called NetMonFull. For Windows 2000 Server, the root folder will be called NetMon. Depending on which version of Netmon you are using, copy the parser DLL to either the ...\NetMonFull\Parsers, or ...\NetMon\parsers folder.
2. Start Netmon

3. Set the adapter to capture from (Capture|Networks...|Local Computer). Be sure to choose the adapter with the "Dial-up Connection" property set to FALSE.

You are now ready to start capturing traffic.

Capturing DirectPlay Network Traffic

To start the capture process, click the Start Capture button on the Netmon toolbar to open the capture view. Initially, you will see all the traffic that is passing through your adapter. You can filter that raw traffic stream to focus on only those packets that are of interest. By installing the Microsoft® DirectPlay® parsers, you essentially add four DirectPlay-oriented filters to Netmon that allow you to filter everything but DirectPlay traffic from your capture view.

To select a filter:

1. Click the Edit Display Filter button on the Netmon toolbar.
2. Double-click "Protocol == Any".
3. Click "Disable All".
4. Under "Disabled Protocols", double-click DPLAYSESSION, DPLAYSP, DPLAYTRANSPORT, and DPLAYVOICE.

Click OK twice to return to the capture view, and you are ready to start viewing DirectPlay traffic.

You can also apply a filter to the capture process itself, rather than to the capture view. This allows you, for instance, to capture only IP packets with specified source and destination ports. See the Netmon documentation for details.

Tips for Using Netmon with DirectPlay

Here are a few tips to using Netmon with Microsoft® DirectPlay®:

- By default, Netmon only captures 1MB of the most recent traffic. You will probably want to increase this value to at least 10-20 MB.
- Netmon doesn't stream to the hard drive, so all you can see is what is in the capture buffer. To stream captured traffic to a hard drive, you will need to implement your own capturer. See MSDN for details.
- By default, DirectPlay parsing uses the [2302,2400]U{6073} port/socket range to filter IP and IPX packets. If you are using non-standard port/socket ranges, you can have the parsers recognize user-defined port/socket values. To do so, add two DWORD values, MinUserPort and MaxUserPort, to the \HKEY_CURRENT_USER\Software\Microsoft\DirectX\DirectPlay\Parsers registry key. This allows you to extend the port/socket range to [2302,2400]U{6073}U[MinUserPort,MaxUserPort].
- Because the DirectPlay and RTP protocols are both layered on top of the UDP protocol, their parsers may conflict. You should disable the RTP parser when analyzing DirectPlay traffic, and vice versa.

Implementing a Callback Function in DirectPlay and DirectPlay Voice

[Visual Basic]
This topic pertains only to applications written in C++.

[C++]
Microsoft® DirectPlay® and DirectPlay Voice both require you to implement and register several callback functions to handle the events fired by DirectPlay. DirectPlay is multithreaded and will fire multiple events concurrently. Therefore, in order to correctly and reliably access data in DirectPlay callbacks, you are required to implement a method of multithreading synchronization. This is known as making your callback *re-entrant* or *threadsafe*.

Callback Function Structure

The structure of the callback follows standard Microsoft Windows Win32 API programming guidelines.

```
HRESULT WINAPI Callback(
  PVOID pvUserContext,
  DWORD dwMessageType,
  PVOID pMessage );
```

pvUserContext is the a context value you supply when you register the callback function with DirectPlay. If you pass this value to DirectPlay when you register your callback, the context value will be returned when DirectPlay invokes your callback.

dwMessageType is one of the ID values passed to your callback by DirectPlay.

pMessage will contain the message passed by DirectPlay.

Registering Your Callback

Microsoft DirectPlay networking callback functions are of type PFNDPNMESSAGEHANDLER. Depending on the type of networking session, you register the address of your callback function with IDirectPlay8Peer::Initialize, IDirectPlay8Client::Initialize, or IDirectPlay8Server::Initialize. If you are registering a DirectPlay voice callback function, register the address of your callback with IDirectPlayVoiceClient::Initialize or IDirectPlayVoiceServer::Initialize, depending on the type of DirectPlay voice session you wish to create.

The following code snippet demonstrates how to register a callback function with the IDirectPlay8Peer interface.

```
HRESULT WINAPI Callback(PVOID, DWORD, PVOID);
IDirectPlay8Peer* pdp8Peer;
```

```
// Get the server interface
hr = CoCreateInstance( CLSID_DirectPlay8Peer, ...)
...

// Register the callback
hr = pdp8Peer->Initialize(NULL, Callback, 0);
```

Implementing a DirectPlay Networking Callback Using Critical Section Objects

[C++]

Microsoft® DirectPlay® networking and voice callback are multithreaded. Therefore, in order to correctly and reliably access data in DirectPlay callbacks, you are required to implement a method of multithreading synchronization.

Currently, there are three methods of synchronizing mulithreaded callback data.

- Mutex Objects
- Semaphore Objects
- Critical Section Objects

The DirectPlay voice samples that ship with the DirectX 8.0 SDK demonstrate synchronization using Critical Section Objects, and the following topics will also demonstrate how Critical Section Objects are used. If you wish to implement a Mutex or Semaphore Object, these topics are discussed in the Microsoft Platform Software Development Kit (SDK) as well as in many reference books. Implementing any of these synchronization methods requires an expert knowledge level in these areas due of the level of complexity and difficulty in debugging should any issues arise.

```
CRITICAL_SECTION g_csPlayerContext;
InitializeCriticalSection(&g_csPlayerContext);
```

Next, implement the DirectPlay message callback handler.

```
HRESULT WINAPI DirectPlayMessageHandler( PVOID pvUserContext,
                                         DWORD dwMessageId,
                                         PVOID pMsgBuffer )
{
    switch( dwMessageId )
    {
        case DPN_MSGID_CREATE_PLAYER:
        {
            EnterCriticalSection( &g_csPlayerContext );
            //callback is now locked
            //perform operation on player data
```

```
            LeaveCriticalSection( &g_csPlayerContext );
        }
    }
}
```

Finally, during application exit, ensure that you call the **DeleteCriticalSection** function to free the memory associated with your critical section object.

```
DeleteCriticalSection( &g_csPlayerContext );
```

[Visual Basic]
Multithreaded callbacks are not used in Visual Basic.

Using the DirectPlay DPNSVR Application

Microsoft Windows does not allow multiple processes to share a single IP or IPX port. Each application that wants to act as a communication host must use a separate port. This restriction creates several issues, especially when doing such tasks as enumerating running games:

- Avoiding port conflicts. You must choose a port that does not conflict with other applications.
- Managing multiple communications hosts on a single system. Each instance of a host must use a different port. Client applications then have to determine which port a particular host is using.
- Avoiding ports that are already in use. If your preferred port is in use, your application must be able to use another port.

The DPNSVR application addresses these issues by acting as a forwarding service for enumeration requests. When an application begins hosting, it informs DPNSVR which port it is running on. DPNSVR listens on a well-known port, and forwards any enumeration requests to all Microsoft® DirectPlay® hosts on the system. Responses to enumeration requests contain the actual port that the host is connected to. DPNSVR offers developers the following advantages:

- You can write generic enumeration routines that enumerate all the games running on a particular system.
- You can use DirectPlay to select the host's port. Client applications can use the services of DPNSVR to enumerate the running games on a well-known port, and the responses will contain the actual port that the host is connected to.
- You do not have to allow for the situation where your application does not get the port it requests.

- You do not need to be concerned about conflicts with other applications on the system

While most applications will want to use the services of DPNSVR, there are some circumstances where you may want to disable it. Two examples are:

- You know what port you want to use, and only one instance of your application will be running on the computer.
- You want to restrict the ability of players to enumerate your session. If you disable DPNSVR, only those players that know the port that your host is connected to will be able to enumerate your host.

For more information, see How to Use DPNSVR.

How to Use DPNSVR

[C++]
To determine whether DPNSVR is supported by your service provider, call the **GetSPCaps** methods supported by the **IDirectPlay8Peer**, **IDirectPlay8Client**, or **IDirectPlay8Server**. If the service provider supports DPNSVR, the DPNSPCAPS_SUPPORTSDPNSRV flag will be set in the **dwFlags** member or the returned **DPN_SP_CAPS** structure. Only IP and IPX service providers currently support DPNSVR.

Using DPNSVR requires no special effort, because it is selected by default. If you do not want enumeration requests forwarded to your host, you must explicitly disable DPNSVR by setting the DPNSESSION_NODPNSVR flag in the **dwFlags** member of the **DPN_APPLICATION_DESC** structure.

Note

Applications can always enumerate your host if they know the port that it is running on, even if the DPNSESSION_NODPNSVR flag is set.

[Visual Basic]
To determine whether DPNSVR is supported by your service provider, call the **GetSPCaps** methods supported by the **DirectPlay8Peer**, **DirectPlay8Client**, or **DirectPlay8Server**. If the service provider supports DPNSVR, the DPNSPCAPS_SUPPORTSDPNSRV flag will be set in the **dwFlags** member or the returned **DPN_SP_CAPS** type. Only IP and IPX service providers currently support DPNSVR.

Using DPNSVR requires no special effort, because it is selected by default. If you do not want enumeration requests forwarded to your host, you must explicitly disable DPNSVR by setting the DPNSESSION_NODPNSVR flag in the **dwFlags** member of the **DPN_APPLICATION_DESC** type.

Note

Applications can always enumerate your host if they know the port that it is running on, even if the DPNSESSION_NODPNSVR flag is set.

Implementing Capture Focus

[C++]
The concept of capturing focus is integral to creating lobbyable game applications and lobby applications with Microsoft® DirectPlay® Voice support. If your game application does not properly implement focus capture, it is possible that voice communication will not function if your game was launched from a lobby application.

Before the DirectPlay voice session is created, there are several initialization steps to be completed:

```
// Create a DirectPlay voice client interface
hr = CoCreateInstance( CLSID_DirectPlayVoiceClient,
                       NULL,
                       CLSCTX_INPROC_SERVER,
                       IID_IDirectPlayVoiceClient,
                       (LPVOID*) &m_pVoiceClient )

// Set the hwndAppWindow member to the window that will have
// focus when your game application is running
DVSOUNDDEVICECONFIG dvSoundDeviceConfig;

dvSoundDeviceConfig.hwndAppWindow = hWnd;
// Set dwFlags to DVSOUNDCONFIG_STRICTFOCUS so that only application
// windows that are in the foreground will gain focus
dvSoundDeviceConfig.dwFlags = DVSOUNDCONFIG_STRICTFOCUS;

// Connect to the voice session
hr = m_pVoiceClient->Connect( &dvSoundDeviceConfig,
                              pdvClientConfig,
                              DVFLAGS_SYNC )
```

Once the application is connected to the session, message will be received by your DirectPlay voice callback. The two messages which you must handle to properly implement capture focus are **DVMSGID_GAINFOCUS** and **DVMSGID_LOSTFOCUS**.

```
HRESULT CALLBACK DirectPlayVoiceClientMessageHandler(
                       LPVOID lpvUserContext,
                       DWORD dwMessageType,
                       LPVOID lpMessage )
```

```
{
   switch( dwMessageType )
   {
    case DVMSGID_GAINFOCUS:
      // the game application has gained focus in the system
         break;
      case DVMSGID_LOSTFOCUS:
      // the game application has lost focus in the system
         break;
   }
  ...
}
```

If recording is muted, focus will be lost. DirectPlay voice will send a DVMSGID_LOSTFOCUS message to your callback. When recording is unmuted, focus will be regained, with the exception that applications using DVSOUNDCONFIG_STRICTFOCUS will only gain focus is the application window is in the foreground.

```
DVCLIENTCONFIG dvClientConfig;
dvClientConfig.dwFlags = DVCLIENTCONFIG_RECORDMUTE;

// Mute recording, DVMSGID_LOSTFOCUS is sent to DirectPlay
// voice callback function
m_pVoiceClient->SetClientConfig(dvClientConfig);
```

[Visual Basic]
This topic pertains only to applications written in C++.

DirectPlay C++ Samples

The following sample applications demonstrate the use and capabilities of the Microsoft® DirectPlay® application programming interface for the C++ programming language. Refer to the Readme.txt file in each sample folder for details. All sample folders can be found under the SDK root directory, typically C:\mssdk.

- AddressOverride
- ChatPeer
- DataRelay
- LobbyClient
- Maze

- SimpleClientServer
- SimplePeer
- StagedPeer
- VoiceClientServer
- VoiceConnect
- VoiceGroup
- VoicePosition

AddressOverride

AddressOverride demonstrates how to override the Microsoft® DirectPlay® addressing in order to host or connect to another session on the network.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\AddressOverride

ChatPeer

The ChatPeer sample is similar in form to SimplePeer. Once a player hosts or connects to a session, the players can chat with each other by passing text strings.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\ChatPeer

DataRelay

The DataRelay sample is similar to SimplePeer but differs by sending a single target (or everyone) a packet of data with options specified in the dialog box's UI.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\DataRelay

LobbyClient

LobbyClient is a simple lobby client application. It displays all registered Microsoft® DirectPlay® applications on the local system. It enables the user to launch one or more of these applications using a chosen service provider. A launched lobbied application may be told to either join or host a game.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\LobbyClient

Maze

The maze sample is a Microsoft® DirectPlay® client/server application. The client comes in two flavors: a console-based version and a D3D client. The D3D client can optionally be run as screen saver by copying Mazeclient.exe to your \winnt\system32\ folder and renaming it Mazeclient.scr. Doing so will make it a screen saver that can be detected by the display control panel application.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\Maze

SimpleClientServer

The SimpleClientServer sample is a simple client/server application.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\SimpleClientServer

SimplePeer

The SimplePeer sample illustrates how to implement a simple peer-to-peer application. After joining or creating a session, the game begins immediately. Other players may join the session in progress at any time.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\SimplePeer

StagedPeer

The StagedPeer sample connects players together with two dialog boxes that prompt users on the connection settings needed to join or create a session. After the user connects to a session, the sample displays a multiplayer stage that enables all players connected to the same session to chat. Players can start a new game when everyone is ready and the host decides to begin. The host may reject players or close player slots.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\StagedPeer

VoiceClientServer

The VoiceClientServer sample is a simple Microsoft® DirectPlay® voice-based client/server application.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\VoiceClientServer

VoiceConnect

The VoiceConnect sample shows how to network other players to start a Microsoft® DirectPlay® Voice chat session. After joining or creating a session, the players may use a microphone to talk to one other. Other players may join the session in progress at any time.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\VoiceConnect

VoiceGroup

The VoiceGroup sample shows how use Microsoft® DirectPlay® voice to enable users to talk to a specific group of players.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\VoiceGroup

VoicePosition

The VoicePosition sample shows how use Microsoft® DirectPlay® Voice with 3-D positioning. The sample uses a simple 2-D grid to represent a playing field. Players can move around the playing field to hear the effects of 3-D spatialization.

Path
(SDK Root)\Samples\Multimedia\DirectPlay\VoicePosition

DirectPlay Visual Basic Samples

The following sample applications demonstrate the use and capabilities of the Microsoft® DirectPlay® application programming interface for the Visual Basic programming language. Refer to the Readme.txt file in each sample folder for details. All sample folders can be found under the SDK root directory, typically C:\mssdk.

- Chat
- Conferencer
- DataRelay
- DXVB Messenger
- Memory
- SimpleClient
- SimplePeer
- SimpleServer

- SimpleVoice
- StagedPeer
- VoiceGroup

Chat

The Chat sample is similar in form to SimplePeer. Once a player hosts or connects to a session, the players can chat with either other by passing text strings.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\Chat

Conferencer

The Conferencer sample is similar in form to Microsoft NetMeeting.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\Conferencer

DataRelay

The DataRelay sample is similar to SimplePeer but differs by sending a packet of data to a single target (or everyone) with options specified in the dialog box's UI.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\DataRelay

DXVB Messenger

The DXVBMessenger sample is a simple client/server instant messaging application.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\DXVBMessenger

Memory

Memory is a simple game in which you match 'tiles' and try to score the most points.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\Memory

SimpleClient

The SimpleClient sample is a simple application that can connect to a server, and make funny faces to other players on that server.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\SimpleClient

SimplePeer

The SimplePeer sample allows players to make funny faces. Players can either be peers, or the session host.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\SimplePeer

SimpleServer

The SimpleServer sample is a simple Server application that can connect and route client messages.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\SimpleServer

SimpleVoice

The SimpleVoice sample is similar in form to SimplePeer. Once a player hosts or connects to a session, the players can chat with either other.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\SimpleVoice

StagedPeer

The StagedPeer sample connects players together with two dialog boxes that prompt users on the connection settings needed to join or create a session. After the user connects to a session, the sample displays a multiplayer stage that allows all players connected to the same session to chat. Players can start a new game when everyone is ready and the host decides to begin. The host may reject players or close player slots.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\StagedPeer

VoiceGroup

The VoiceGroup sample is similar in form to SimpleVoice. Once a player hosts or connects to a session, the players can chat with either other.

Path
(SDK Root)\Samples\Multimedia\VBSamples\DirectPlay\VoiceGroup

y

DirectPlay C/C++ Reference

Reference material for the Microsoft® DirectPlay® C/C++ application programming interface is divided into the following categories.

- Interfaces
- Functions
- Callback Functions
- System Messages
- Structures
- Return Values

Interfaces

This section contains references for methods of the following Microsoft® DirectPlay® interfaces.

- **IDirectPlay8Peer**
- **IDirectPlay8Client**
- **IDirectPlay8Server**
- **IDirectPlayVoiceClient**
- **IDirectPlayVoiceServer**
- **IDirectPlayVoiceTest**
- **IDirectPlay8LobbyClient**
- **IDirectPlay8LobbiedApplication**
- **IDirectPlay8Address**
- **IDirectPlay8AddressIP**

IDirectPlay8Peer

Applications use the methods of the **IDirectPlay8Peer** interface to create a peer-to-peer Microsoft® DirectPlay® session.

The methods of the **IDirectPlay8Peer** interface can be organized into the following groups.

Session Management	**Close**
	Connect
	EnumHosts

	EnumServiceProviders
	GetApplicationDesc
	GetCaps
	GetConnectionInfo
	GetSPCaps
	Host
	SetApplicationDesc
	SetCaps
	SetSPCaps
	TerminateSession
Message Management	**GetSendQueueInfo**
	Initialize
	ReturnBuffer
	SendTo
Player Management	**DestroyPeer**
	GetPeerInfo
	SetPeerInfo
	GetPlayerContext
Group Management	**AddPlayerToGroup**
	CreateGroup
	DestroyGroup
	EnumPlayersAndGroups
	EnumGroupMembers
	GetGroupContext
	GetGroupInfo
	RemovePlayerFromGroup
	SetGroupInfo
Miscellaneous	**CancelAsyncOperation**
	GetLocalHostAddresses
	GetPeerAddress
	RegisterLobby

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::AddPlayerToGroup

Adds a peer to a group.

When this method is called, all peers connected to the application receive a **DPN_MSGID_ADD_PLAYER_TO_GROUP** message.

```
HRESULT AddPlayerToGroup(
const DPNID idGroup,
const DPNID idClient,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

idGroup
Variable of type **DPNID** that specifies the identifier of the group to add the peer to.

idClient
Variable of type **DPNID** that specifies the identifier of the peer that is added to the group.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNADDPLAYERTOGROUP_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method runs asynchronously and usually returns DPNSUCCESS_PENDING. It may also return one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Remarks

Any peer can add itself or another peer to an existing group. Once the peer is successfully added to the group, all messages sent to the group are also sent to the peer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::CancelAsyncOperation

Cancels asynchronous requests. For instance, several methods of the **IDirectPlay8Peer** interface run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this interface that can run asynchronously return an *hAsyncHandle* parameter.

Specific requests are canceled by passing the *hAsyncHandle* of the request in this method's *hAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, specifying NULL in the *hAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *dwFlags* parameter. If a specific handle is provided to this method, no flags should be set.

HRESULT CancelAsyncOperation(
const DPNHANDLE *hAsyncHandle*,
const DWORD *dwFlags*
);

Parameters

hAsyncHandle
Handle of the asynchronous operation to stop. You receive this handle when you call one of several methods that support asynchronous operations. This value can be set to NULL to stop all operations or a particular type of asynchronous request. If you specify a particular handle for the request, the *dwFlags* parameter must be 0.

dwFlags
Flag that specifies which asynchronous request is to be canceled. One of the following flags can be set.

DPNCANCEL_ENUM
Cancel all asynchronous **IDirectPlay8Peer::EnumHosts** requests. A single **EnumHosts** request can be canceled by specifying the handle returned from the **EnumHosts** method.

DPNCANCEL_CONNECT
Cancel an asynchronous **IDirectPlay8Peer::Connect** request.

DPNCANCEL_SEND
Cancel an asynchronous **IDirectPlay8Peer::SendTo** request.

DPNCANCEL_ALL_OPERATIONS
Cancel all asynchronous **Connect, Send, and SentTo** operations.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_CANNOTCANCEL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHANDLE

Remarks

You can use this method to cancel an asynchronous operation for the **IDirectPlay8Peer::Connect**, **IDirectPlay8Peer::SendTo**, and **IDirectPlay8Peer::EnumHosts** methods. Microsoft® DirectPlay® 8.0 does not support cancellation of other asynchronous operations.

You can cancel a send request by providing the handle returned from the **IDirectPlay8Peer::SendTo** method. A **DPN_MSGID_SEND_COMPLETE**, or **DPN_CONNECT_COMPLETE** system message is still posted to the application's message handler for each asynchronous send request that is sent without the DPNSEND_NOCOMPLETE flag set. Send requests that are canceled by this method return DPNERR_USERCANCEL in the **hResultCode** member of the **DPN_MSGID_SEND_COMPLETE** message.

If you set the DPNCANCEL_ALL_OPERATIONS, DPN_CANCELCONNECT, DPN_CANCELSEND, or DPNCANCEL_ENUM flags in *dwFlags*, DirectPlay will attempt to cancel all matching operations. This method will return an error if any attempted cancellation fails, even though some cancellations may have been successful.

Note

The completion message might not arrive until after this method returns. Do not assume that the operation has been terminated until you have received a **DPN_MSGID_SEND_COMPLETE**, **DPN_MSGID_CONNECT_COMPLETE**, or **DPN_MSGID_ASYNC_OP_COMPLETE** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::Close

Closes the open connection with a session. This method must be called on any object successfully initialized with **IDirectPlay8Peer::Initialize**.

```
HRESULT Close(
const DWORD dwFlags
);
```

Parameters

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_UNINITIALIZED

Remarks

This method will cancel any operations still outstanding. It will block until all callback indications on other threads have returned.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::Connect

Establishes the connection to all the peers in a peer-to-peer session. Once a connection is established, the communication channel on the interface is open and the application should expect messages to arrive immediately. No messages can be sent by way of the **IDirectPlay8Peer::SendTo** method until the connection has completed.

Before this method is called, you can obtain application descriptions and the addresses of the associated hosts by calling **IDirectPlay8Peer::EnumHosts**. The **EnumHosts** method returns a **DPN_APPLICATION_DESC** structure for each currently hosted application. The structure describes the application, including the instance GUID of the application.

If this method is called asynchronously (which is the default choice) and returns DPNSUCCESS_PENDING, when the connection completes a **DPN_MSGID_CONNECT_COMPLETE** message is sent to the application's message handler.

```
HRESULT Connect(
const DPN_APPLICATION_DESC *const pdnAppDesc,
IDirectPlay8Address *const pHostAddr,
IDirectPlay8Address *const pDeviceInfo,
const DPN_SECURITY_DESC *const pdnSecurity,
const DPN_SECURITY_CREDENTIALS *const pdnCredentials,
const void *const pvUserConnectData,
const DWORD dwUserConnectDataSize,
void *const pvPlayerContext,
void *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdnAppDesc
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application. Only some of the members of this structure are used by this method. The only members that you must set are **dwSize** and **guidApplication**. You can also set **guidInstance**, **pwszPassword**, and **dwFlags**.

pHostAddr
Pointer to an **IDirectPlay8Address** interface that specifies the addressing information to use to connect to the computer that is hosting.

pDeviceInfo
Pointer to an **IDirectPlay8Address** interface that specifies the network adapter (for example, NIC, modem, and so on) to use to connect to the server.

pdnSecurity
Reserved. Must be NULL.

pdnCredentials
Reserved. Must be NULL.

pvUserConnectData
Pointer to application-specific data provided to the host or server to further validate the connection. This data is sent to the **DPN_MSGID_INDICATE_CONNECT** message in the **pvUserConnectData** member. This parameter is optional and may be set to NULL if no additional connection validation is provided by the user code.

dwUserConnectDataSize
Variable of type **DWORD** that specifies the size of the data contained in the *pvUserConnectData* parameter.

pvPlayerContext
Pointer to the context value of the local player. This value is preset when the local computer handles the **DPN_MSGID_CREATE_PLAYER** message. This parameter is optional and may be set to NULL.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_CONNECT_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle

A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Peer::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNCONNECT_SYNC flag in *dwFlags.*

dwFlags

Flag that describes the connection mode. You can set the following flags.

DPNCONNECT_OKTOQUERYFORADDRESSING

Setting this flag will display a standard Microsoft® DirectPlay® dialog box, which queries the user for more information if not enough information is passed in this method.

DPNCONNECT_SYNC

Process the connection request synchronously. Your message handler still receives a DPN_MSGID_CONNECT_COMPLETE message, so that you can process any connection reply data from the host. You will receive this message before **IDirectPlay8Peer::Connect** returns.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method runs asynchronously and normally returns DPNSUCCESS_PENDING. It may also return one of the following error values.

DPNERR_HOSTREJECTEDCONNECTION

DPNERR_INVALIDAPPLICATION

DPNERR_INVALIDDEVICEADDRESS

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHOSTADDRESS

DPNERR_INVALIDINSTANCE

DPNERR_INVALIDINTERFACE

DPNERR_INVALIDPASSWORD

DPNERR_NOCONNECTION

DPNERR_NOTHOST

DPNERR_SESSIONFULL

Remarks

Although multiple enumerations can be run concurrently and can be run across the duration of a connection, only one connection is allowed per interface. To establish a connection to more than one application, you must create another interface.

If this method is called asynchronously (which is the default choice), when the connection completes a **DPN_MSGID_CONNECT_COMPLETE** message is posted to the application's message handler. If this method returns an error, no completion message is posted.

When this method is called, a **DPN_MSGID_INDICATE_CONNECT** message is posted to the host's message handler. When the host handles this message, it can specify connection reply data that the player will receive with the **DPN_MSGID_CONNECT_COMPLETE** message. If the host accepts the connection, the connection reply data might contain custom startup information. If the connection was rejected, the connection reply data might contain an explanation of the rejection.

You must call **IDirectPlay8Peer::Close** to end the connection to the host.

Note

If you set the DPNCONNECT_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::CreateGroup

Creates a group in the current session. A group is a logical collection of players.

Note

Multicasting is not supported for this release.
When this method is called, all peers connected to the application receive a **DPN_MSGID_CREATE_GROUP** system message.

```
HRESULT CreateGroup(
const DPN_GROUP_INFO *const pdpnGroupInfo,
VOID *const pvGroupContext,
VOID *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that contains the group description.

pvGroupContext

Pointer to the group's context value. This value is preset when the local application's message handler receives the associated DPN_MSGID_CREATE_GROUP message. This parameter is optional and may be set to NULL.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle

A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags

Flag that controls how this method is processed. The following flag can be set for this method.

DPNCREATEGROUP_SYNC

Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return the following error value.

DPNERR_INVALIDFLAGS

Remarks

Microsoft® DirectPlay® does not maintain hierarchical groups because these can easily be implemented with flat groups and expeditious use of the group data.

All peers receive a **DPN_MSGID_CREATE_GROUP** message when this method is called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::DestroyPeer

Deletes a peer from the session.

HRESULT DestroyPeer(
const DPNID *dpnidClient*,
void *const *pDestroyInfo*,
const DWORD *dwDestroyInfoSize*,

const DWORD *dwFlags*
);

Parameters

dpnidClient
Variable of type **DPNID** that specifies the identifier of the peer to delete.

pDestroyInfo
Pointer to a value that describes additional delete data information.

dwDestroyInfoSize
Variable of type **DWORD** that specifies the size of the data contained in the *pDestroyInfo* parameter.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

DPNERR_NOTHOST

Remarks

A player can only be deleted by the session host. The deleted player will be notified through a **DPN_MSGID_TERMINATE_SESSION** message. The structure associated with the message will contain the data passed through the *pDestroyInfo* parameter. If any other session member calls this method, it will fail, and return DPNERR_NOTHOST.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::DestroyGroup

Deletes a group created by the **IDirectPlay8Peer::CreateGroup** method. This method can be called by any peer in the session.

HRESULT DestroyGroup(
const DPNID *idGroup*,
PVOID const *pvAsyncContext*,
DPNHANDLE *const *phAsyncHandle*,
const DWORD *dwFlags*

);

Parameters

idGroup
Variable of type **DPNID** that should be set to the identifier of the group to be deleted.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNDESTROYGROUP_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::EnumPlayersAndGroups

Retrieves a list of all the player and/or group identifiers for the session.

HRESULT EnumPlayersAndGroups(
DPNID *const *prgdpnid*,
DWORD *const *pcdpnid*,
const DWORD *dwFlags*
);

Parameters

prgdpnid
Pointer to an array that will be filled with the session's group and/or player identifiers.

pcdpnid
Pointer to a variable of type **DWORD** that specifies the number of identifiers that can be contained in the buffer pointed to by *prgdpnid*. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the number of entries that are required.

dwFlags
Flag that describes enumeration behavior. You can set one or both of the following flags.

DPNENUM_PLAYERS
Return a list of player identifiers.

DPNENUM_GROUPS
Return a list of group identifiers.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

Remarks

Because group and player information changes frequently, the required buffer size returned may change between subsequent calls. Check and reallocate the buffer until the method succeeds.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::EnumGroupMembers

Retrieves a list of all players in a group.

HRESULT EnumGroupMembers(
const DPNID *dpnid*,
DPNID *const *prgdpnid*,
DWORD *const *pcdpnid*,
const DWORD *dwFlags*
);

Parameters

dpnid
Variable of type **DPNID** that specifies the group that contains the players to enumerate.

prgdpnid
Pointer to an array that will contain the identifiers of the group's players.

pcdpnid
Pointer to a variable of type **DWORD** that specifies the number of identifiers that can be contained in the buffer pointed to by *dpnid*. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the number of entries that are required.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Remarks

Because player information changes frequently, the required buffer size returned may change between subsequent calls. Check and reallocate the buffer until the method succeeds.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::EnumHosts

Enumerates applications that host Microsoft® DirectPlay® games. When an application is found that meets the enumeration criteria, the application's message handler is called with a **DPN_MSGID_ENUM_HOSTS_RESPONSE** system message. The message contains a **DPN_APPLICATION_DESC** structure describing the applications found.

Any number of enumerations can be run concurrently. The *pvUserContext* value is provided in the message handler to help differentiate replies to different enumerations.

Because of the variation in the number of ways enumeration can happen, an application should not attempt to specify *dwEnumPeriod, dwRetryInterval,* or *dwTimeOut* unless the application has some specific media knowledge.

The default enumeration count and timeout values will cause **EnumHosts** to complete within a reasonable amount of time. These values are set by the service provider, and can be obtained by calling **IDirectPlay8Peer::GetSPCaps**. Asynchronous enumerations can be stopped at any time by calling **IDirectPlay8Peer::CancelAsyncOperation** and either passing the handle returned in the *pAsyncHandle* parameter or setting the DPENUM_CANCEL flag in the *dwFlags* parameter. An enumeration can also be stopped by returning anything other than S_OK from the message handler.

HRESULT EnumHosts(
PDPN_APPLICATION_DESC const *pApplicationDesc*,
IDirectPlay8Address *const *pdpaddrHost*,
IDirectPlay8Address *const *pdpaddrDeviceInfo*,
PVOID const *pvUserEnumData*,
const DWORD *dwUserEnumDataSize*,
const DWORD *dwEnumCount*,
const DWORD *dwRetryInterval*,
const DWORD *dwTimeOut*,
PVOID const *pvUserContext*,
HANDLE *const *pAsyncHandle*
const DWORD *dwFlags*,
);

Parameters

pApplicationDesc

Pointer to a **DPN_APPLICATION_DESC** structure that specifies which application hosts to enumerate. You must set the *pApplicationDesc*.dwSize member to the appropriate value. To reduce the number of responses, set *pApplicationDesc*.guidApplication to the GUID of the application to be found. If this member is not set, the search will include all applications.

pdpaddrHost

Pointer to an **IDirectPlay8Address** object that specifies the address of the computer that is hosting the application. If you specify NULL, DirectPlay will create an address. By default, DirectPlay will create the address from the *pdpaddrDeviceInfo* parameter. If you set the DPNENUMHOSTS_OKTOQUERYFORADDRESSING flag in *dwFlags*, the user can be queried for address information.

pdpaddrDeviceInfo

Pointer to an **IDirectPlay8Address** object that specifies the service provider and local device settings to use when enumerating.

pvUserEnumData

Pointer to a block of data that is sent in the enumeration request to the host. The size of the data is limited depending on the network type. Call **IDirectPlay8Peer::GetSPCaps** to obtain the exact value.

dwUserEnumDataSize
Variable of type **DWORD** that specifies the size of the data pointed at in the *pvUserEnumData* parameter.

dwEnumCount
Value specifying how many times that the enumeration data will be sent. Set this parameter to zero to use the default value. You can obtain the default value for *dwEnumCount* by calling **IDirectPlay8Peer::GetSPCaps**. If *dwEnumCount* is set to INFINITE, the enumeration will continue until canceled.

dwRetryInterval
Value specifying how many milliseconds between enumeration retries. Set this parameter to zero to use the default value. You can obtain the default value for *dwRetryInterval* by calling **IDirectPlay8Peer::GetSPCaps**.

dwTimeOut
Variable of type **DWORD** that specifies the number of milliseconds that DirectPlay will wait for replies after the last enumeration is sent. Set this parameter to zero to use the default value. You can obtain the default value for *dwTimeOut* by calling **IDirectPlay8Peer::GetSPCaps**. If INFINITE is specified, the enumeration continues until it is canceled.

pvUserContext
Context that is provided in the peer's message handler when it is called with responses to the enumeration. This can be useful to differentiate replies from concurrent enumerations.

pAsyncHandle
A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Peer::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNENUMHOSTS_SYNC flag in *dwFlags.*

dwFlags
The following flags can be set.

DPNENUMHOSTS_SYNC
Causes the method to process synchronously.

DPNENUMHOSTS_OKTOQUERYFORADDRESSING
Setting this flag will display a standard DirectPlay® dialog box, which queries the user for more information if not enough information is passed in this method.

DPNENUMHOSTS_NOBROADCASTFALLBACK
If the service provider supports broadcasting, setting this flag will disable the broadcast capabilities. Check to see if broadcasting is supported by examining the **DPN_SP_CAPS** structure before setting this flag.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_INVALIDDEVICEADDRESS

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHOSTADDRESS

DPNERR_INVALIDPARAM

DPNERR_ENUMQUERYTOOLARGE

Remarks

If you set the DPNENUMHOSTS_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::EnumServiceProviders

Enumerates all the registered service providers available to the application.

HRESULT EnumServiceProviders(
const GUID *const *pguidServiceProvider*,
const GUID *const *pguidApplication*,
const DPN_SERVICE_PROVIDER_INFO *const *pSPInfoBuffer*,
DWORD *const *pcbEnumData*,
DWORD *const *pcReturned*
constDWORD *dwFlags*,
);

Parameters

pguidServiceProvider

Pointer to a variable of type GUID that specifies a service provider. This optional parameter forces the enumeration of subdevices for the specified service provider. You should normally set this value to NULL, to enumerate all available service providers.

pguidApplication

Pointer to a variable of type GUID that specifies an application. If a pointer is passed in this parameter, only service providers who can be connected by the application are enumerated. You can also pass NULL to enumerate all the registered service providers for the system.

pSPInfoBuffer

Pointer to an array of **DPN_SERVICE_PROVIDER_INFO** structures that will be filled with service provider information.

pcbEnumData

Pointer to **DWORD** that is filled with the size of the *pEnumData* buffer if the buffer is too small.

pcReturned

Pointer to a variable of type **DWORD** that specifies the number of **DPN_SERVICE_PROVIDER_INFO** structures returned in the *pcbEnumData* array.

dwFlags

The following flag can be specified.

DPNENUMSERVICEPROVIDERS_ALL

Enumerates all the registered service providers for the system, including those that are not available to the application or do not have devices installed.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

Remarks

Call this method initially by specifying NULL in the *pguidServiceProvider* parameter to determine the base service providers available to the system. Specific devices for a service provider can then be obtained by passing a pointer to a service provider GUID in the *pguidServiceProvider*. This is useful, for example, when using the Modem Connection for Microsoft® DirectPlay® service provider. You can choose different modems for dialing out and specific modems for hosting.

If the *pcbEnumData* buffer is not big enough to hold the requested service provider information, the method returns DPNERR_BUFFERTOOSMALL and the p*cbEnumData* parameter contains the required buffer size. Typically, the best strategy is to call the method once with a zero-length buffer to determine the required size. Then call it again with the appropriate-sized buffer.

Normally, this method will return only those service providers that can be used by the application. For example, if the IPX networking protocol is not installed, DirectPlay will not return the IPX service provider. To have DirectPlay return all service providers, even those that cannot be used by the application, set the DPNENUMSERVICEPROVIDERS_ALL flag in *dwFlags*.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetApplicationDesc

Retrieves the full application description for the connected application.

```
HRESULT GetApplicationDesc(
DPN_APPLICATION_DESC *const pAppDescBuffer,
DWORD *const pcbDataSize,
const DWORD dwFlags
);
```

Parameters

pAppDescBuffer
Pointer to a **DPN_APPLICATION_DESC** structure where the application description data is to be written. Set this parameter to NULL to request only the size of data. If *pAppDescBuffer* is not set to NULL, you must set the *pAppDescBuffer*.dwSize member to an appropriate value. The *pcbDataSize* parameter is set to the size required to hold the data.

pcbDataSize
Pointer to a variable of type **DWORD** that is initialized to the size of the buffer before calling this method. After the method returns, this parameter is set to the size, in bytes, of the session data. If the buffer is too small, this method returns the DPNERR_BUFFERTOOSMALL error value, and this parameter is set to the buffer size required. If this parameter is NULL, the method returns DPNERR_INVALIDPARAM.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM
DPNERR_NOCONNECTION

Remarks

Call this method initially by passing NULL in the *pAppDescBuffer* parameter to obtain the size of the required buffer. When you call the method a second time to fill the buffer, be sure to set the structures **dwSize** member to the appropriate value.

The returned DPN_APPLICATION_DESC structure will have the **guidInstance**, **guidApplication**, and **pwszSessionName** members set. It will not contain information about other clients that are connected to the session. That information, if available, can be obtained only from the server application. In particular, the **dwCurrentPlayers** member will always be set to 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetCaps

Retrieves the **DPN_CAPS** structure for the current interface.

```
HRESULT GetCaps(
DPN_CAPS *const pdpnCaps,
const DWORD dwFlags
);
```

Parameters

pdpnCaps
Pointer to a **DPN_CAPS** structure to receive caps information. You must set the **dwSize** member of this structure to an appropriate value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetConnectionInfo

Retrieves statistical information about the connection between the local application and the specified remote player.

```
HRESULT GetConnectionInfo(
const DPNID dpnidEndPoint,
DPN_CONNECTION_INFO *const pdnConnectInfo,
const DWORD dwFlags
);
```

Parameters

dpnidEndPoint
The DPNID of the remote player whose connection information will be retrieved.

pdnConnectInfo
Pointer to a **DPN_CONNECTION_INFO** structure to retrieve information about the specified connection. The **dwSize** member of this structure must be set to the size of a **DPN_CONNECTION_INFO** structure.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED
DPNERR_INVALIDPLAYER

Remarks

This method can be called only after a successful **Host** or **Connect** call has completed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetGroupContext

Retrieves the group context value for the specified group.

```
HRESULT GetGroupContext(
const DPNID dpnid,
PVOID *const ppvGroupContext,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group to retrieve context data for.

ppvGroupContext
Pointer to the context value of the group.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDGROUP

DPNERR_INVALIDPARAM

Remarks

Group context values are set by pointing the **pvGroupContext** member of the **DPN_MSGID_CREATE_GROUP** system message to the context value data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetGroupInfo

Retrieves a block of data associated with a group, including the group name.

This method is typically called after a **DPN_MSGID_GROUP_INFO** system message is received indicating that the group data has been modified.

```
HRESULT GetGroupInfo(
const DPNID dpnid,
DPN_GROUP_INFO *const pdpnGroupInfo,
DWORD *const pdwSize,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group whose data block will be retrieved.

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that describes the group data. If *pdwSize* is not set to NULL, you must set *pdpnGroupInfo*.dwSize to the size of a **DPN_GROUP_INFO** structure.

pdwSize
Pointer to a variable of type **DWORD** that returns the size of the data in the *pdpnGroupInfo* parameter. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the required size.

dwFlags
Reserved. Set to 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Remarks

Microsoft® DirectPlay® returns the **DPN_GROUP_INFO** structure, and the pointers assigned to the structure's **pwszName** and **pvData** members in a contiguous buffer. If the two pointers were set, you must have allocated enough memory for the structure, plus the two pointers. The most robust way to use this method is to first call it with *pdwSize* set to NULL. When the method returns, *pdwSize* will point to the correct value. Use that value to allocate memory for your structure and call the method a second time to retrieve the information.

When the method returns, the **dwInfoFlags** member of the **DPN_GROUP_INFO** structure will always have the DPNINFO_DATA and DPNINFO_NAME flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Peer::SetGroupInfo**, to notify DirectPlay which values have changed.

Transmission of nonstatic information should be handled with the **IDirectPlay8Peer::SendTo** method because of the high cost of using the **IDirectPlay8Peer::SetGroupInfo** method.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetLocalHostAddresses

Retrieves the local addresses being used to host the session.

HRESULT GetLocalHostAddresses(
IDirectPlay8Address **const *prgpAddress*,
DWORD *const *pcAddress*,
const DWORD *dwFlags*
);

Parameters

prgpAddress
A pointer to an array of **IDirectPlay8Address** objects that specify the local host addresses. You must release these objects when you no longer need them, or you will create memory leaks.

pcAddress
The maximum number of address objects that can be returned in the array pointed to by *prgpAddress*. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the required size.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED
DPNERR_NOTHOST

Remarks

The most robust way to use this method is to call it first with *pcAddress* set to 0. When the method returns, *pcAddress* will point to the correct value, and you can use that value to call the method a second time to retrieve the information.

If the caller is not the session host, the method returns DPNERR_NOTHOST. Use **IDirectPlay8Peer::GetPeerAddress** to retrieve the address of a remote player.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetPeerAddress

Retrieves the address for the specified player in the session.

```
HRESULT GetPeerAddress(
const DPNID dpnid,
IDirectPlay8Address **const pAddress,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** specifying the identification of the player.

pAddress
Address of a pointer to an **IDirectPlay8Address** object that specifies the address of the peer. You must release this object when you no longer need it.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPLAYER
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

Use **IDirectPlay8Peer::GetLocalHostAddresses** to retrieve addresses that can be used to connect to the session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetPeerInfo

Retrieves peer information set for the specified peer.

```
HRESULT GetPeerInfo(
const DPNID dpnid,
DPN_PLAYER_INFO *const pdpnPlayerInfo,
DWORD *const pdwSize,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the peer whose information will be retrieved.

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure to fill with peer information. If *pdwSize* is not set to NULL, you must set *pdpnPlayerInfo*.dwSize to the size of a **DPN_PLAYER_INFO** structure.

pdwSize
Pointer to a variable of type **DWORD** that contains the size of the peer data returned in the *pdpnPlayerInfo* parameter. If the buffer is too small this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the size of the required buffer.

dwFlags
Reserved. Set to 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

Remarks

Call this method after the peer receives a **DPN_MSGID_PEER_INFO** message from the application, which indicates a peer has updated their information.

Microsoft® DirectPlay® returns the **DPN_PLAYER_INFO** structure, and the pointers assigned to the structure's **pwszName** and **pvData** members in a contiguous buffer. If the two pointers were set, you must have allocated enough memory for the structure, plus the two pointers. The most robust way to use this method is to first call it with *pdwSize* set to NULL. When the method returns, *pdwSize* will point to the correct value. Use that value to allocate memory for the structure and call the method a second time to retrieve the information.

When the method returns, the **dwInfoFlags** member of the **DPN_PLAYER_INFO** structure will always have the DPNINFO_DATA and DPNINFO_NAME flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Peer::SetPeerInfo**, to notify DirectPlay of which values have changed.

Transmission of nonstatic information should be handled with the **IDirectPlay8Peer::SendTo** method because of the high cost of using the **IDirectPlay8Peer::SetPeerInfo** method.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetPlayerContext

Retrieves the player context value for the specified peer.

```
HRESULT GetPlayerContext(
const DPNID dpnid,
PVOID *const ppvPlayerContext,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the player to get context data for.

ppvPlayerContext
Pointer to the context data of the peer.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDPLAYER

Remarks

Player context values are set by pointing the **pvPlayerContext** member of the **DPN_MSGID_CREATE_PLAYER** system message to the context value data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetSendQueueInfo

Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® will not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should slow the rate that messages are sent.

```
HRESULT GetSendQueueInfo(
DWORD *const pdwNumMsgs,
DWORD *const pdwNumBytes,
const DWORD dwFlags
);
```

Parameters

dpnid
DPNID of the player to get send queue information for.

pdwNumMsgs
Pointer to a variable of type **DWORD** that contains the number of messages currently queued. This value is optional, and may be set to NULL.

pdwNumBytes
Pointer to a variable of type **DWORD** that specifies the total number of bytes of data of the messages currently queued. This value is optional, and may be set to NULL.

dwFlags
You may specify the DPNGETSENDQUEUEINFO_PRIORITY_NORMAL, DPNGETSENDQUEUEINFO_PRIORITY_HIGH, or

DPNGETSENDQUEUEINFO_PRIORITY_LOW flag to inquire about specific messages of that priority.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Remarks

You cannot set both *pdwNumMsgs* and *pdwNumBytes* to NULL. At least one of them must be set to a valid pointer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::GetSPCaps

Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT GetSPCaps(
const GUID *const pguidSP,
DPN_SP_CAPS *const pdpnSPCaps,
const DWORD dwFlags
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to get information about.

pdpnSPCaps
Pointer to a **DPN_SP_CAPS** structure to receive the information about the specified service provider. You must set the *pdpnSPCaps*.dwSize member of the structure.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

This method retrieves information about the specified service provider. A successful call to **IDirectPlay8Peer::Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::Host

Creates a new peer-to-peer session, hosted by the local computer.

```
HRESULT Host(
const DPN_APPLICATION_DESC *const pdnAppDesc,
IDirectPlay8Address **const prgpDeviceInfo,
const DWORD cDeviceInfo,
const DPN_SECURITY_DESC *const pdpSecurity,
const DPN_SECURITY_CREDENTIALS *const pdpCredentials,
VOID *const pvPlayerContext,
const DWORD dwFlags
);
```

Parameters

pdnAppDesc
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application.

prgpDeviceInfo
A pointer to an array of **IDirectPlay8Address** objects containing the device addresses that should be used to host the application. You must release these objects when you no longer need them.

cDeviceInfo
Variable of type **DWORD** that specifies the number of device address objects in the array pointed to by *prgpDeviceInfo*.

pdpSecurity
Reserved. Must be NULL.

pdpCredentials
Reserved. Must be NULL.

pvPlayerContext

Pointer to the context value of the local player. This value is preset when the local computer handles the **DPN_MSGID_CREATE_PLAYER** message. This parameter is optional and may be set to NULL.

dwFlags

The following flag can be specified.

DPNHOST_OKTOQUERYFORADDRESSING

Setting this flag will display a standard Microsoft® DirectPlay® dialog box, which queries the user for more information if not enough information is passed in this method.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_DATATOOLARGE

DPNERR_INVALIDPARAM

DPNERR_INVALIDDEVICEADDRESS

Remarks

If you set the DPNHOST_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

The maximum size of the application data that you assign to the **pvApplicationReservedData** member of the **DPN_APPLICATION_DESC** structure is limited by the service provider's Maximum Transmission Unit. If your application data is too large, the method will fail and return DPNERR_DATATOOLARGE.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::Initialize

Registers an entry point in the peer's code that receives all the messages from the **IDirectPlay8Peer** interface and from remote peers. This method must be called before calling any other methods of this interface.

HRESULT Initialize(
PVOID const *pvUserContext*,
const PFNDPNMESSAGEHANDLER *pfn*,
const DWORD *dwFlags*

);

Parameters

pvUserContext

Pointer to the user-provided context value in calls to the message handler. A user-provided context value can be used to differentiate messages coming from multiple interfaces to a common message handler.

pfn

Pointer to a **PFNDPNMESSAGEHANDLER** callback function that is used to receive all messages from remote peers and indications of session changes from the **IDirectPlay8Peer** interface.

dwFlags

You may specify the following flag.

DPNINITIALIZE_DISABLEPARAMVAL

Passing this flag will disable parameter validation for the current object.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

Call this method first after using **CoCreateInstance** to obtain the **IDirectPlay8Peer** interface.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::RegisterLobby

Allows launched applications to automatically propagate game status to the lobby.

HRESULT RegisterLobby(
const DPNHANDLE *dpnHandle*,
IDirectPlay8LobbiedApplication *const *pIDP8LobbiedApplication*,
const DWORD *dwFlags*
);

Parameters

dpnHandle

The connection handle used when making the calls to **IDirectPlay8LobbiedApplication::UpdateStatus**.

pIDP8LobbiedApplication
Pointer to the **IDirectPlay8LobbiedApplication** object that specifies the application.

dwFlags
One of the following flags:

DPNLOBBY_REGISTER
Registers the lobby with the application.

DPNLOBBY_UNREGISTER
Unregisters the lobby with the application.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::RemovePlayerFromGroup

Removes a peer from a group.

When this method is called all peers connected to the application receive a **DPN_MSGID_REMOVE_PLAYER_FROM_GROUP** message.

```
HRESULT RemovePlayerFromGroup(
const DPNID idGroup,
const DPNID idClient,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

idGroup
Variable of type **DPNID** that specifies the identifier of the group that the peer will be removed from.

idClient

Variable of type **DPNID** that specifies the identifier of the peer that will be removed from the group.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle

A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags

Flag that controls how this method is processed. The following flag can be set for this method.

DPNREMOVEPLAYERFROMGROUP_SYNC

Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

DPNERR_INVALIDPLAYER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::ReturnBuffer

Retrieves message buffers provided to the application through the **pReceiveData** member of the **DPN_MSGID_RECEIVE** system message. If the user's message handler returns DPNSUCCESS_PENDING to the RECEIVE callback, Microsoft® DirectPlay® assumes that ownership of the buffer is transferred to the application, and neither frees nor modifies it until ownership is returned to DirectPlay through this call.

```
HRESULT ReturnBuffer(
const DPNHANDLE hBufferHandle,
const DWORD dwFlags
);
```

Parameters

hBufferHandle

Variable of type **DPNHANDLE** that specifies the buffer handle for the message. This is obtained in the **hBufferHandle** member of the **DPN_MSGID_RECEIVE** system message.

dwFlags

Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDHANDLE

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SendTo

Transmits data to another peer or group within the session by sending a message to the appropriate message handlers. The message can be sent synchronously or asynchronously.

```
HRESULT SendTo(
const DPNID dpnid,
const DPN_BUFFER_DESC *const pBufferDesc,
const DWORD cBufferDesc,
const DWORD dwTimeOut,
void *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

dpnid

Identifier of the peer or group that receives data. Set this parameter to DPNID_ALL_PLAYERS_GROUP to send a message to all players in the session.

pBufferDesc

Pointer to a **DPN_BUFFER_DESC** structure that contains the data to be sent.

cBufferDesc

The number of **DPN_BUFFER_DESC** structures pointed to by *pBufferDesc*. There can only be one buffer in this version of Microsoft® DirectPlay®.

dwTimeOut

Number of milliseconds to wait for the message to send. If the message has not been sent by the *dwTimeOut* value, the message is not sent. If you do not want a time out for message sends, set this parameter to 0.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_SEND_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle

A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Peer::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNSEND_SYNC flag in *dwFlags.*

dwFlags

Flags that describe send behavior. You can set one or more of the following flags.

DPNSEND_SYNC

Process the **SendTo** request synchronously.

DPNSEND_NOCOPY

Use the data in the **DPN_BUFFER_DESC** structure and do not make an internal copy. This may be a more efficient method of sending data. However, it is less robust because the sender might be able to modify message before the receiver has processed it. This flag cannot be combined with DPNSEND_NOCOMPLETE.

DPNSEND_NOCOMPLETE

Does not send the **DPN_MSGID_SEND_COMPLETE** to the message handler. This flag may not be used with DPNSEND_NOCOPY or DPNSEND_GUARANTEED. Additionally, when using this flag the *pvAsyncContext* must be NULL.

DPNSEND_COMPLETEONPROCESS

Sends the **DPN_MSGID_SEND_COMPLETE** to the message handler when this message has been delivered to the target and the target's message handler returns from indicating its reception. There is additional internal message overhead when this flag is set, and the message transmission process may become significantly slower. If you set this flag, DPNSEND_GUARANTEED must also be set.

DPNSEND_GUARANTEED

Sends the message by a guaranteed method of delivery.

DPNSEND_PRIORITY_HIGH

Sets the priority of the message to high. This flag cannot be used with DPNSEND_PRIORITY_LOW.

DPNSEND_PRIORITY_LOW

Sets the priority of the message to low. This flag cannot be used with DPNSEND_PRIORITY_HIGH.

DPNSEND_NONSEQUENTIAL

If this flag is set, the target application will receive the messages in the order that they arrive at the user's computer. If this flag is not set, messages are delivered sequentially, and will be received by the target application in the order that they were sent. Doing so may require buffering incoming messages until missing messages arrive.

DPNSEND_NOLOOPBACK
Suppresses the **DPN_MSGID_RECEIVE** system message to your message handler when you are sending to a group that includes the local player. For example, this flag is useful if you are broadcasting to the entire session.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_CONNECTIONLOST

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

DPNERR_TIMEDOUT

Remarks

This method generates a **DPN_MSGID_RECEIVE** system message in the receiver's message handler. The data buffer is contained in the **pReceiveData** member of the associated structure.

Messages can have one of three priorities: low, normal, and high. To specify a low or high priority for the message, set the appropriate flag in *dwFlags.* If neither of the priority flags is set, the message will have normal priority. See Basic Networking for a discussion of send priorities.

When the **SendTo** request is completed, a **DPN_MSGID_SEND_COMPLETE** system message is normally posted to the sender's message handler. The success or failure of the request is contained in the **hResultCode** member of the associated structure. You can suppress send completions by setting the DPNSEND_NOCOMPLETE flag in *dwFlags.*

Send completions are typically posted on the source computer as soon as the message is sent. In other words, a send completion does not necessarily mean that the message has been processed on the target. It may still be in a queue. If you want to be certain that the message has been processed by the target, set the DPNSEND_COMPLETEONPROCESS flag in *dwFlags.* This flag ensures that the send completion will not be sent until the target's message handler has processed the message, and returned.

Note

Do not assume that resources such as the data buffer will remain valid until the method has returned. If you call this method asynchronously, the **DPN_MSGID_SEND_COMPLETE** message may be received and processed by your message handler before the call has returned. If your message handler deallocates or otherwise invalidates a resource such as the data buffer, that resource may become invalid at any time after the method has been called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SetApplicationDesc

Changes the settings for the application that is being hosted. Only some settings can be changed.

```
HRESULT SetApplicationDesc(
const DPN_APPLICATION_DESC *const pad,
const DWORD dwFlags
);
```

Parameters

pad
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application settings to modify.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_DATATOOLARGE
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM
DPNERR_NOTHOST

Remarks

You can use this method to modify only the following members of the **DPN_APPLICATION_DESC** structure:

- **dwMaxPlayers**
- **pwszSessionName**
- **pwszPassword**

- **pvApplicationReservedData**
- **dwApplicationReservedDataSize**

The maximum size of the application data that you assign to the **pvApplicationReservedData** member of the **DPN_APPLICATION_DESC** structure is limited by the service provider's Maximum Transmission Unit. If your application data is too large, the method will fail and return DPNERR_DATATOOLARGE.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SetCaps

Sets the **DPN_CAPS** structure for the current interface.

```
HRESULT SetCaps(
const DPN_CAPS *const pdpCaps,
const DWORD dwFlags
);
```

Parameters

pdpCaps
Pointer to a **DPN_CAPS** structure used to set the information about the current interface.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SetGroupInfo

Sets a block of data associated with a group, including the name of the group.

Calling this method generates a **DPN_MSGID_GROUP_INFO** message, which is sent to all the peers connected to the application.

```
HRESULT SetGroupInfo(
const DPNID dpnid,
DPN_GROUP_INFO *const pdpnGroupInfo,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group whose data block will be modified.

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that describes the group data to set. To change the values of the **pwszName** or **pvData** members, you must set the corresponding DPNINFO_NAME or DPNINFO_DATA flags in the **dwInfoFlags** member.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method:

DPNSETGROUPINFO_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDGROUP

Remarks

Transmission of nonstatic information should be handled with the **IDirectPlay8Peer::SendTo** method because of the high cost of using the **IDirectPlay8Peer::SetGroupInfo** method.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SetPeerInfo

Sets the static settings of the local peer. Call this method before connecting to relay basic player information with the application. Once the peer successfully connects with the application, information set through this method can be retrieved by other players by calling the **IDirectPlay8Peer::GetPeerInfo** method.

HRESULT SetPeerInfo(
const DPN_PLAYER_INFO *const *pdpnPlayerInfo*,
PVOID const *pvAsyncContext*,
DPNHANDLE *const *phAsyncHandle*,
const DWORD *dwFlags*
);

Parameters

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure that contains the peer information to set.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNSETPEERINFO_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and normally returns DPNSUCCESS_PENDING. It can also return one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_NOCONNECTION

Remarks

This method can be called at any time during the session.

Transmission of nonstatic information should be handled with the **IDirectPlay8Peer::SendTo** method because of the high cost of using the **IDirectPlay8Peer::SetPeerInfo** method.

You can modify the peer information with this method after connecting to the application. Calling this method after connection generates a **DPN_MSGID_PEER_INFO** system message to all players, informing them that data has been updated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::SetSPCaps

Sets the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT SetSPCaps(
const GUID *const pguidSP,
const DPN_SP_CAPS *const pdpSPCaps
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to set information about.

pdpSPCaps
Pointer to a **DPN_SP_CAPS** structure to set the information about the specified service provider.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called. Currently only the *dwNumThreads* field can be set by this call, the *dwFlags* field must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Peer::TerminateSession

Terminates the current Microsoft® DirectPlay® session.

```
HRESULT TerminateSession(
void *const pvTerminateData,
const DWORD dwTerminateDataSize,
const DWORD dwFlags
);
```

Parameters

pvTerminateData
Pointer to termination data. This data is also sent in the **pvTerminateData** member of the **DPN_MSGID_TERMINATE_SESSION** system message.

dwTerminateDataSize
Size of data contained in the *pvTerminateData* parameter.

dwFlags
Reserved. Must be 0.

Remarks

This method may be called only by the host player.

When this method is called, the **DPN_MSGID_TERMINATE_SESSION** will be sent to the message handler of each player in the session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client

Applications use the methods of the **IDirectPlay8Client** interface to create and manage client applications for client/server sessions.

The methods of the **IDirectPlay8Client** interface can be organized into the following groups.

Session Management	**Close**
	Connect
	EnumHosts
	EnumServiceProviders
	GetApplicationDesc
	GetCaps
	GetSPCaps
	SetCaps
	SetSPCaps
Message Management	**GetSendQueueInfo**
	Initialize
	ReturnBuffer
	Send
Client Information	**SetClientInfo**
Server Information	**GetServerInfo**
Miscellaneous	**CancelAsyncOperation**
	RegisterLobby
	GetConnectionInfo
	GetServerAddress

IDirectPlay8Client::CancelAsyncOperation

Cancels asynchronous requests. Many methods of the **IDirectPlay8Client** interface run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this interface that can be run asynchronously return a *hAsyncHandle* parameter.

Specific requests are canceled by passing the *hAsyncHandle* of the request in this method's *hAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, specifying NULL in the *hAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *dwFlags* parameter. If a specific handle is provided to this method, no flags should be set.

```
HRESULT CancelAsyncOperation(
const DPNHANDLE hAsyncHandle,
const DWORD dwFlags
);
```

Parameters

hAsyncHandle

Handle of the asynchronous operation to stop. You receive this handle when you call one of several methods that support asynchronous operations. This value can be set to NULL to stop all requests or a particular type of asynchronous request. If a particular handle is specified, the *dwFlags* parameter must be 0.

dwFlags

Flag that specifies which asynchronous request to canceled. You can set one of the following flags.

DPNCANCEL_ENUM

Cancel all asynchronous **IDirectPlay8Client::EnumHosts** requests. A single **EnumHosts** request can be canceled by specifying the handle returned from the **EnumHosts** method.

DPNCANCEL_CONNECT

Cancel an asynchronous **IDirectPlay8Client::Connect** request.

DPNCANCEL_SEND

Cancel an asynchronous **IDirectPlay8Client::Send** request.

DPNCANCEL_ALL_OPERATIONS

Cancel all asynchronous requests.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_CANNOTCANCEL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHANDLE

DPNSUCCESS_PENDING

Remarks

You can use this method to cancel an asynchronous operation for the **IDirectPlay8Client::Connect**, **IDirectPlay8Client::Send**, and **IDirectPlay8Client::EnumHosts** methods. Microsoft® DirectPlay® 8.0 does not support cancellation of other asynchronous operations.

You can cancel a send by providing the handle returned from **IDirectPlay8Client::Send** method. A **DPN_MSGID_SEND_COMPLETE** system message is still posted to the applications message handler for each asynchronous send that is sent without the DPNSEND_NOCOMPLETE flag set. Sends that are canceled by this method return DPNERR_USERCANCEL in their **hResultCode** member of the **DPN_MSGID_SEND_COMPLETE** message.

If you set the DPNCANCEL_ALL_OPERATIONS, DPN_CANCELCONNECT, DPN_CANCELSEND, or DPNCANCEL_ENUM flags in *dwFlags*, DirectPlay will attempt to cancel all matching operations. This method will return an error if any attempted cancellation fails, even though some cancellations may have been successful.

Note

The completion message might not arrive until after this method returns. Do not assume that the operation has been terminated until you have received a **DPN_MSGID_SEND_COMPLETE**, **DPN_MSGID_CONNECT_COMPLETE**, or **DPN_MSGID_ASYNC_OP_COMPLETE** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::Close

Closes the open connection with a session. This method must be called on any object that is successfully initialized with a call to the **IDirectPlay8Client::Initialize** method.

```
HRESULT Close(
const DWORD dwFlags
);
```

Parameters

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_UNINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::Connect

Establishes the connection to the server. After a connection is established, the communication channel on the interface is open and the application should expect messages to arrive immediately. No messages can be sent by means of the **IDirectPlay8Client::Send** method until the connection has completed.

Before this method is called, you can obtain an application description by calling **IDirectPlay8Client::EnumHosts**. The **EnumHosts** method returns a **DPN_APPLICATION_DESC** structure for each hosted application. The structure describes the application, including the GUID of the application.

If this method is called asynchronously (by default), when the connection completes a **DPN_MSGID_CONNECT_COMPLETE** message is sent to the application's message handler.

```
HRESULT Connect(
const DPN_APPLICATION_DESC *const pdnAppDesc,
IDirectPlay8Address *const pHostAddr,
IDirectPlay8Address *const pDeviceInfo,
const DPN_SECURITY_DESC *const pdnSecurity,
const DPN_SECURITY_CREDENTIALS *const pdnCredentials,
const void *const pvUserConnectData,
const DWORD dwUserConnectDataSize,
void *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdnAppDesc
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application. The only member of this structure that you must set is the **guidApplication** member. Only some of the members of this structure are used by this method. The only member that you must set is **guidApplication**. You can also set **guidInstance**, **pwszPassword**, **dwFlags**, and **dwSize**.

pHostAddr
Pointer to an **IDirectPlay8Address** interface that specifies the addressing information to use to connect to the computer that is hosting.

pDeviceInfo

Pointer to an **IDirectPlay8Address** object that specifies what network adapter (for example, NIC, modem, and so on) to use to connect to the server.

pdnSecurity
Reserved. Must be NULL.

pdnCredentials
Reserved. Must be NULL.

pvUserConnectData
Pointer to application-specific data provided to the host or server to further validate the connection. This data is sent to the **DPN_MSGID_INDICATE_CONNECT** message in the **pvUserConnectData** member. This parameter is optional and you may pass NULL to bypass the connection validation provided by the user code.

dwUserConnectDataSize
Variable of type **DWORD** that specifies the size of the data contained in *pvUserConnectData*.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_CONNECT_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle
A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Client::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNCONNECT_SYNC flag in *dwFlags.*

dwFlags
Flag that describes the connection mode. You can set the following flag.

DPNCONNECT_OKTOQUERYFORADDRESSING
Setting this flag will display a standard Microsoft® DirectPlay® dialog box, which queries the user for more information if not enough information is passed in this method.

DPNCONNECT_SYNC
Process the connection request synchronously. Setting this flag does not generate a **DPN_MSGID_CONNECT_COMPLETE** system message.

Return Values

Returns S_OK if this method is processed synchronously and is successful. If the request is processed asynchronously, S_OK will be returned if the method is instantly processed. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_HOSTREJECTEDCONNECTION

DPNERR_INVALIDAPPLICATION

DPNERR_INVALIDDEVICEADDRESS

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHOSTADDRESS
DPNERR_INVALIDINSTANCE
DPNERR_INVALIDINTERFACE
DPNERR_INVALIDPASSWORD
DPNERR_NOCONNECTION
DPNERR_NOTHOST
DPNERR_SESSIONFULL
DPNERR_ALREADYCONNECTED

Remarks

Although multiple enumerations can be run concurrently, and can be run across the duration of a connection, only one connection is allowed per interface. To establish a connection to more than one application, you must create another interface.

When this method is called, a **DPN_MSGID_INDICATE_CONNECT** message is posted to the server's message handler. On retrieval of this message, the host can pass back connection reply data to the **Connect** method. Connection reply data can send a message indicating that the host does not approve the connection. The calling application can then handle this reply appropriately.

The hResultCode on the completion will indicate S_OK if the Connect() attempt was successful, or an error otherwise. If the Host player returned anything other than S_OK from the **DPN_MSGID_INDICATE_CONNECT** message, the likely error code in the completion will be DPNERR_HOSTREJECTEDCONNECTION.

When the connection completes, a **DPN_MSGID_CONNECT_COMPLETE** message is sent to the application's message handler.

To close the connection established with this method, call the **IDirectPlay8Client::Close** method.

Note

If you set the DPNCONNECT_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::EnumHosts

Enumerates applications that host Microsoft® DirectPlay® games. When an application is found that meets the enumeration criteria, the application's message handler is called with a **DPN_MSGID_ENUM_HOSTS_RESPONSE** system message. The message contains a **DPN_APPLICATION_DESC** structure describing the applications found.

Any number of enumerations can be run concurrently. The *pvUserContext* value is provided in the message handler to help differentiate replies to different enumerations.

Because of the variation in the number of ways enumeration can happen, it is not recommended that an application attempt to specify *dwEnumPeriod, dwRetryInterval,* or *dwTimeOut* unless the application has some specific media knowledge.

The default enumeration count and timeout values will cause **EnumHosts** to complete within a reasonable amount of time. These values are set by the service provider, and can be obtained by calling **IDirectPlay8Client::GetSPCaps**. Asynchronous enumerations can be stopped at any time by calling **IDirectPlay8Client::CancelAsyncOperation** and either passing the handle returned in the *pAsyncHandle* parameter or setting the DPENUM_CANCEL flag in the *dwFlags* parameter. An enumeration can also be stopped by returning anything other than S_OK from the message handler.

HRESULT EnumHosts(
PDPN_APPLICATION_DESC const *pApplicationDesc*,
IDirectPlay8Address *const *pdpaddrHost*,
IDirectPlay8Address *const *pdpaddrDeviceInfo*,
PVOID const *pvUserEnumData*,
const DWORD *dwUserEnumDataSize*,
const DWORD *dwEnumCount*,
const DWORD *dwRetryInterval*,
const DWORD *dwTimeOut*,
PVOID const *pvUserContext*,
HANDLE *const *pAsyncHandle,*
const DWORD *dwFlags*
);

Parameters

pApplicationDesc

Pointer to a **DPN_APPLICATION_DESC** structure that specifies which application hosts to enumerate. You must set the *pApplicationDesc*.dwSize member to the appropriate value. To reduce the number of responses, set *pApplicationDesc*.guidApplication to the GUID of the application to be found. If this member is not set, the search will include all applications.

pdpaddrHost

Pointer to an **IDirectPlay8Address** object that specifies the address of the computer that is hosting the application. If you specify NULL. DirectPlay will create an address. By default, DirectPlay will create the address from the *pdpaddrDeviceInfo* parameter. If you set the DPNENUMHOSTS_OKTOQUERYFORADDRESSING flag in *dwFlags*, the user can be queried for address information.

pdpaddrDeviceInfo

Pointer to an **IDirectPlay8Address** object that specifies the service provider and local device settings to use when enumerating.

pvUserEnumData

Pointer to a block of data that is sent in the enumeration request to the host. The size of the data is limited depending on the network type. Call **IDirectPlay8Client::GetSPCaps** to obtain the exact value.

dwUserEnumDataSize

Variable of type **DWORD** that specifies the size of the data pointed at in the *pvUserEnumData* parameter.

dwEnumCount

Value specifying how many times the enumeration data will be sent. Set this parameter to zero to use the default value. You can obtain the default value for *dwEnumCount* by calling **IDirectPlay8Client::GetSPCaps**. If *dwEnumCount* is set to INFINITE, the enumeration will continue until canceled.

dwRetryInterval

Value specifying how many milliseconds between enumeration retries. Set this parameter to zero to use the default value. You can obtain the default value for *dwRetryInterval* by calling **IDirectPlay8Client::GetSPCaps**.

dwTimeOut

Variable of type **DWORD** that specifies the number of milliseconds that DirectPlay will wait for replies after the last enumeration is sent. Set this parameter to zero to use the default value. You can obtain the default value for *dwTimeOut* by calling **IDirectPlay8Client::GetSPCaps** If INFINITE is specified, the enumeration continues until it is canceled.

pvUserContext

Context that is provided in the client's message handler when it is called with responses to the enumeration. This can be useful to differentiate replies from concurrent enumerations.

pAsyncHandle

A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Client::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNENUMHOSTS_SYNC flag in *dwFlags.*

dwFlags

The following flags can be set.

DPNENUMHOSTS_SYNC

Causes the method to process synchronously.

DPNENUMHOSTS_OKTOQUERYFORADDRESSING

Setting this flag will display a standard DirectPlay dialog box, which queries the user for more information if not enough information is passed in this method.

DPNENUMHOSTS_NOBROADCASTFALLBACK
If the service provider supports broadcasting, setting this flag will disable the broadcast capabilities. Check to see if broadcasting is supported by examining the **DPN_SP_CAPS** structure before setting this flag.

Return Values

Returns S_OK if this method is processed synchronously and is successful. If the request is processed asynchronously, S_OK can return if the method is instantly processed. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDDEVICEADDRESS

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHOSTADDRESS

DPNERR_INVALIDPARAM

DPNERR_ENUMQUERYTOOLARGE

Remarks

If you set the DPNENUMHOSTS_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::EnumServiceProviders

Enumerates the registered service providers available to the application.

HRESULT EnumServiceProviders(
const GUID *const *pguidServiceProvider*,
const GUID *const *pguidApplication*,
DPN_SERVICE_PROVIDER_INFO *const *pSPInfoBuffer*,
PDWORD const *pcbEnumData*,
PDWORD const *pcReturned*,
const DWORD *dwFlags*

);

Parameters

pguidServiceProvider
Pointer to a variable of type **GUID** that specifies a service provider. This optional parameter forces the enumeration of subdevices for the specified service provider. You should normally set this value to NULL, to enumerate all available service providers.

pguidApplication
Pointer to a variable of type **GUID** that specifies an application. If a pointer is passed in this parameter, only service providers who can be connected to the application are enumerated. You can also pass NULL to enumerate the registered service providers for the system.

pSPInfoBuffer
Pointer to an array of **DPN_SERVICE_PROVIDER_INFO** structures that will be filled with service provider information.

pcbEnumData
Pointer to **DWORD**, which is filled with the size of the *pEnumData* buffer if the buffer is too small.

pcReturned
Pointer to a variable of type **DWORD** that specifies the number of **DPN_SERVICE_PROVIDER_INFO** structures returned in the *pEnumData* array.

dwFlags
The following flag can be specified.

DPNENUMSERVICEPROVIDERS_ALL
Enumerates all the registered service providers for the system, including those that are not available to the application or do not have devices installed.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

Remarks

Call this method initially by specifying NULL in the *pguidServiceProvider* parameter to determine the base service providers available to the system. Specific devices for a service provider can be obtained by passing a pointer to a service provider GUID in the *pguidServiceProvider*. This is useful, for example, when using the Modem Connection for Microsoft® DirectPlay® service provider. You can choose among different modems for dialing out and select specific modems for hosting.

If the *pEnumData* buffer is not big enough to hold the requested service provider information, the method returns DPNERR_BUFFERTOOSMALL and the *cbEnumData* parameter contains the required buffer size. Typically, the best strategy is to call the method once with a zero-length buffer to determine the required size. Then call the method again with the appropriate-sized buffer.

Normally, this method will return only those service providers that can be used by the application. For example, if the IPX networking protocol is not installed, DirectPlay will not return the IPX service provider. To have DirectPlay return all service providers, even those that cannot be used by the application, set the DPNENUMSERVICEPROVIDERS_ALL flag in *dwFlags*.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetApplicationDesc

Retrieves the full application description for the connected application.

HRESULT GetApplicationDesc(
DPN_APPLICATION_DESC *const *pAppDescBuffer*,
DWORD *const *pcbDataSize*,
const DWORD *dwFlags*
);

Parameters

pAppDescBuffer
Pointer to a **DPN_APPLICATION_DESC** structure where the application description data is written. Set this parameter to NULL to request only the size of data. If *pAppDescBuffer* is not set to NULL, you must set the *pAppDescBuffer*.dwSize member to an appropriate value. The *pcbDataSize* parameter is set to the size required to hold the data.

pcbDataSize
Pointer to a variable of type **DWORD** that is initialized to the size of the buffer before calling this method. After the method returns, this parameter is set to the size, in bytes, of the session data. If the buffer is too small, this method returns the DPNERR_BUFFERTOOSMALL error value, and this parameter is set to the buffer size required. If this parameter is NULL, the method returns DPNERR_INVALIDPARAM.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM
DPNERR_NOCONNECTION

Remarks

Call this method initially by passing NULL in the *pvData* parameter to obtain the size of the required buffer. When you call the method a second time to fill the buffer, be sure to set the structures **dwSize** member to the appropriate value.

The returned DPN_APPLICATION_DESC structure will have the **guidInstance**, **guidApplication**, and **pwszSessionName** members set. It will not contain information about other clients that are connected to the session. That information, if available, can be obtained only from the server application. In particular, the **dwCurrentPlayers** member will always be set to 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetCaps

Retrieves the **DPN_CAPS** structure for the current interface.

```
HRESULT GetCaps(
DPN_CAPS *const pdpCaps,
const DWORD dwFlags
);
```

Parameters

pdpnCaps
Pointer to a **DPN_CAPS** structure to receive caps information. You must set the **dwSize** member of this structure to an appropriate value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetConnectionInfo

Retrieves statistical information about the connection between the local client and the server.

```
HRESULT GetConnectionInfo(
DPN_CONNECTION_INFO *const pdnConnectInfo,
const DWORD dwFlags
);
```

Parameters

pdnConnectInfo
Pointer to a **DPN_CONNECTION_INFO** structure to retrieve information about the specified connection. You must set the **dwSize** member of this structure to an appropriate value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

This method can be called only after a successful **Connect** call has completed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetSendQueueInfo

Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® does not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should decrease the rate that messages are sent.

HRESULT GetSendQueueInfo(
DWORD *const *pdwNumMsgs*,
DWORD *const *pdwNumBytes*,
const DWORD *dwFlags*
);

Parameters

pdwNumMsgs
Pointer to a variable of type **DWORD** that contains the number of messages currently queued. This value is optional, and may be set to NULL.

pdwNumBytes
Pointer to a variable of type **DWORD** that specifies the total number of bytes of data of the messages currently queued. This value is optional, and may be set to NULL.

dwFlags
You may specify the DPNGETSENDQUEUEINFO_PRIORITY_NORMAL, DPNGETSENDQUEUEINFO_PRIORITY_HIGH, or DPNGETSENDQUEUEINFO_PRIORITY_LOW flag to inquire about specific messages of that priority.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Remarks

You cannot set both *pdwNumMsgs* and *pdwNumBytes* to NULL. At least one of them must be set to a valid pointer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetServerAddress

Retrieves the address of the server for the session.

```
HRESULT GetServerAddress(
IDirectPlay8Address **const ppAddress,
const DWORD dwFlags
);
```

Parameters

ppAddress
Address of a pointer to an **IDirectPlay8Address** object that specifies the address of the server. You must release this object when you no longer need it.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetServerInfo

Retrieves the data set for the server set by the call to the **IDirectPlay8Server::SetServerInfo** method.

```
HRESULT GetServerInfo(
DPN_PLAYER_INFO *const pdpnPlayerInfo,
DWORD *const pdwSize,
const DWORD dwFlags
);
```

Parameters

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure to be filled with the server's information. If *pdwSize* is not set to NULL, you must set *pdpnPlayerInfo*.dwSize to the size of a **DPN_PLAYER_INFO** structure.

pdwSize
Pointer to a variable of type **DWORD** that contains the size of the data returned in the *pdpnPlayerInfo* parameter. If this value is too small, the method returns DPNERR_BUFFERTOOSMALL, and this parameter is set to the required size of the buffer.

dwFlags
Reserved. Must be set to 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

Remarks

Call this method after the client receives a DPN_MSGID_SERVER_INFO message, indicating that the server has updated its information.

Microsoft® DirectPlay® returns the **DPN_PLAYER_INFO** structure and the pointers assigned to the structure's **pwszName** and **pvData** members in a contiguous buffer. If the two pointers were set, you must have allocated enough memory for the structure, plus the two pointers. The most robust way to use this method is to first call it with *pdwSize* set to NULL. When the method returns, *pdwSize* will point to the correct value. Use that value to allocate memory for your structure and call the method a second time to retrieve the information.

When the method returns, the **dwInfoFlags** member of the **DPN_PLAYER_INFO** structure will always have the DPNINFO_DATA and DPNINFO_NAME flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Server::SetServerInfo**, to notify DirectPlay of which values have changed.

Transmission of nonstatic information should be handled with the **IDirectPlay8Server::SendTo** method because of the high cost of using the **IDirectPlay8Server::SetServerInfo** method.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::GetSPCaps

Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT GetSPCaps(
const GUID *const pguidSP,
DPN_SP_CAPS *const pdpnSPCaps,
const DWORD dwFlags
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to get information about.

pdpnSPCaps
Pointer to a **DPN_SP_CAPS** structure to receive the information about the specified service provider. You must set the *pdpnSPCaps*.dwSize member to the size of a **DPN_SP_CAPS** structure.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

This method retrieves information about the specified service provider. A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::Initialize

Registers an entry point in the client's code that receives the messages from the **IDirectPlay8Client** interface and from the server. This method must be called before calling any other methods of this interface.

HRESULT Initialize(
PVOID const *pvUserContext*,
const PFNDPNMESSAGEHANDLER *pfn*,
const DWORD *dwFlags*
);

Parameters

pvUserContext
Pointer to the user-provided context value in calls to the message handler. Providing a user-context value can be useful to differentiate messages coming from multiple interfaces to a common message handler.

pfn
Pointer to a **PFNDPNMESSAGEHANDLER** callback function that receives all messages from the server, and receives indications of session changes from the **IDirectPlay8Client** interface.

dwFlags
You may specify the following flag.

DPNINITIALIZE_DISABLEPARAMVAL
Disable parameter validation for the current object.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

This is the first method you should call after using **CoCreateInstance** to obtain the **IDirectPlay8Client** interface.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::RegisterLobby

Allows launched applications to automatically propagate game status to the lobby.

HRESULT RegisterLobby(
const DPNHANDLE *dpnHandle*,
IDirectPlay8LobbiedApplication *const *pIDP8LobbiedApplication*,
const DWORD *dwFlags*

);

Parameters

dpnHandle
Connection handle used when making the calls to **IDirectPlay8LobbiedApplication::UpdateStatus**.

pIDP8LobbiedApplication
Pointer to the **IDirectPlay8LobbiedApplication** object that specifies the application.

dwFlags
One of the following flags:

DPNLOBBY_REGISTER
Registers the lobby with the application.

DPNLOBBY_UNREGISTER
Unregisters the lobby with the application.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::ReturnBuffer

Retrieves message buffers provided to the application through the **pReceiveData** member of the **DPN_MSGID_RECEIVE** system message. If the user's message handler returns DPNSUCCESS_PENDING to the RECEIVE callback, Microsoft® DirectPlay® assumes ownership of the buffer has been transferred to the application, and neither frees nor modifies it until ownership is returned to DirectPlay through this call.

HRESULT ReturnBuffer(
const DPNHANDLE *hBufferHandle*,
const DWORD *dwFlags*
);

Parameters

hBufferHandle

Variable of type **DPNHANDLE** that specifies the buffer handle for the message. This is obtained in the **hBufferHandle** member of the **DPN_MSGID_RECEIVE** system message.

dwFlags

Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDHANDLE

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::Send

Transmits data to the server. The message can be sent synchronously or asynchronously.

```
HRESULT Send(
const DPN_BUFFER_DESC *const pBufferDesc,
const DWORD cBufferDesc,
const DWORD dwTimeOut,
void *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pBufferDesc

Pointer to a **DPN_BUFFER_DESC** structure that describes the data to send.

cBufferDesc

Number of **DPN_BUFFER_DESC** structures pointed to by *pBufferDesc*. There can only be one buffer in this version of Microsoft® DirectPlay®.

dwTimeOut

Number of milliseconds to wait for the message to be sent. If the message has not been sent by the *dwTimeOut* value, the message is not sent. If you do not want a time out for message sends, set this parameter to 0.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_SEND_COMPLETE** system message.

phAsyncHandle

A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Client::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNSEND_SYNC flag in *dwFlags.*

dwFlags

Flags that describe send behavior. You can set one or more of the following flags.

DPNSEND_SYNC

Process the **Send** request synchronously.

DPNSEND_NOCOPY

Use the data in the **DPN_BUFFER_DESC** structure and do not make an internal copy. This may be a more efficient method of sending data to the server. However, it is less robust, because the sender might be able to modify the message before the receiver has processed it. This flag cannot be combined with DPNSEND_NOCOMPLETE.

DPNSEND_NOCOMPLETE

Does not send **DPN_MSGID_SEND_COMPLETE** to the message handler. This flag may not be used with DPNSEND_NOCOPY or DPNSEND_GUARANTEED. Additionally, when using this flag *pvAsyncContext* must be NULL.

DPNSEND_COMPLETEONPROCESS

Send **DPN_MSGID_SEND_COMPLETE** to the message handler when this message has been delivered to the target and the target's message handler returns from indicating its reception. There is additional internal message overhead when this flag is set, and the message transmission process may become significantly slower. If you set this flag, DPNSEND_GUARANTEED must also be set.

DPNSEND_GUARANTEED

Send the message by a guaranteed method of delivery.

DPNSEND_PRIORITY_HIGH

Sets the priority of the message to high. This flag cannot be used with DPNSEND_PRIORITY_LOW.

DPNSEND_PRIORITY_LOW

Sets the priority of the message to low. This flag cannot be used with DPNSEND_PRIORITY_HIGH.

DPNSEND_NOLOOPBACK

Suppress the **DPN_MSGID_RECEIVE** system message to your message handler if you are sending to yourself.

DPNSEND_NONSEQUENTIAL

If the flag is not set, messages are delivered to the target application in the order that they are sent, which may necessitate buffering out of sequence messages until the missing messages arrive. Messages are simply delivered to the target application in the order that they are received.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_TIMEDOUT

Remarks

This method generates a **DPN_MSGID_RECEIVE** system message in the server's message handler. The data buffer is contained in the **pReceiveData** member of the associated structure.

Messages can have one of three priorities: low, normal, and high. To specify a low or high priority for the message, set the appropriate flag in *dwFlags.* If neither of the priority flags is set, the message will have normal priority. See Basic Networking for a discussion of send priorities.

When the **Send** request is completed, a **DPN_MSGID_SEND_COMPLETE** system message is posted to the sender's message handler. The success or failure of the request is contained in the **hResultCode** member of the associate structure. You can suppress the send completion by setting the DPN_NOCOMPLETE flag in *dwflags*.

Send completions are typically posted on the source computer as soon as the message is sent. In other words, a send completion does not necessarily mean that the message has been processed on the target. It may still be in a queue. If you want to be certain that the message has been processed by the target, set the DPN_COMPLETEONPROCESS flag in *dwFlags.* This flag ensures that the send completion will not be sent until the target's message handler has processed the message, and returned.

Note

Do not assume that resources such as the data buffer will remain valid until the method has returned. If you call this method asynchronously, the **DPN_MSGID_SEND_COMPLETE** message may be received and processed by your message handler before the call has returned. If your message handler deallocates or otherwise invalidates a resource such as the data buffer, that resource may become invalid at any time after the method has been called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::SetCaps

Sets the **DPN_CAPS** structure for the current interface.

```
HRESULT SetCaps(
const DPN_CAPS *const pdpCaps,
const DWORD dwFlags
);
```

Parameters

pdpCaps
Pointer to a **DPN_CAPS** structure used to set the information about the current interface.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPOINTER

DPNERR_INVALIDPARAM

DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::SetClientInfo

Sets the static settings of a client with an application. Call this method before connecting to relay basic player information to the application. Once the client successfully connects with the application, the server can retrieve information obtained through this method by calling the **IDirectPlay8Server::GetClientInfo** method.

```
HRESULT SetClientInfo(
const DPN_PLAYER_INFO *const pdpnPlayerInfo,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure that contains the client information to set.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNSETCLIENTINFO_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. If the request is processed asynchronously, S_OK can return if the method is instantly processed. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_NOCONNECTION

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

This method can be called at any time during the session.

Transmission of nonstatic information should be handled with the **IDirectPlay8Client::Send** method because of the high cost of using the **IDirectPlay8Client::SetClientInfo** method.

You can modify the client information with this method after connecting to the application. Calling this method after connection generates a **DPN_MSGID_CLIENT_INFO** system message to all players, informing them that data has been updated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Client::SetSPCaps

Sets the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT SetSPCaps(
const GUID *const pguidSP,
constDPN_SP_CAPS *const pdpnSPCaps
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to set information about.

pdpnSPCaps
Pointer to a **DPN_SP_CAPS** structure to set the information about the specified service provider.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called. Currently only the *dwNumThreads* member can be set by this call; *dwFlags* must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server

Applications use the methods of the **IDirectPlay8Server** interface to create and manage the server for a Microsoft® DirectPlay® client/server transport session

The methods of the **IDirectPlay8Server** interface can be organized into the following groups.

Session Management **Close**

	EnumServiceProviders
	GetApplicationDesc
	GetCaps
	GetSPCaps
	GetSendQueueInfo
	Host
	Initialize
	ReturnBuffer
	SendTo
	SetApplicationDesc
	SetCaps
	SetServerInfo
	SetSPCaps
Client Management	**DestroyClient**
	GetClientInfo
	GetPlayerContext
Group Management	**AddPlayerToGroup**
	CreateGroup
	DestroyGroup
	EnumPlayersAndGroups
	EnumGroupMembers
	GetGroupContext
	GetGroupInfo
	RemovePlayerFromGroup
	SetGroupInfo
Miscellaneous	**CancelAsyncOperation**
	GetClientAddress
	GetConnectionInfo
	GetLocalHostAddresses
	RegisterLobby

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::AddPlayerToGroup

Adds a client to a group. After the client is successfully added to the group, all messages sent to the group are sent to the client.

```
HRESULT AddPlayerToGroup(
const DPNID idGroup,
const DPNID idClient,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

idGroup

Variable of type **DPNID** that specifies the identifier of the group to add the client to.

idClient

Variable of type **DPNID** that specifies the identifier of the client to add to the group.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message. This parameter is optional and can be set to NULL.

phAsyncHandle

A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags

Flag that controls how this method is processed. The following flag can be set for this method.

DPNADDPLAYERTOGROUP_SYNC

Causes this method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

DPNERR_INVALIDPLAYER

Remarks

The server can add itself or a client to an existing group. After a player is successfully added to a group, all messages sent to the group will be received by the player.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::CancelAsyncOperation

Cancels asynchronous requests. Many methods of the **IDirectPlay8Server** interface run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this interface that can be run asynchronously return an *hAsyncHandle* parameter.

Specific requests are canceled by passing the *hAsyncHandle* of the request in this method's *hAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, specifying NULL in the *hAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *dwFlags* parameter. If a specific handle is provided to this method, no flags should be set.

```
HRESULT CancelAsyncOperation(
const DPNHANDLE hAsyncHandle,
const DWORD dwFlags
);
```

Parameters

hAsyncHandle
Handle of the asynchronous operation to stop. This value can be NULL to stop all requests or a particular type of asynchronous request. If a specific handle for the request to be canceled is specified, the *dwFlags* parameter must be 0. You will receive this handle when you call one of several methods that support asynchronous operations.

dwFlags
Flag that specifies which asynchronous request to cancel. You can set one of the following flags.

DPNCANCEL_SEND
Cancel an asynchronous **IDirectPlay8Server::SendTo** request.

DPNCANCEL_ALL_OPERATIONS
Cancel all asynchronous requests.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_CANNOTCANCEL
DPNERR_INVALIDFLAGS
DPNERR_INVALIDHANDLE
DPNSUCCESS_PENDING

Remarks

You can use this method to cancel an asynchronous operation for the **IDirectPlay8Server::SendTo** method. Microsoft® DirectPlay® 8.0 does not support cancellation of other asynchronous operations.

You can cancel a **Send** request by providing the handle returned from **IDirectPlay8Server::SendTo** method. A **DPN_MSGID_SEND_COMPLETE** system message is still posted to the applications message handler for each asynchronous **Send** request that is sent without the DPNSEND_NOCOMPLETE flag set. Send requests that are canceled by this method return DPNERR_USERCANCEL in their **hResultCode** member of the **DPN_MSGID_SEND_COMPLETE** message.

If you set the DPNCANCEL_ALL_OPERATIONS or DPNCANCEL_SEND flags in *dwFlags*, DirectPlay will attempt to cancel all matching operations. This method will return an error if any attempted cancellation fails, even though some cancellations may have been successful.

Note

The completion message might not arrive until after this method returns. Do not assume that the operation has been terminated until you have received a **DPN_MSGID_SEND_COMPLETE**, **DPN_MSGID_CONNECT_COMPLETE**, or **DPN_MSGID_ASYNC_OP_COMPLETE** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::Close

Closes the open connection with a session.

```
HRESULT Close(
const DWORD dwFlags
);
```

Parameters

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_UNINITIALIZED

Remarks

This method must be called on any object successfully initialized with **IDirectPlay8Server::Initialize**.

This method is a counterpart to **IDirectPlay8Server::Host**. It closes all active network connections hosted by the server. This method is synchronous, and will not return until the server has processed all DPN_MSGID_DESTROY_PLAYER messages. This feature guarantees that when **Close** returns, you can safely shut down the server application.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::CreateGroup

Creates a group in the current session. When this method is called, the server's message handler receives a DPN_MSGID_CREATE_GROUP message.

Note

Multicasting is not supported for this release.

```
HRESULT CreateGroup(
const DPN_GROUP_INFO *const pdpnGroupInfo,
VOID *const pvGroupContext,
VOID *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that contains the group description.

pvGroupContext

Pointer to the context value for the group. This value is preset when the local application's message handler processes the DPN_MSGID_CREATE_GROUP message. This parameter is optional and may be set to NULL.

pvAsyncContext

Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle

A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags

Flag that controls how this method is processed. The following flag can be set for this method.

DPNCREATEGROUP_SYNC

Causes this method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or the following error value.

DPNERR_INVALIDFLAGS

Remarks

Microsoft® DirectPlay® does not maintain hierarchical groups because these can easily be implemented with flat groups and expeditious use of the group data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::DestroyClient

Deletes a client from the session.

```
HRESULT DestroyClient(
const DPNID dpnidClient,
const VOID *const pDestroyInfo,
const DWORD dwDestroyInfoSize,
const DWORD dwFlags
);
```

Parameters

dpnidClient

Variable of type **DPNID** that specifies the identifier of the client to delete.

pDestroyInfo
Pointer that describes additional delete data information.

dwDestroyInfoSize
Variable of type **DWORD** that specifies the size of the data in the *pDestroyInfo* parameter.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDPLAYER
DPNERR_NOTHOST

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::DestroyGroup

Deletes a group created by the **IDirectPlay8Server::CreateGroup** method.

```
HRESULT DestroyGroup(
const DPNID idGroup,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

idGroup
DPNID of the group to delete.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message. This parameter is optional and may be set to NULL.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags

Flag that controls how this method is processed. The following flag can be set for this method.

DPNDESTROYGROUP_SYNC
Causes the method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::EnumPlayersAndGroups

Retrieves a list of all the player and/or group identifiers for the application.

```
HRESULT EnumPlayersAndGroups(
DPNID *const prgdpnid,
DWORD *const pcdpnid,
const DWORD dwFlags
);
```

Parameters

prgdpnid
Pointer to an array that will be filled with the session's group and/or player identifiers.

pcdpnid
Pointer to a variable of type **DWORD** that specifies the number of identifiers in the *prgdpnid* parameter. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the number of entries that are required.

dwFlags
Flag that describes enumeration behavior. You can set one or both of the following flags.

DPNENUM_PLAYERS
Return a list of player identifiers.

DPNENUM_GROUPS
Return a list of group identifiers.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDFLAGS

Remarks

Because group and player information changes frequently, the required buffer size returned may change between subsequent calls. Check and reallocate the buffer until the method succeeds.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::EnumGroupMembers

Retrieves a list of all players in a group.

```
HRESULT EnumGroupMembers(
const DPNID dpnid,
DPNID *const prgdpnid,
DWORD *const pcdpnid,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the group that contains the players to enumerate.

prgdpnid
Pointer to an array that contains the identifiers of the group's players.

pcdpnid
Pointer to a variable of type **DWORD** that contains the number of player identifiers in the *prgdpnid* parameter. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter is set to the number of entries that are required.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDFLAGS
DPNERR_INVALIDGROUP

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::EnumServiceProviders

Enumerates the registered service providers available to the application.

HRESULT EnumServiceProviders(
const GUID *const *pguidServiceProvider*,
const GUID *const *pguidApplication*,
DPN_SERVICE_PROVIDER_INFO *const *pSPInfoBuffer*,
PDWORD const *pcbEnumData*,
PDWORD const *pcReturned*
const DWORD *dwFlags*
);

Parameters

pguidServiceProvider
Pointer to a variable of type GUID that specifies a service provider. This optional parameter forces the enumeration of subdevices for the specified service provider. You should normally set this value to NULL, to enumerate all available service providers.

pguidApplication
Pointer to a variable of type GUID that specifies an application. If a pointer is passed in this parameter, only service providers who can be connected to the application are enumerated. You can also pass NULL to enumerate the registered service providers for the system.

pSPInfoBuffer
Pointer to an array of **DPN_SERVICE_PROVIDER_INFO** structures that will be filled with service provider information.

pcbEnumData
Pointer to **DWORD**, which is filled with the size of the *pEnumData* buffer if the buffer is too small.

pcReturned

Pointer to a variable of type **DWORD** that specifies the number of **DPN_SERVICE_PROVIDER_INFO** structures returned in the *pEnumData* array.

dwFlags

The following flag can be specified.

DPNENUMSERVICEPROVIDERS_ALL

Enumerates all the registered service providers for the system including those that are not available to the application or do not have devices installed.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

Remarks

Call this method initially by specifying NULL in the *pguidServiceProvider* parameter to determine the base service providers available to the system. Specific devices for a service provider can be obtained by passing a pointer to a specific service provider GUID in the *pguidServiceProvider*. This is useful, for example, when using the Modem Connection for Microsoft® DirectPlay® service provider. You can choose between different modems for dialing out and select specific modems for hosting.

If the *pEnumData* buffer is not big enough to hold the requested service provider information, the method returns DPNERR_BUFFERTOOSMALL and the *cbEnumData* parameter contains the required buffer size. Typically, the best strategy is to call the method once with a zero-length buffer to determine the required size. Then call the method again with the appropriate sized buffer.

Normally, this method will return only those service providers that can be used by the application. For example, if the IPX networking protocol is not installed, DirectPlay will not return the IPX service provider. To have DirectPlay return all service providers, even those that cannot be used by the application, set the DPNENUMSERVICEPROVIDERS_ALL flag in *dwFlags*.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetApplicationDesc

Retrieves the full application description for the connected application.

```
HRESULT GetApplicationDesc(
DPN_APPLICATION_DESC *const pAppDescBuffer,
DWORD *const pcbDataSize,
const DWORD dwFlags
);
```

Parameters

pAppDescBuffer

Pointer to a **DPN_APPLICATION_DESC** structure where the application description data is to be written. Set this parameter to NULL to request only the size of data. If *pAppDescBuffer* is not set to NULL, you must set the *pAppDescBuffer*.dwSize member to an appropriate value. The *pcbDataSize* parameter is set to the size required to hold the data.

pcbDataSize

Pointer to a variable of type **DWORD** that is initialized to the size of the buffer before calling this method. After the method returns, this parameter is set to the size, in bytes, of the session data. If the buffer is too small, this method returns the DPNERR_BUFFERTOOSMALL error value, and this parameter is set to the buffer size required. If this parameter is NULL, the method returns DPNERR_INVALIDPARAM.

dwFlags

Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_NOCONNECTION

Remarks

Call this method initially by passing NULL in the *pvData* parameter to obtain the size of the required buffer. When you call the method a second time to fill the buffer, be sure to set the structures **dwSize** member to the appropriate value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetCaps

Retrieves the **DPN_CAPS** structure for the current interface.

```
HRESULT GetCaps(
DPNCAPS *const pdpnCaps,
const DWORD dwFlags
);
```

Parameters

pdpnCaps
Pointer to a **DPN_CAPS** structure to receive caps information. You must set the **dwSize** member of this structure to an appropriate value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetClientAddress

Retrieves the address for the specified player in the session.

```
HRESULT GetClientAddress(
const DPNID dpnid,
IDirectPlay8Address **const pAddress,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** specifying the identification of the player.

pAddress
Address of a pointer to an **IDirectPlay8Address** object that specifies the address of the client. You must release this object when you no longer need it.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPLAYER
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

Use the **IDirectPlay8Server::GetLocalHostAddresses** method to retrieve addresses that can be used to connect to the session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetClientInfo

Retrieves the client information set for the specified client.

```
HRESULT GetClientInfo(
const DPNID dpnid,
DPN_PLAYER_INFO *const pdpnPlayerInfo,
DWORD *const pdwSize,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the client to retrieve the information for.

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure that is filled with client information. If *pdwSize* is not set to NULL, you must set *pdpnPlayerInfo*.dwSize to an appropriate value.

pdwSize
Pointer to a variable of type **DWORD** that contains the size of the client data returned in the *pdpnPlayerInfo* parameter. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the size of the required buffer.

dwFlags
Flags describing the information returned for the client. Currently, both of the following flags are returned.

DPNINFO_NAME
The **DPN_PLAYER_INFO** structure contains the name set for the client.

DPNINFO_DATA
The **DPN_PLAYER_INFO** structure contains the data set for the client.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

Remarks

Call this method after the server receives a **DPN_MSGID_CLIENT_INFO** message from the application. This message indicates that a client has updated its information.

Microsoft® DirectPlay® returns the **DPN_PLAYER_INFO** structure, and the pointers assigned to the structure's **pwszName** and **pvData** members in a contiguous buffer. If the two pointers were set, you must have allocated enough memory for the structure, plus the two pointers. The most robust way to use this method is to first call it with *pdwSize* set to NULL. When the method returns, *pdwSize* will point to the correct value. Use that value to allocate memory for the structure and call the method a second time to retrieve the information.

When the method returns, the **dwInfoFlags** member of the **DPN_PLAYER_INFO** structure will always have the DPNINFO_DATA and DPNINFO_NAME flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Client::SetClientInfo**, to notify DirectPlay of which values have changed.

Transmission of nonstatic information should be handled with the **IDirectPlay8Client::Send** method because of the high cost of using the **IDirectPlay8Client::SetPeerInfo** method.

The player sets the information by calling **IDirectPlay8Client::SetClientInfo**.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetConnectionInfo

Retrieves statistical information about the connection between the local server and the specified remote client.

```
HRESULT GetConnectionInfo(
const DPNID dpnidEndPoint,
DPN_CONNECTION_INFO *const pdnConnectInfo,
const DWORD dwFlags
);
```

Parameters

dpnidEndPoint
DPNID of the player whose connection information will be retrieved.

pdnConnectInfo
Pointer to a **DPN_CONNECTION_INFO** structure to retrieve information about the specified connection. The dwSize member of this structure must be set to the size of a **DPN_CONNECTION_INFO** structure.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

This method can be called only after a successful **Host** call has completed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetGroupContext

Retrieves the group context value for a group.

```
HRESULT GetGroupContext(
const DPNID dpnid,
PVOID *const ppvGroupContext,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group to get context data for.

ppvGroupContext
Pointer to the context value of the group.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDGROUP

DPNERR_INVALIDPARAM

Remarks

Group context values are set by pointing the **pvGroupContext** member of the **DPN_MSGID_CREATE_GROUP** system message to the context value data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetGroupInfo

Retrieves a block of data associated with a group, including the group name.

This method is typically called after a **DPN_MSGID_GROUP_INFO** system message is received, indicating that the group data has been modified.

```
HRESULT GetGroupInfo(
const DPNID dpnid,
DPN_GROUP_INFO *const pdpnGroupInfo,
DWORD *const pdwSize,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group whose data block will be retrieved.

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that describes the group data. If pdwSize is not set to NULL, you must set *pdpnGroupInfo*.dwSize to the size of a **DPN_GROUP_INFO** structure.

pdwSize
Pointer to a variable of type **DWORD** that returns the size of the data in the *pdpnGroupInfo* parameter. If the buffer is too small, this method returns DPNERR_BUFFERTOOSMALL and this parameter contains the required size.

dwFlags
Flags describing the information returned for the group. Currently, both of the following flags are returned.

DPNINFO_NAME
The **DPN_PLAYER_INFO** structure contains the name set for the client.

DPNINFO_DATA
The **DPN_PLAYER_INFO** structure contains the data set for the client.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Remarks

Microsoft® DirectPlay® returns the **DPN_GROUP_INFO** structure, and the pointers assigned to the structure's **pwszName** and **pvData** members in a contiguous buffer. If the two pointers were set, you must have allocated enough memory for the structure, plus the two pointers. The most robust way to use this method is to first call it with *pdwSize* set to NULL. When the method returns, *pdwSize* will point to the correct

value. Use that value to allocate memory for the structure and call the method a second time to retrieve the information.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetLocalHostAddresses

Retrieves the local addresses being used to host the session.

HRESULT GetLocalHostAddresses(
IDirectPlay8Address **const *prgpAddress*,
DWORD *const *pcAddress*,
constDWORD *dwFlags*
);

Parameters

prgpAddress
Address of a pointer to an array of **IDirectPlay8Address** objects that specify the local host addresses. You must release these objects when you no longer need them or you will create memory leaks.

pcAddress
Maximum number of address objects that can be contained in the array pointed to by *prgpAddress*. If the buffer is too small, the method returns DPNERR_BUFFERTOOSMALL, and *pcAddress* will be set to the required value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

The most robust way to use this method is to first call it with *pcAddress* set to 0. When the method returns, *pcAddress* will point to the required value. You can use that value when you call the method for a second time to retrieve the information.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetPlayerContext

Retrieves the player context value for a client.

```
HRESULT GetPlayerContext(
const DPNID dpnid,
PVOID *const ppvPlayerContext,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the player to get context data for.

ppvPlayerContext
Pointer to the context data of the client.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

Remarks

Player context values are set by pointing the **pvPlayerContext** member of the **DPN_MSGID_CREATE_PLAYER** system message to the context value data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetSendQueueInfo

Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® does not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should slow the rate that messages are sent.

```
HRESULT GetSendQueueInfo(
const DPNID dpnid,
DWORD *const pdwNumMsgs,
DWORD *const pdwNumBytes,
const DWORD dwFlags
);
```

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the player to get the send-queue information for.

pdwNumMsgs
Pointer to a variable of type **DWORD** that contains the number of messages currently queued. This value is optional, and may be set to NULL.

pdwNumBytes
Pointer to a variable of type **DWORD** that specifies the total number of bytes of data of the messages currently queued. This value is optional, and may be set to NULL.

dwFlags
You may specify the DPNGETSENDQUEUEINFO_PRIORITY_NORMAL, DPNGETSENDQUEUEINFO_PRIORITY_HIGH, or DPNGETSENDQUEUEINFO_PRIORITY_LOW flag to inquire about specific messages of that priority.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Remarks

You cannot set both *pdwNumMsgs* and *pdwNumBytes* to NULL. At least one of them must be set to a valid pointer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::GetSPCaps

Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT GetSPCaps(
const GUID *const pguidSP,
DPN_SP_CAPS *const pdpnSPCaps,
const DWORD dwFlags
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to get information about.

pdpnSPCaps
Pointer to a **DPN_SP_CAPS** structure to receive the information about the specified service provider. You must set the *pdpnSPCaps*.dwSize member of this structure to an appropriate value.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

This method retrieves information about the specified service provider. A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::Host

Creates a new client/server session, hosted by the local computer.

```
HRESULT Host(
const DPN_APPLICATION_DESC *const pdnAppDesc,
IDirectPlay8Address **const prgpDeviceInfo,
const DWORD cDeviceInfo,
const DPN_SECURITY_DESC *const pdpSecurity,
const DPN_SECURITY_CREDENTIALS *const pdpCredentials,
VOID *const pvPlayerContext,
const DWORD dwFlags
);
```

Parameters

pdnAppDesc
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application.

prgpDeviceInfo
Pointer to an array of **IDirectPlay8Address** objects containing device addresses that should be used to host the application.

cDeviceInfo
Variable of type **DWORD** that specifies the number of device address objects in the array pointed to by *prgpDeviceInfo*.

pdpSecurity
Reserved. Must be set to NULL.

pdpCredentials
Reserved. Must be set to NULL.

pvPlayerContext
Pointer to the context value of the player. This value is preset when the local computer handles the **DPN_MSGID_CREATE_PLAYER** message. This parameter is optional, and may be set to NULL.

dwFlags
The following flag can be specified.

DPNHOST_OKTOQUERYFORADDRESSING
Setting this flag will display a standard Microsoft® DirectPlay® dialog box, which queries the user for more information if not enough information is passed in this method.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_DATATOOLARGE
DPNERR_INVALIDPARAM

Remarks

If you set the DPNHOST_OKTOQUERYFORADDRESSING flag in *dwFlags*, the service provider may attempt to display a dialog box to ask the user to complete the address information. You must have a visible window present when the service provider tries to display the dialog box, or your application will lock.

The maximum size of the application data that you assign to the **pvApplicationReservedData** member of the **DPN_APPLICATION_DESC** structure is limited by the service provider's Maximum Transmission Unit. If your application data is too large, the method will fail and return DPNERR_DATATOOLARGE.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::Initialize

Registers an entry point in the server's code that receives the messages from the **IDirectPlay8Server** interface and from remote clients. This method must be called before calling any other methods of this interface.

HRESULT Initialize(
PVOID const *pvUserContext*,
const PFNDPNMESSAGEHANDLER *pfn*,
const DWORD *dwFlags*
);

Parameters

pvUserContext
Pointer to the user-provided context value in calls to the message handler. Providing a user-context value is useful to differentiate messages from multiple interfaces to a common message handler.

pfn
Pointer to a **PFNDPNMESSAGEHANDLER** callback function that receives all messages from remote clients and indications of session changes from the **IDirectPlay8Server** interface.

dwFlags
You may specify the following flag.
DPNINITIALIZE_DISABLEPARAMVAL
Passing this flag will disable parameter validation for the current object.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

Call this method first after using **CoCreateInstance** to obtain the **IDirectPlay8Server** interface.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::RegisterLobby

Allows launched applications to automatically propagate game status to the lobby.

```
HRESULT RegisterLobby(
const DPNHANDLE dpnHandle,
IDirectPlay8LobbiedApplication *const pIDP8LobbiedApplication,
const DWORD dwFlags
);
```

Parameters

dpnHandle
Connection handle used when making the calls to **IDirectPlay8LobbiedApplication::UpdateStatus**.

pIDP8LobbiedApplication
Pointer to the **IDirectPlay8LobbiedApplication** object that specifies the application.

dwFlags
One of the following flags.
DPNLOBBY_REGISTER
Registers the lobby with the application.
DPNLOBBY_UNREGISTER
Unregisters the lobby with the application.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::RemovePlayerFromGroup

Removes a client from a group.

HRESULT RemovePlayerFromGroup(
const DPNID *idGroup*,
const DPNID *idClient*,
PVOID const *pvAsyncContext*,
DPNHANDLE *const *phAsyncHandle*,
const DWORD *dwFlags*
);

Parameters

idGroup
Variable of type **DPNID** that specifies the identifier of the group to remove the client from.

idClient
Variable of type **DPNID** that specifies the identifier of the client to remove from the group.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNREMOVEPLAYERFROMGROUP_SYNC
Causes this method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Remarks

When this method is called, the server's message handler receives a **DPN_MSGID_REMOVE_PLAYER_FROM_GROUP** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::ReturnBuffer

Retrieves message buffers provided to the application through the **pReceiveData** member of the **DPN_MSGID_RECEIVE** system message. If the user's message handler returns DPNSUCCESS_PENDING to the RECEIVE callback, Microsoft® DirectPlay® assumes ownership of the buffer has been transferred to the application, and neither frees nor modifies it until ownership is returned to DirectPlay through this call.

```
HRESULT ReturnBuffer(
const DPNHANDLE hBufferHandle,
const DWORD dwFlags
);
```

Parameters

hBufferHandle
Variable of type **DPNHANDLE** that specifies the buffer handle for the message. This is obtained in the **hBufferHandle** member of the **DPN_MSGID_RECEIVE** system message.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDHANDLE

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SendTo

Transmits data to a client or group within the session. The message can be sent synchronously or asynchronously.

```
HRESULT SendTo(
const DPNID dpnid,
const DPN_BUFFER_DESC *const pBufferDesc,
const DWORD cBufferDesc,
const DWORD dwTimeOut,
void *const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

dpnid
Identifier of the client or group to receive data. Set this parameter to DPNID_ALL_PLAYERS_GROUP to send a message to all players in the session.

pBufferDesc
Pointer to a **DPN_BUFFER_DESC** structure that describes the data to send.

cBufferDesc
Number of **DPN_BUFFER_DESC** structures pointed to by *pBufferDesc*. There can be only one buffer in this version of Microsoft® DirectPlay®.

dwTimeOut
Number of milliseconds to wait for the message to be sent. If the message has not been sent by the *dwTimeOut* value, the message is not sent. If you do not want a time out for message sends, set this parameter to 0.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_SEND_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. When the method returns, *phAsyncHandle* will point to a handle that you can pass to **IDirectPlay8Server::CancelAsyncOperation** to cancel the operation. This parameter must be set to NULL if you set the DPNSEND_SYNC flag in *dwFlags*.

dwFlags

Flags that describe send behavior. You can set one or more of the following flags.

DPNSEND_SYNC

Process the **SendTo** request synchronously.

DPNSEND_NOCOPY

Use the data in the **DPN_BUFFER_DESC** structure and do not make an internal copy. This may be a more efficient method of sending data. However, it is less robust because the sender might be able to modify the message before the receiver has processed it. This flag cannot be used with DPNSEND_NOCOMPLETE.

DPNSEND_NOCOMPLETE

Do not send the **DPN_MSGID_SEND_COMPLETE** structure to the message handler. This flag may not be used with DPNSEND_NOCOPY or DPNSEND_GUARANTEED. Additionally, when using this flag *pvAsyncContext* must be NULL.

DPNSEND_COMPLETEONPROCESS

Send the **DPN_MSGID_SEND_COMPLETE** to the message handler when this message has been delivered to the target and the target's message handler returns from indicating its reception. There is additional internal message overhead when this flag is set, and the message transmission process may become significantly slower. If you set this flag, DPNSEND_GUARANTEED must also be set.

DPNSEND_GUARANTEED

Send the message by a guaranteed method of delivery.

DPNSEND_PRIORITY_HIGH

Sets the priority of the message to high. This flag cannot be used with DPNSEND_PRIORITY_LOW.

DPNSEND_PRIORITY_LOW

Sets the priority of the message to low. This flag cannot be used with DPNSEND_PRIORITY_HIGH.

DPNSEND_NOLOOPBACK

Suppress the **DPN_MSGID_RECEIVE** system message to your message handler when you are sending to a group that includes the local player. For example, this flag is useful if you are broadcasting to the entire session.

DPNSEND_NONSEQUENTIAL

If this flag is set, the target application will receive the messages in the order that they arrive at the user's computer. If this flag is not set, messages are delivered sequentially, and will be received by the target application in the order that they were sent. Doing so may require buffering incoming messages until missing messages arrive.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_CONNECTIONLOST

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_INVALIDPLAYER

DPNERR_TIMEDOUT

Remarks

This method generates a **DPN_MSGID_RECEIVE** system message in the receiver's message handler. The data is contained in the **pReceiveData** member of the associated structure.

Messages can have one of three priorities: low, normal, and high. To specify a low or high priority for the message set the appropriate flag in *dwFlags.* If neither of the priority flags is set, the message will have normal priority. See Basic Networking for a discussion of send priorities.

When the **SendTo** request is completed, a **DPN_MSGID_SEND_COMPLETE** system message is posted to the sender's message handler. The success or failure of the request is contained in the **hResultCode** member of the associated structure. You can suppress the send completion by setting the DPNSEND_NOCOMPLETE flag in *dwflags*.

Send completions are typically posted on the source computer as soon as the message is sent. In other words, a send completion does not necessarily mean that the message has been processed on the target. It may still be in a queue. If you want to be certain that the message has been processed by the target, set the DPNSEND_COMPLETEONPROCESS flag in *dwFlags.* This flag ensures that the send completion will not be sent until the target's message handler has processed the message, and returned.

Note

Do not assume that resources such as the data buffer will remain valid until the method has returned. If you call this method asynchronously, the **DPN_MSGID_SEND_COMPLETE** message may be received and processed by your message handler before the call has returned. If your message handler deallocates or otherwise invalidates a resource such as the data buffer, that resource may become invalid at any time after the method has been called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SetApplicationDesc

Changes the settings for the application that is being hosted. Only some settings can be changed.

```
HRESULT SetApplicationDesc(
const DPN_APPLICATION_DESC *const pad,
const DWORD dwFlags
);
```

Parameters

pad
Pointer to a **DPN_APPLICATION_DESC** structure that describes the application settings to modify.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_DATATOOLARGE
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

You can use this method to modify only the following members of the **DPN_APPLICATION_DESC** structure:

- **dwMaxPlayers**
- **pwszSessionName**
- **pwszPassword**
- **pvApplicationReservedData**
- **dwApplicationReservedDataSize**

The maximum size of the application data that you assign to the **pvApplicationReservedData** member of the **DPN_APPLICATION_DESC** structure is limited by the service provider's Maximum Transmission Unit. If your application data is too large, the method will fail and return DPNERR_DATATOOLARGE.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SetCaps

Sets the **DPN_CAPS** structure for the current interface.

```
HRESULT SetCaps(
const DPNCAPS *const pdpCaps,
const DWORD dwFlags
);
```

Parameters

pdpCaps
Pointer to a **DPN_CAPS** structure used to set the information about the current interface.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SetGroupInfo

Sets a block of data associated with a group, including the name of the group.

```
HRESULT SetGroupInfo(
```

const DPNID *dpnid*,
DPN_GROUP_INFO *const *pdpnGroupInfo*,
PVOID const *pvAsyncContext*,
DPNHANDLE *const *phAsyncHandle*,
const DWORD *dwFlags*
);

Parameters

dpnid
Variable of type **DPNID** that specifies the identifier of the group whose data block will be modified.

pdpnGroupInfo
Pointer to a **DPN_GROUP_INFO** structure that describes the group data to set. To change the values of the **pwszName** or **pvData** members of this structure, you must set the corresponding DPNINFO_NAME OR DPNINFO_DATA flag in the **dwInfoFlags** member.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNSETGROUPINFO_SYNC
Causes this method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDGROUP

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SetServerInfo

Sets the static settings of a server with an application. After clients successfully connect to the server, they can retrieve the information set by this method by calling the**IDirectPlay8Client::GetServerInfo** method.

```
HRESULT SetServerInfo(
const DPN_PLAYER_INFO *const pdpnPlayerInfo,
PVOID const pvAsyncContext,
DPNHANDLE *const phAsyncHandle,
const DWORD dwFlags
);
```

Parameters

pdpnPlayerInfo
Pointer to a **DPN_PLAYER_INFO** structure that contains the server information to set.

pvAsyncContext
Pointer to the user-supplied context, which is returned in the **pvUserContext** member of the **DPN_MSGID_ASYNC_OP_COMPLETE** system message.

phAsyncHandle
A **DPNHANDLE**. A value will be returned. However, Microsoft® DirectPlay® 8.0 does not permit cancellation of this operation, so the value cannot be used.

dwFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNSETSERVERINFO_SYNC
Causes this method to process synchronously.

Return Values

Returns S_OK if this method is processed synchronously and is successful. If the request is processed asynchronously, S_OK can return if the method is instantly processed. By default, this method is run asynchronously and generally returns DPNSUCCESS_PENDING or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_NOCONNECTION

Remarks

This method may be called before calling **IDirectPlayServer::Host**, and at any time during the session.

Handle transmission of nonstatic information with the **IDirectPlay8Server::SendTo** method because of the high cost of using the **IDirectPlay8Server::SetServerInfo** method.

You can modify the server information with this method after clients have connected to the application. Calling this method after connection generates a **DPN_MSGID_SERVER_INFO** system message to all players, informing them that data has been updated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlay8Server::SetSPCaps

Sets the **DPN_SP_CAPS** structure for the specified service provider.

```
HRESULT SetSPCaps(
const GUID *const pguidSP,
const DPN_SP_CAPS *const pdpnSPCaps
);
```

Parameters

pguidSP
Pointer to a GUID specifying the service provider you want to set information about.

pdpnSPCaps
Pointer to a **DPN_SP_CAPS** structure to set the information about the specified service provider.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_UNINITIALIZED

Remarks

This method sets parameters for the specified service provider. A successful call to **Initialize** must be made before this method can be called. Currently only the *dwNumThreads* member can be set by this call; the *dwFlags* member must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

IDirectPlayVoiceClient

Applications use the methods of the **IDirectPlayVoiceClient** interface to manage clients in a voice session.

The methods of the **IDirectPlayVoiceClient** interface can be organized into the following groups.

Buffer management	**Create3DSoundBuffer**
	Delete3DSoundBuffer
Miscellaneous	**GetCaps**
	GetCompressionTypes
	GetSoundDeviceConfig
	SetNotifyMask
Session management	**Connect**
	Disconnect
	GetClientConfig
	GetSessionDesc
	GetTransmitTargets
	Initialize
	SetClientConfig
	SetTransmitTargets

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::Connect

Connects the client to a Microsoft® DirectPlay® Voice session.

HRESULT Connect(
PDVSOUNDDEVICECONFIG *pSoundDeviceConfig*,
PDVCLIENTCONFIG *pdvClientConfig*,
DWORD *dwFlags*
);

Parameters

pSoundDeviceConfig
Pointer to a **DVSOUNDDEVICECONFIG** structure that describes the sound device configuration.

pdvClientConfig
Pointer to a **DVCLIENTCONFIG** structure that describes the general configuration of the client.

dwFlags
Flag. You can specify the following flag.

DVFLAGS_SYNC
The method does not return until the operation is completed.

Return Values

If the method is processed synchronously and is successful, it returns DV_OK. By default, this method is run asynchronously and returns DVERR_PENDING. On error, this method will return one of the following values.

DVERR_ALREADYPENDING
DVERR_COMPRESSIONNOTSUPPORTED
DVERR_INCOMPATIBLEVERSION
DVERR_INVALIDBUFFER
DVERR_INVALIDDEVICE
DVERR_INVALIDFLAGS
DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTINITIALIZED
DVERR_OUTOFMEMORY
DVERR_RUNSETUP
DVERR_SENDERROR
DVERR_SOUNDINITFAILURE
DVERR_TIMEOUT
DVERR_TRANSPORTNOPLAYER
DVERR_TRANSPORTNOSESSION
DVERR_CONNECTED
DVERR_NOVOICESESSION

Remarks

You must test the sound devices selected for playback and capture by invoking the setup wizard before connecting the client to the DirectPlay Voice session. On

application startup, check the audio configuration by using **IDirectPlayVoiceTest::CheckAudioSetup**. If this method returns DVERR_RUNSETUP, the sound configuration specified has not been tested. The setup wizard needs to be run only once for any configuration.

If you specify a buffer that is not the right format, the method will return DVERR_INVALIDBUFFER.

If the buffer or a portion of the buffer is locked when DirectPlay Voice attempts to write to it, the method will return DVERR_INVALIDBUFFER, and DirectPlay Voice will disconnect from the session. You will also receive a DVMSGID_SESSIONLOST message. The **hResult** member of the associated structure will be set to DVERR_LOCKEDBUFFER. Subsequent method calls will return a DVERR_NOTCONNECTED error code.

If full duplex operation is not supported, DirectPlay Voice falls back to half duplex (listen only) mode. To determine if you are in half-duplex mode, call **IDirectPlayVoiceClient::GetSoundDeviceConfig** after you have completed the connection. If you are in half-duplex mode, the **dwFlags** member of the **DVSOUNDDEVICECONFIG** structure will have the DVSOUNDCONFIG_HALFDUPLEX flag set.

Regardless of how the interfaces are obtained, the **DirectPlayVoiceClient** object maintains a reference, through a call to **AddRef**, to the **IDirectSound** and **IDirectSoundCapture** interfaces it uses until **IDirectPlayVoiceClient::Disconnect** is called. When **Disconnect** is called, the **DirectPlayVoiceClient** object calls **Release** on both interfaces.

If this method is called synchronously by setting the DVFLAGS_SYNC flag, the **DVMSG_CONNECTRESULT** message is not sent to the message handler. In this case, the connection result is determined by the return value of this method.

If this method is called asynchronously (by default), calling this method immediately returns a DVERR_PENDING error value and proceeds to process the connection request in the background. The status of the connection is not be known until the DirectPlay Voice client generates a **DVMSG_CONNECTRESULT** message with the connection result.

Any calls to **IDirectPlayVoiceClient::Connect** while a connection is pending return DVERR_ALREADYPENDING. Additionally, only one connection can be pending at a time.

A transport session must be started on the specified DirectPlay object before calling this method. A successful call to **IDirectPlayVoiceClient::Initialize** must be made before calling the **Connect** method.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::Create3DSoundBuffer

Retrieves a 3-D sound buffer for a player or group. You can use the methods of the 3-D sound buffer object to change the virtual 3-D position of incoming voice transmissions from the specified group or player.

```
HRESULT Create3DSoundBuffer(
DVID dvID,
LPDIRECTSOUNDBUFFER lpdsSourceBuffer,
DWORD dwPriority,
DWORD dwFlags,
LPDIRECTSOUND3DBUFFER* lpUserBuffer
);
```

Parameters

dvID

Variable of type **DVID** that specifies the identification of the player or group that the user wants to reserve a buffer for. You can also specify DVID_REMAINING to create a 3-D user buffer for all players or groups that do not have a user buffer. If DVID_REMAINING is specified, the *lpdsBufferDesc* must be NULL and the *dwPriority* and *dwFlags* parameters must be set to 0.

lpdsSourceBuffer

Pointer to an **IDirectSoundBuffer** interface, which is used to create the Microsoft® DirectPlay® Voice main buffer. This can be either NULL or a user-created Microsoft DirectSound® buffer. If this member is set to NULL, then DirectPlay Voice creates a buffer for you.

dwPriority

Direct pass-through. This value is passed in the *dwPriority* parameter when the call to **IDirectSoundBuffer::Play** is made. For more information, see **IDirectSoundBuffer8::Play**. This parameter must be 0 if *lpdsMainBufferDesc* is NULL.

dwFlags

Direct pass-through. This value is passed to the *dwFlags* parameter when the call to **IDirectSoundBuffer::Play** is made. For more information, see **IDirectSoundBuffer8::Play**. This parameter must be 0 if *lpdsMainBufferDesc* is NULL.

lpUserBuffer

Pointer to memory where the reserved buffer is placed.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYBUFFERED

DVERR_INVALIDOBJECT

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTALLOWED
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_OUTOFMEMORY
DVERR_SESSIONLOST

Remarks

If the DirectPlay voice session is a mixing server session, this method fails and returns DVERR_NOTALLOWED.

Although you can access all the member functions of the 3-D sound buffer object, because the DirectPlay voice client uses the buffer to stream incoming audio, do not use the **Lock**, **UnLock**, or **Play** methods of the DirectSound3DBuffer object.

If the user specifies a buffer, DirectPlay uses that buffer for the player's or group's buffer. User-created buffers have the following restrictions.

- The buffer must be 22 kilohertz, 16-bit, Mono format.
- The buffer must be at least 1 second in length.
- The buffer must have been created with the DSBCAPS_GETCURRENTPOSITION2 and DSBCAPS_CTRL3D flags.
- The buffer must not be a primary buffer.
- The buffer must not be playing when it is passed to DirectPlay.

If the buffer is not the right format, the method will return DVERR_INVALIDBUFFER.

The buffer must not be locked when you pass it to DirectPlay. When the buffer for the individual user is no longer required or when a player leaves the voice session, it is important to call **IDirectPlayVoiceClient::Delete3DSoundBuffer** to free up resources.

If the buffer or a portion of the buffer is locked when DirectPlay Voice attempts to write to it, the method will return DVERR_INVALIDBUFFER. If you lock the buffer after the method has returned, you will receive a DVMSGID_SESSIONLOST message. The **hResult** member of the associated structure will be set to DVERR_LOCKEDBUFFER. Subsequent method calls will return a DVERR_NOTCONNECTED error code.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::Delete3DSoundBuffer

Returns exclusive control of the 3-D sound buffer object to the Microsoft® DirectPlay® voice client object.

```
HRESULT Delete3DSoundBuffer(
DVID dvID
LPDIRECTSOUND3DBUFFER* lpUserBuffer
);
```

Parameters

dvID
DVID of the player or group that the user wants to delete a buffer for.

lpUserBuffer
Pointer to the user buffer to delete. This must be a user buffer obtained through the **IDirectPlayVoiceClient::Create3DSoundBuffer** method.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYBUFFERED
DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTALLOWED
DVERR_NOTBUFFERED
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

If the DirectPlay Voice session is a mixing server session, this method fails and returns DVERR_NOTALLOWED.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::Disconnect

Disconnects the Microsoft® DirectPlay® Voice client from the existing DirectPlay Voice session.

```
HRESULT Disconnect(
DWORD dwFlags
);
```

Parameters

dwFlags

Flag. You can specify the following flag.

DVFLAGS_SYNC

Do not return until the operation is completed.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYPENDING

DVERR_CONNECTABORTING

DVERR_INVALIDFLAGS

DVERR_INVALIDPARAM

DVERR_NOTCONNECTED

DVERR_NOTINITIALIZED

DVERR_PENDING

DVERR_SESSIONLOST

DVERR_TIMEOUT

Remarks

On calling this method, all recording and playback is stopped. If a connection is being processed, it is canceled by this call.

Unless the DVFLAGS_SYNC is specified, calling this method immediately returns a DVERR_PENDING error value and proceeds to process the disconnection request in the background. The status of the disconnection is not known until the DirectPlay Voice client generates a **DVMSG_DISCONNECTRESULT** message that contains the disconnection result. Only one disconnection can be pending at a time. If you call **IDirectPlayVoiceClient::Disconnect** while a disconnect is pending, DirectPlay will return a DVERR_ALREADYPENDING error value.

If this method is called synchronously by setting the DVFLAGS_SYNC flag, the method does not return until the **Disconnect** method completes. The result of the disconnection is the return value from this method. No **DVMSGID_DISCONNECTRESULT** message is generated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetCaps

Retrieves the Microsoft® DirectPlay® Voice capabilities.

```
HRESULT GetCaps(
PDVCAPS pCaps
);
```

Parameters

pCaps
Pointer to the **DVCAPS** structure that contains the capabilities of the DirectPlayVoiceClient object.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM

DVERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetClientConfig

Retrieves the client configuration.

```
HRESULT GetClientConfig(
PDVCLIENTCONFIG pClientConfig
);
```

Parameters

pClientConfig
Pointer to a **DVCLIENTCONFIG** structure that contains the configuration of the local client.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

Before calling this member, you must set the **dwSize** member of the **DVCLIENTCONFIG** structure.

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetCompressionTypes

Retrieves the available compression types on the system.

```
HRESULT GetCompressionTypes(
PVOID pData,
PDWORD pdwDataSize,
PDWORD pdwNumElements,
DWORD dwFlags
);
```

Parameters

pData
Pointer to buffer that receives an array of **DVCOMPRESSIONINFO** structures, one structure for every compression type supported through this object.

pdwDataSize
Pointer to a **DWORD** that contains the size of the buffer, in bytes, passed in the *pData* parameter.

pdwNumElements
Pointer to a **DWORD** where the method writes the number of elements returned in the array of **DVCOMPRESSIONINFO** structures. This contains the number of structures only if the buffer specified in the *pData* is large enough to hold the information.

dwFlags

Reserved. Must be 0.

Return Values

Returns DP_OK if successful, or one of the following error values.

DVERR_BUFFERTOOSMALL
DVERR_INVALIDFLAGS
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER

Remarks

If the buffer passed is not large enough to store the list of compression types, the method returns DVERR_BUFFERTOOSMALL and the *pdwDataSize* parameter is set to the minimum required size.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetSessionDesc

Retrieves the session properties.

HRESULT GetSessionDesc(
PDVSESSIONDESC *pvSessionDesc*
);

Parameters

pvSessionDesc
Pointer to a **DVSESSIONDESC** structure to receive the session description.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

Before calling this method, make sure to set the **dwSize** member of the **DVSESSIONDESC** structure.

This method may be called only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetSoundDeviceConfig

Retrieves the sound device configuration of the session.

```
HRESULT GetSoundDeviceConfig(
PDVSOUNDDEVICECONFIG pSoundDeviceConfig,
PDWORD pdwSize
);
```

Parameters

pSoundDeviceConfig
Pointer to a **DVSOUNDDEVICECONFIG** structure that is filled with the configuration of the sound device.

pdwSize
Pointer to a **DWORD** that specifies the size of the buffer in *pSoundDeviceConfig* parameter. If the buffer is too small, the method returns DVERR_BUFFERTOOSMALL and this parameter contains the size of the required buffer.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::GetTransmitTargets

Retrieves the transmit targets, if any, of the voice stream from this client.

HRESULT GetTransmitTargets(
PDVID *pdvIDTargets*,
PDWORD *pdwNumTargets*,
DWORD *dwFlags*
);

Parameters

pdvIDTargets
Member to fill with an array of DVIDs that specify the targets that were set by the **IDirectPlayVoiceClient::SetTransmitTargets** or **IDirectPlayVoiceServer::SetTransmitTargets** method. You can retrieve the number of targets by specifying NULL for this parameter.

pdwNumTargets
Number of DVIDs in the array. When you call this method, this should be the same value as the number of targets set in the **IDirectPlayVoiceClient::SetTransmitTargets** method. If the call is successful, Microsoft® DirectPlay® returns the number of elements written to the *pdvIDTargets* array.

If *pdvIDTargets* is NULL, this must be 0.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_BUFfERTOOSMALL
DVERR_INVALIDFLAGS
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER

DVERR_NOTALLOWED
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED

Remarks

The value returned in the *pdvIDTargets* parameter can be player or group DVIDs or the DVID_ALLPLAYERS constant.

If the buffer specified in *pdvIDTargets* is not large enough to store the list of targets, this method returns DVERR_INVALIDPOINTER and *pdwNumTargets* is set to the required number of elements.

If there is no target specified, *pdwNumTargets* is set to 0 and the return value is DV_OK.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::Initialize

Initializes the DirectPlayVoiceClient object by associating it with a DirectPlay object. Additionally, this method registers a message handler with the DirectPlayVoiceClient object.

This method must be called successfully before **IDirectPlayVoiceClient::Connect** method is called.

HRESULT Initialize(
PVOID *pVoid*,
PDVMESSAGEHANDLER *pMessageHandler*,
PVOID *pUserContext*,
PDWORD *pdwMessageMask*,
DWORD *dwMessageMaskElements*
);

Parameters

pVoid
Pointer to the **IUnknown** interface for the DirectPlay object that this DirectPlayVoiceClient object should use.

pMessageHandler
User-defined callback function that is called when there is a DirectPlayVoiceClient message to be processed. Threads within the

DirectPlayVoiceClient object call the callback function, so it will not be called in the context of your process's main thread.

pUserContext
Pointer to an application-defined structure that is passed to the callback function each time the function is called.

pdwMessageMask
Array of **DWORD**s that contain the message identifiers that you want DirectPlay Voice to send to your callback function. If a message identifier is not specified in this array, it is not sent. Each message identifier should appear only once in the array and only valid message identifiers are allowed. For example, DVMSGID_CONNECTRESULT is not valid for the server interface, but is for the client interface. To enable all messages, specify NULL for this value.

dwMessageMaskElements
Number of elements specified in the *pdwMessageMask* parameter. If *pdwMessageMask* is NULL, this must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYINITIALIZED
DVERR_GENERIC
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOCALLBACK
DVERR_TRANSPORTNOTINIT

Remarks

You can call **IDirectPlayVoiceClient::SetNotifyMask** to change the notify mask during the course of the voice session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::SetClientConfig

Sets the client configuration.

HRESULT SetClientConfig(
PDVCLIENTCONFIG *pClientConfig*
);

Parameters

pClientConfig
Pointer to the **DVCLIENTCONFIG** structure that contains the configuration description to set.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDFLAGS
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

Calling this method sets all the parameters in the **DVCLIENTCONFIG** structure. Therefore, to leave a setting unmodified, you must retrieve the current configuration with **IDirectPlayVoiceClient::GetClientConfig**. Then modify the parameters to change and call **IDirectPlayVoiceClient::SetClientConfig**.

If the session is running in half duplex, the members of **GetClientConfig** related to recording are ignored.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::SetNotifyMask

Specifies which messages are sent to the message handler.

HRESULT SetNotifyMask(
PDWORD *pdwMessageMask*,
DWORD *dwMessageMaskElements*
);

Parameters

pdwMessageMask

Pointer to an array of **DWORD**s containing the message identifiers that you want Microsoft® DirectPlay® Voice to send to your callback function. If a message identifier is not specified in this array, it is not sent. Each message identifier should appear only once in the array, and only valid message identifiers are allowed. For example, DVMSGID_CONNECTRESULT is not valid for the server interface, but is for the client interface. To enable all messages, specify NULL for this value.

dwMessageMaskElements

Number of elements specified in the *pdwMessageMask* parameter. If *pdwMessageMask* is NULL, this must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM

DVERR_INVALIDPOINTER

DVERR_NOCALLBACK

DVERR_NOTINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceClient::SetTransmitTargets

Specifies which players and/or groups receive audio transmissions from the local client.

HRESULT SetTransmitTargets(
PDVID *pdvIDTargets*,
DWORD *dwNumTargets*,
DWORD *dwFlags*
);

Parameters

pdvIDTargets

Pointer an array of DVIDs that specify your targets. To specify no targets, pass NULL for this parameter. Additionally, this parameter can be set to the following value.

DVID_ALLPLAYERS

The client is targeting all players in the session. This must be the only element in the array.

dwNumTargets
Number of DVIDs in the array. This value cannot exceed 64. If *pdvIDTargets* is NULL, this must be 0.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDFLAGS
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_INVALIDTARGET
DVERR_NOTINITIALIZED

Remarks

For Microsoft® DirectX® 8.0, the number of individual targets that you can transmit to is limited to 64. If you exceed this value, the method will fail, and return DVERR_NOTALLOWED. However, you can transmit to more than 64 players. To do so, form the players into groups, and then use the group as your target.

The *pdvIDTargets* parameter specifies an array of player and/or group DVIDs. There must be no duplicate targets in this parameter, and all entries must be valid DVIDs. If a target contains a player as its individual DVID and through a group that the target belongs to, Microsoft® DirectPlay® Voice ensures duplicate speech packets are not sent to the player.

If the session was created with the DVSESSION_SERVERCONTROLTARGET flag, only the server can set the targets for this local client. A call to this method returns DVERR_NOTALLOWED.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer

Applications use the methods of the **IDirectPlayVoiceServer** interface to manage the host of the voice session.

The methods of the **IDirectPlayVoiceServer** interface can be organized into the following groups.

Miscellaneous **GetCaps**

	GetCompressionTypes
	SetNotifyMask
Session management	**GetSessionDesc**
	GetTransmitTargets
	Initialize
	SetSessionDesc
	SetTransmitTargets
	StartSession
	StopSession

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::GetCaps

Retrieves the capabilities of the Microsoft® DirectPlay® Voice server for this system.

```
HRESULT GetCaps(
PDVCAPS pDVCaps
);
```

Parameters

pDVCaps
Pointer to the **DVCAPS** structure that contains the capabilities of the DirectPlayVoiceServer object.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDOBJECT

DVERR_INVALIDPARAM

DVERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::GetCompressionTypes

Retrieves available compression types for the system.

HRESULT GetCompressionTypes(
PVOID *pData,*
PDWORD *pdwDataSize,*
PDWORD *pdwNumElements,*
DWORD *dwFlags*
);

Parameters

pData
Pointer to the buffer that receives an array of **DVCOMPRESSIONINFO** structures that describe the compression types supported by this object.

pdwDataSize
Pointer to a **DWORD** that contains the size of the buffer, in bytes, passed in the *pData* parameter.

pdwNumElements
Pointer to a **DWORD** where the method writes the number of elements returned in the array of **DVCOMPRESSIONINFO** structures.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_BUFFERTOOSMALL

DVERR_INVALIDFLAGS

DVERR_INVALIDPARAM

DVERR_INVALIDPOINTER

Remarks

If the buffer is not large enough to store the list of compression types, the method returns DVERR_BUFFERTOOSMALL and the *pdwDataSize* parameter is set to the minimum required size.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::GetSessionDesc

Retrieves the Microsoft® DirectPlay® Voice session settings.

HRESULT GetSessionDesc(
PDVSESSIONDESC *pvSessionDesc*
);

Parameters

pvSessionDesc
Pointer to a **DVSESSIONDESC** structure to receive the session description.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTHOSTING
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

Before calling this method, make sure to set the **dwSize** member of the **DVSESSIONDESC** structure.

A successful call to **IDirectPlayVoiceServer::StartSession** must be made before this method can be called.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::GetTransmitTargets

Retrieves the transmit targets, if any, of the voice stream for a player in a session.

HRESULT GetTransmitTargets(
DVID *dvSource*,
PDVID *pdvIDTargets*,
PDWORD *pdwNumTargets*,

DWORD *dwFlags*
);

Parameters

dvSource
DVID of the user or group whose target is returned.

pdvIDTargets
Array of DVIDs that specify the current targets of the player or group that were set by the **IDirectPlayVoiceServer::SetTransmitTargets** method. You can retrieve the number of targets by specifying NULL for this parameter.

pdwNumTargets
Number of DVIDs in the array. When you call this method, this should be the same value as the number of targets set in the **IDirectPlayVoiceServer::SetTransmitTargets** method. If the call is successful, Microsoft® DirectPlay® returns the number of elements in the *pdvIDTargets* array.

If *pdvIDTargets* is NULL, this must be 0.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_BUFFERTOOSMALL
DVERR_INVALIDFLAGS
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTALLOWED
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED

Remarks

This method can be used only if the DVSESSION_SERVERCONTROLTARGET flag is specified on creation of the DirectPlay Voice session. If the flag is not specified, this method returns DVERR_NOTALLOWED.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::Initialize

Initializes the DirectPlayVoiceServer object by associating it with a DirectPlay object. Additionally, this method registers a message handler with this interface.

HRESULT Initialize(
LPVOID *lpVoid*,
PDVMESSAGEHANDLER *pMessageHandler*,
PVOID *pUserContext*,
LPDWORD *lpdwMessageMask*,
DWORD *dwMessageMaskElements*
);

Parameters

lpVoid
Pointer to the **IUnknown** interface for the DirectPlay object that this DirectPlayVoiceServer object should use.

pMessageHandler
User-defined callback function that is called when there is a DirectPlayVoiceClient message to process. A thread within the DirectPlayVoiceClient object calls the callback function, so it is not called in the context of your process's main thread.

pUserContext
Pointer to an application-defined structure that is passed to the callback function each time the method is called.

lpdwMessageMask
Array of **DWORD**s that contain the message identifiers that you want DirectPlay Voice to send to your callback function. If a message identifier is not specified in this array, it is not sent. Each message identifier should appear only once in the array, and only valid message identifiers are allowed. For example, DVMSGID_CONNECTRESULT is not valid for the server interface but is for the client interface. To enable all messages, specify NULL for this value.

dwMessageMaskElements
Number of elements specified in the *lpdwMessageMask* parameter. If *lpdwMessageMask* is NULL, this must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYINITIALIZED
DVERR_GENERIC
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOCALLBACK
DVERR_TRANSPORTNOTINIT

Remarks

You can call **IDirectPlayVoiceServer::SetNotifyMask** to change the notify mask during the course of the voice session.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::SetNotifyMask

Specifies which messages are sent to the message handler.

HRESULT SetNotifyMask(
PDWORD *pdwMessageMask*,
DWORD *dwMessageMaskElements*
);

Parameters

pdwMessageMask
Pointer to an array of **DWORD**s that contain the message identifiers that you want Microsoft® DirectPlay® Voice to send to your callback function. If a message identifier is not specified in this array, it is not sent. Each message identifier should appear only once in the array, and only valid message identifiers are allowed. For example, DVMSGID_CONNECTRESULT is not valid for the server interface but is for the client interface. To enable all messages, specify NULL for this value.

dwMessageMaskElements
Number of elements specified in the *pdwMessageMask* parameter. If *pdwMessageMask* is NULL, this must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOCALLBACK
DVERR_NOTINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::SetSessionDesc

Sets the session settings.

HRESULT SetSessionDesc(
PDVSESSIONDESC *pSessionDesc*
);

Parameters

pSessionDesc
Pointer to a **DVSESSIONDESC** structure that contains the session description.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTHOSTING
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

After the Microsoft® DirectPlay® voice session has started, not all the session properties of the **DVSESSIONDESC** structure can be changed. For more information, see **DVSESSIONDESC**.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::SetTransmitTargets

Controls the transmission of audio from the client to the specified members of the session.

HRESULT SetTransmitTargets(
DVID *dvSource*,
PDVID *pdvIDTargets*,
DWORD *dwNumTargets*,
DWORD *dwFlags*
);

Parameters

dvSource
DVID of the user whose targets are set.

pdvIDTargets
List of player DVIDs and/or group DVIDs that are the target for audio transmission. To specify no targets, pass NULL for this parameter. Additionally, this parameter can be set to the following value.

DVID_ALLPLAYERS
This client is targeting all players in the session. This must be the only element in the array.

dwNumTargets
Number of DVIDs in the array. This value cannot exceed 64. If *pdvIDTargets* is NULL this must be 0.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_INVALIDFLAGS

DVERR_INVALIDPARAM

DVERR_INVALIDPOINTER

DVERR_INVALIDTARGET

DVERR_NOTALLOWED

DVERR_NOTINITIALIZED

Remarks

For Microsoft® DirectX® 8.0, the number of individual targets that you can transmit to is limited to 64. If you exceed this value, the method will fail, and return DVERR_NOTALLOWED. However, you can transmit to more than 64 players. To do so, form the players into groups, and then use the group as your target.

There must be no duplicate targets in this parameter, and all entries must be valid DVIDs. If a target contains a player as its individual DVID and through a group that the target belongs to, Microsoft® DirectPlay® Voice ensures duplicate speech packets are not sent to the player.

This method can be used only if the DVSESSION_SERVERCONTROLTARGET flag is specified on creation of the DirectPlay Voice session. If the flag is not specified, this method returns DVERR_NOTALLOWED.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::StartSession

Starts an initialized Microsoft® DirectPlay® Voice session within a running DirectPlay transport session. This method must be successfully called before the clients can complete a connection-to-the-voice session.

HRESULT StartSession(
PDVSESSIONDESC *pSessionDesc*,
DWORD *dwFlags*
);

Parameters

pSessionDesc
Pointer to a **DVSESSIONDESC** structure that contains the session description.

dwFlags
Reserved. Must be 0.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYPENDING
DVERR_HOSTING
DVERR_INVALIDFLAGS
DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTINITIALIZED

Remarks

The **IDirectPlayVoiceServer::Initialize** method must be called before this method is called. The voice session can be hosted on any client in the session if the voice session is peer-to-peer. If the voice session is not peer-to-peer, it must be hosted on the transport client, which is the host of a active transport session.

The **DVSESSIONDESC** structure contains the type of voice session to start. The type of voice session can have a dramatic effect on the CPU and bandwidth usage for both the client and the server. You can set the **guidCT** member of **DVSESSIONDESC** to DPVCTGUID_DEFAULT.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceServer::StopSession

Stops the Microsoft® DirectPlay® Voice session.

HRESULT StopSession(
DWORD *dwFlags*
);

Parameters

dwFlags
Flag. The following flag can be set.
DVFLAGS_NOHOSTMIGRATE
The host will not migrate regardless of session and transport settings. Use this flag when you want to shut down the voice session completely.

Return Values

Returns DV_OK if successful, or one of the following error values.

DVERR_ALREADYPENDING
DVERR_INVALIDFLAGS
DVERR_INVALIDOBJECT
DVERR_INVALIDPARAM
DVERR_NOTHOSTING
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

This method returns DVERR_ALREADYPENDING if it is called while another thread is processing a **StopSession** request.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceTest

Applications use the **CheckAudioSetup** method of the **IDirectPlayVoiceTest** interface to test the Microsoft® DirectPlay® Voice audio configuration.

Audio Configuration **CheckAudioSetup**

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlayVoiceTest::CheckAudioSetup

Runs the Audio Setup Wizard on the specified devices. This wizard runs a series of tests on the devices to determine if they are capable of full duplex audio and to ensure that the microphone is plugged in and working correctly on the capture device.

```
HRESULT CheckAudioSetup(
const GUID * pguidPlaybackDevice,
const GUID * pguidCaptureDevice,
HWND hwndParent,
DWORD dwFlags
);
```

Parameters

pguidPlaybackDevice

Pointer to the GUID that identifies the playback device to test. If NULL is passed for this parameter, Microsoft® DirectPlay® Voice tests the system default playback device defined by Microsoft® DirectSound®. You can also pass one of the DirectSound default GUIDs:

DSDEVID_DefaultPlayback
The system default playback device.

DSDEVID_DefaultVoicePlayback
The default voice playback device.

pguidCaptureDevice

Pointer to the GUID that identifies the capture device to test. If NULL is passed for this parameter, DirectPlay Voice tests the system default capture device

defined by DirectSound. You can also pass one of the DirectSound default GUIDs:

DSDEVID_DefaultCapture
The default system capture device. You can also specify this device by passing a NULL pointer in the device GUID parameter.

DSDEVID_DefaultVoiceCapture
The default voice communications capture device. Typically, this is a secondary device such as a USB headset with microphone.

hwndParent
The test wizard invoked by this method is modal. If the calling application has a window that should be the parent window of the wizard, it should pass a handle to that window in this parameter. If the calling application does not have a window, it can pass NULL. If the DVFLAGS_QUERYONLY flag is specified, this parameter is not used and the application can pass NULL.

dwFlags
Flags. The following flags can be set.

DVFLAGS_QUERYONLY
Audio setup is not run. Instead, the method checks the registry to see if the devices have been tested. If the devices have not been tested, the method returns DVERR_RUNSETUP. If the devices have been tested, the method returns DV_FULLDUPLEX if the devices support full duplex audio, or DV_HALFDUPLEX if the devices do not support full duplex audio.

DVFLAGS_ALLOWBACK
Passing this flag enables the **Back** button on the wizard's Welcome page. If the user clicks the **Back** button on the Welcome page, the wizard exits, and **CheckAudioSetup** returns DVERR_USERBACK.

Return Values

Returns DV_OK, DV_FULLDUPLEX, DV_HALFDUPLEX if successful, or one of the following error values.

DVERR_INVALIDPARAM

DVERR_RUNSETUP

DVERR_INVALIDDEVICE

Remarks

This method contains user interface (UI) elements and displays dialog boxes. If the DVFLAGS_QUERYONLY flag is specified, the tests are not actually run and no UI is raised. Instead, the registry is checked to determine the results of a previous test of these devices.

If the user cancels the wizard, the **CheckAudioSetup** call returns DVERR_USERCANCEL. The calling application can then handle the situation appropriately. For example, in DirectPlay Voice part of the gaming options control

panel application, if the user clicks **Cancel**, the dialog box displays a message indicating that voice cannot be used because the wizard has been canceled.

This method might return DVERR_INVALIDDEVICE if the device specified does not exist. Also, if you specify the default device and this method still returns this error, then there are no sound devices on the system.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

IDirectPlay8LobbyClient

The **IDirectPlay8LobbyClient** interface is used by a lobby client application and is responsible for enumerating and launching lobby-enabled game applications on the local computer, and communicating with them once they are running. The lobby client must register a message handler routine to process messages from the lobby and the lobbied game application.

The methods of the **IDirectPlay8LobbyClient** interface are:

IDirectPlay8LobbyClient Methods	**Close**
	ConnectApplication
	EnumLocalPrograms
	Initialize
	ReleaseApplication
	Send
	GetConnectionSettings
	SetConnectionSettings

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::Close

Deletes the lobby client.

```
HRESULT Close(
const DWORD dwFlags
);
```

Parameters

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_UNINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::ConnectApplication

Connects a lobby-enabled application to the session specified in the **DPL_CONNECT_INFO** structure. If the application is not running, this method can be used to launch the application.

When the connection is successfully established, the lobbied application generates a **DPL_MSGID_CONNECT** system message to the message handler.

```
HRESULT ConnectApplication(
DPL_CONNECT_INFO *const pdplConnectionInfo,
const PVOID pvUserApplicationContext,
DPNHANDLE *const phApplication,
const DWORD dwTimeOut,
const DWORD dwFlags
);
```

Parameters

pdplConnectionInfo
Pointer to a **DPL_CONNECT_INFO** structure, which describes the connection parameters, including the GUID of the application to connect to.

pvUserApplicationContext
Pointer to a context value defined for the lobby client that is passed in calls to the lobby client's message handler.

phApplication
Pointer to a **DPNHANDLE** that specifies the application connect handle that is set if this method succeeds. This handle is used for further communication with the application. Additionally, this handle is used in the *phApplication* parameter in the **IDirectPlay8LobbyClient::ReleaseApplication** method.

dwTimeOut
Variable of type **DWORD** that specifies the number of milliseconds to wait for the connection to process.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_CANTLAUNCHAPPLICATION

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_TIMEDOUT

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::EnumLocalPrograms

Enumerates the lobbied applications that are registered on the system.

HRESULT EnumLocalPrograms(
GUID *const *pGuidApplication*,
BYTE *const *pEnumData*,
DWORD *const *pdwEnumData*,
DWORD *const *pdwItems.*
const DWORD *dwFlags*
);

Parameters

pGuidApplication
Pointer to a variable of type **GUID** that specifies the lobbied application to enumerate. This parameter is optional, and passing NULL enumerates all available lobbied applications.

pEnumData
Pointer to a variable of type **BYTE**, which is filled with a description of the lobbied application.

pdwEnumData
Pointer to variable of type **DWORD** that specifies the number of bytes contained in the *pEnumData* buffer. If the buffer in *pEnumData* is too small, this method

returns DPNERR_BUFFERTOOSMALL and sets this parameter to the size of the required buffer.

pdwItems
Pointer to a variable of type **DWORD** that contains the number of **DPL_APPLICATION_INFO** structures in the *pEnumData* buffer. This parameter is filled only if the method succeeds.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

This method is generally called twice—once to obtain the size of the required buffer, and then with the correct buffer size.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::Initialize

Registers an entry point in the lobby client's code that receives notifications on changes of state for any launched applications. The message handler also receives messages from the lobbied application. This method must be called before calling any other methods of this interface.

HRESULT Initialize(
const PVOID *pvUserContext*,
const PFNDPNMESSAGEHANDLER *pfn*,
const DWORD *dwFlags*
);

Parameters

pvUserContext
Pointer to the user-provided context value provided in calls to the message handler. Providing a user-context value is useful to differentiate messages from multiple interfaces to a common message handler.

pfn

Pointer to a **PFNDPNMESSAGEHANDLER** callback function that receives all messages from the **IDirectPlay8LobbyClient** interface and indications of session changes from the **IDirectPlay8LobbiedApplication** interface.

dwFlags

The following flag can be specified.

DPLINITIALIZE_DISABLEPARAMVAL
Disables parameter validation.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

Call this is method first after using **CoCreateInstance** to obtain the **IDirectPlay8LobbyClient** interface.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::ReleaseApplication

Releases a lobbied application and closes the connection between the lobby client and the application. This method should be called whenever a lobby client has finished its session with an application.

HRESULT ReleaseApplication(
const DPNHANDLE *hApplication.*
const DWORD *dwFlags*
);

Parameters

hApplication

The **DPNHANDLE** of the lobbied application to release. This value is set in the *phApplication* parameter of the **IDirectPlay8LobbyClient::ConnectApplication** method. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS

All application connections will be released.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDHANDLE
DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::Send

Sends a message to a lobbied application that was launched by this lobby client or was connected by this lobby client.

This method sends a **DPL_MSGID_RECEIVE** system message to the target's message handler.

```
HRESULT Send(
const DPNHANDLE hConnection,
BYTE *const pBuffer,
const DWORD pBufferSize,
const DWORD dwFlags
);
```

Parameters

hConnection
Variable of type **DPNHANDLE** that specifies the target for the message transmission. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS
The message you have specified will be sent to all lobbied applications that are connected to your lobby client application.

pBuffer
Pointer to an array of bytes that contains the message.

pBufferSize
Variable of type **DWORD** that specifies the size of the message buffer in the *pBuffer* parameter, in bytes. This parameter must be at least 1 byte and no more than 64 KB.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDHANDLE
DPNERR_INVALIDPARAM
DPNERR_SENDTOOLARGE

Remarks

If the buffer size is larger than 64 KB, the method returns DPNERR_SENDTOOLARGE. If the buffer size is set to 0, the method returns DPNERR_INVALIDPARAM.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::GetConnection Settings

Retrieves the set of connection settings for the specified connection. These settings can be set through a call to the **IDirectPlay8LobbyClient::ConnectApplication**, **IDirectPlay8LobbyClient::SetConnectionSettings**, or **IDirectPlay8LobbiedApplication::SetConnectionSettings** method.

When you get connection settings, a reference will be added for each address object that is returned to the user. Therefore, users must be sure to call **Release** on each address object when they are finished with the structure.

```
HRESULT GetConnectionSettings(
const DPNHANDLE hConnection,
DPL_CONNECTION_SETTINGS *const pdplConnectSettings,
DWORD*pdwDataSize,
const DWORD dwFlags
);
```

Parameters

hConnection
Handle to the connection for which to retrieve the settings.

pdplConnectSettings
Pointer to a buffer to receive the connection settings for the specified connection.

pdwDataSize
Pointer to a **DWORD** containing the size, in bytes, of the buffer specified in the *pdplConnectSettings* structure. If the buffer is not large enough to hold the connection settings, DPNERR_BUFFERTOOSMALL is returned and this value is set to the required buffer size. On success, this value contains the number of bytes written to the specified buffer.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDOBJECT

DPNERR_INVALIDFLAGS

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbyClient::SetConnection Settings

Sets the connection settings to be associated with the specified connection. Calling this method will generate a DPL_MSGID_CONNECTION_SETTINGS message to be sent to the client specified by *hConnection*.

When you set connection settings, the lobby application will add a reference to each of the address objects specified in the call.

```
HRESULT SetConnectionSettings(
const DPNHANDLE hConnection,
const DPL_CONNECTION_SETTINGS *const pdplConnectSettings,
const DWORD dwFlags
);
```

Parameters

hConnection
Handle to the connection to set the settings for. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS
The connection settings will be updated for all the lobbied applications you are connected to.

pdplConnectSettings
Pointer to a DPL_CONNECTION_SETTINGS structure containing the settings associated with the specified connection.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDOBJECT

DPNERR_INVALIDFLAGS

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication

The **IDirectPlay8LobbiedApplication** interface is used by an application that supports lobbying. This interface allows the application to register with the system so that it can be lobby launched. Additionally, it also lets the application get the connection information necessary to launch a game without querying the user. Lastly, this interface allows the lobbied application to send messages and notifications to the lobby client that launched the application.

The methods of the **IDirectPlay8LobbiedApplication** interface are:

IDirectPlay8LobbiedApplication Methods

Close
SetAppAvailable
Initialize
RegisterProgram
Send
UnRegisterProgram
UpdateStatus
GetConnectionSettings
SetConnectionSettings

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::Close

Deletes the lobbied application.

```
HRESULT Close(
const DWORD dwFlags
);
```

Parameters

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_UNINITIALIZED

DPNERR_INVALIDOBJECT

DPNERR_OUTOFMEMORY

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::SetApp Available

Makes an application available or unavailable for a lobby client to connect to. This method is typically called if a lobbied application is independently launched, that is, not launched by a lobby client. Additionally, this method should be called if a game has ended and the lobbied application needs to be available to connect to a lobby client at the start of another game.

```
HRESULT SetAppAvailable(
const BOOL fAvailable,
const DWORD dwFlags
);
```

Parameters

fAvailable

Boolean value that sets the availability of the application. Set this value to TRUE to indicate that your application is available, or to FALSE to indicate that it is not available.

dwFlags

The following flag can be set for this method.

DPLAVAILABLE_ALLOWMULTIPLECONNECT

The default behavior for this method is to automatically mark the interface as Unavailable when the first connection is established. By specifying this flag, the interface is not automatically marked unavailable after the first connection is established, thereby allowing multiple connections.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDOBJECT

DPNERR_UNINITIALIZED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::Initialize

Registers a message handler function that receives notifications about changes in the state of the lobby client and receives messages from the lobby client.

```
HRESULT Initialize(
const PVOID pvUserContext,
const PFNDPNMESSAGEHANDLER pfn,
DPNHANDLE *const pdpnhConnection,
const DWORD dwFlags
);
```

Parameters

pvUserContext

Pointer to the user-provided context value in calls to the message handler. Providing a user-context value is useful to differentiate messages from multiple interfaces to a common message handler.

pfn

Pointer to a **PFNDPNMESSAGEHANDLER** callback function that receives all messages from the **IDirectPlay8LobbyClient** interface and indications of session changes from the **IDirectPlay8LobbiedApplication** interface.

pdpnhConnection

Value used to detect if your application was lobby launched. If your application was lobby launched, this parameter will be set to the connection handle for the lobby client. If your process was not lobby launched, this parameter is set to NULL.

dwFlags

The following flag can be specified.

DPLINITIALIZE_DISABLEPARAMVAL
Disables parameter validation.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

Call this method first after using **CoCreateInstance** to obtain the **IDirectPlay8LobbiedApplication** interface.

This method automatically establishes a connection to the lobby client if you were lobby launched. If you call **Initialize** and you were lobby launched and the lobbied application interface is unable to contact the lobby client process, **Initialize** will time out after four seconds. In this case, **Initialize** will return DPNERR_TIMEDOUT but will still succeed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::RegisterProgram

Registers a lobby-aware application with DirectPlay. Applications must be registered to enable lobby launching.

HRESULT RegisterProgram(
PDPL_PROGRAM_DESC *pdplProgramDesc*,
const DWORD *dwFlags*
);

Parameters

pdplProgramDesc
Pointer to the **DPL_PROGRAM_DESC** structure that describes the lobby-aware application to register.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

The application needs to register only once. It should be unregistered with a call to the **IDirectPlay8LobbiedApplication::UnRegisterProgram** method when it is uninstalled.

In Microsoft DirectX® 8.0, **RegisterProgram** must be used. You cannot manually enter application information in the registry. Failure to use this interface makes your application nonportable and incompatible with future versions of DirectPlay.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::Send

Sends a message from the lobbied application to the lobby client.

```
HRESULT Send(
const DPNHANDLE hConnection,
BYTE *const pBuffer,
const DWORD pBufferSize,
const DWORD dwFlags
);
```

Parameters

hConnection
Variable of type **DPNHANDLE** that specifies the lobby client that the message is sent to. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS

The message you have specified will be sent to all lobby clients to which you are connected.

pBuffer
Pointer to a variable of type **BYTE** that contains the message buffer.

pBufferSize
Variable of type **DWORD** that specifies the size of the message buffer in the *pBuffer* parameter, in bytes. This parameter must be at least 1 byte and no more than 64 KB.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDHANDLE

DPNERR_INVALIDPARAM

DPNERR_SENDTOOLARGE

Remarks

If the buffer size is larger than 64 KB, the method returns DPNERR_SENDTOOLARGE. If the buffer size is set to 0, the method returns DPNERR_INVALIDPARAM.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::UnRegisterProgram

Unregisters a lobby-aware application that was registered through the **IDirectPlay8LobbiedApplication::RegisterProgram** method.

```
HRESULT UnRegisterProgram(
GUID* pguidApplication,
const DWORD dwFlags

);
```

Parameters

pguidApplication
Pointer to the GUID of the application to unregister.

dwFlags
Reserved. Must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::Update Status

Updates the status of a connected lobby client.

```
HRESULT UpdateStatus(
const DPNHANDLE hConnection,
const DWORD dwStatus,
const DWORD dwFlags
);
```

Parameters

hConnection
Variable of type **DPNHANDLE** that specifies the lobby client. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS
The status update will be sent to all lobby clients to which you are connected.

dwStatus
Variable of type **DWORD** that is filled with one of the following values that indicate the status between the lobby client and the lobbied application.

DPLSESSION_CONNECTED
The lobby client and lobbied application are currently connected.

DPLSESSION_COULDNOTCONNECT
The lobby client was not able to connect to the lobbied application.

DPLSESSION_DISCONNECTED

The lobby client and lobbied application are currently disconnected.

DPLSESSION_TERMINATED
The connection between the lobby client and lobbied application has been terminated.

DPLSESSION_HOSTMIGRATED
The peer object associated with the connection is involved in a session where a host migration takes place and the local client is not the new host.

DPLSESSION_HOSTMIGRATEDHERE
The peer object associated with the connection is involved in a session where a host migration takes place and the local client becomes the new host.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDHANDLE

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::GetConnectionSettings

Retrieves the set of connection settings for the specified connection. These settings can be set through a call to the **IDirectPlay8LobbyClient::ConnectApplication**, **IDirectPlay8LobbyClient::SetConnectionSettings**, or **IDirectPlay8LobbiedApplication::SetConnectionSettings** method.

When you get connection settings, a reference will be added for each address object that is returned to the user. Therefore, users must be sure to call **Release** on each address object when they are done with the structure.

```
HRESULT GetConnectionSettings(
const DPNHANDLE hLobbyClient,
DPL_CONNECTION_SETTINGS *const pdplSessionInfo,
DWORD* pdwInfoSize,
const DWORD dwFlags
);
```

Parameters

hLobbyClient
Handle to the connection for which to retrieve the settings.

pdplSessionInfo
Pointer to a **DPL_CONNECTION_SETTINGS** structure to receive the connection settings for the specified connection.

pdwInfoSize
Pointer to a **DWORD** containing the size, in bytes, of the buffer specified in the *pdplSessionInfo* structure. If the buffer is not large enough to hold the connection settings, DPNERR_BUFFERTOOSMALL is returned and this value will be set to the required buffer size. On success, this value will contain the number of bytes written to the specified buffer.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPARAM

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDOBJECT

DPNERR_INVALIDFLAGS

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8LobbiedApplication::SetConnectionSettings

Sets the connection settings to be associated with the specified connection. Calling this method generates a DPL_MSGID_CONNECTION_SETTINGS message to be sent to the client specified by *hConnection*.

When you set connection settings, the lobby application will add a reference to each of the address objects specified in the call.

```
HRESULT SetConnectionSettings(
const DPNHANDLE hConnection,
const DPL_CONNECTION_SETTINGS *const pdplConnectSettings,
const DWORD dwFlags
);
```

Parameters

hConnection
Handle to the connection to set the settings for. You may also specify the following flag.

DPLHANDLE_ALLAPPLICATIONS
The connection settings will be updated for all the lobby clients to which you are connected.

pdplConnectSettings
Pointer to a **DPL_CONNECTION_SETTINGS** structure containing the settings associated with the specified connection.

dwFlags
Reserved, must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDOBJECT

DPNERR_INVALIDFLAGS

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

IDirectPlay8Address

The **IDirectPlay8Address** interface contains generic addressing methods used to create and manipulate addresses for Microsoft® DirectPlay®. This interface is one of the interfaces available through the CLSID_DirectPlayAddress COM object. To create an object that supports this interface, use the **CoCreateInstanceEx** method for the CLSID CLSID_DirectPlayAddress that specifies the IID_IDirectPlayAddress8 interface.

The **IDirectPlay8Address** interface contains the following methods.

IDirectPlay8Address Methods	**BuildFromURLW**
	BuildFromURLA
	Duplicate
	SetEqual
	IsEqual
	Clear

GetURLW
GetURLA
GetSP
GetUserData
SetSP
SetUserData
GetNumComponents
GetComponentByName
GetComponentByIndex
AddComponent
GetDevice
SetDevice
BuildFromDPADDRESS

Remarks

In order to deliver messages, each participant in a multiplayer game must have a unique address. Addresses can refer either to the computer that your application is running on (*device address*), or a computer that your application needs to communicate with (*host address*).

DirectPlay represents addresses as URLs. These URLs are then encapsulated in the address object so that they can be passed to or from the DirectPlay API. In general, address URLs are strings that consist of three basic components in the following order: scheme, scheme separator, and data string.

All DirectPlay addresses use "x-directplay" as the scheme, and ":/" as the scheme separator. Using ":/" as a separator implies that the data that follows is *opaque.* In other words, the data string does not conform to any Internet standard, and should simply be passed on to the receiving application without modification. All DirectPlay URLs thus have the following general form:

x-directplay:/[*data string*]

There are two basic approaches to handling address objects:

- Handle the data string directly, using normal string manipulation techniques.
- Use the methods exposed by **IDirectPlay8Address** to obtain or modify the individual elements of the data string.

For more information on DirectPlay addresses, see DirectPlay Addressing.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::BuildFromURLW

Sets the object equal to the address in the DirectPlay 8 URL. It erases the contents of the object.

HRESULT BuildFromURLW(
WCHAR* *pwszSourceURL*
);

Parameters

pwszSourceURL
Pointer to a NULL-terminated Unicode string that contains a properly formatted DirectPlay 8 address.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPOINTER

DPNERR_INVALIDURL

DPNERR_NOTALLOWED

Remarks

The Dpaddr.h header file defines a number of standard strings that you can use to construct your URL instead using a literal string. All of the string names have the form DPNA_XXX. For example, DPNA_HEADER can be used in place of L"x-directplay:/" for the URL header.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::BuildFromURLA

Sets the object equal to the specified in the DirectPlay 8 URL. It erases the contents of the object.

HRESULT BuildFromURLA(
CHAR* *pszSourceURL*

);

Parameters

pszSourceURL
Pointer to a NULL-terminated ANSI string that contains a properly formatted DirectPlay 8 address.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPOINTER

DPNERR_INVALIDURL

DPNERR_NOTALLOWED

Remarks

The Dpaddr.h header file defines a number of standard strings that you can use to construct your URL instead using a literal string. All of the string names have the form DPNA_XXX. For example, DPNA_HEADER can be used in place of L"x-directplay:/" for the URL header.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::Duplicate

Creates a DirectPlay Address object that duplicates the address in this object.

HRESULT Duplicate(
PDIRECTPLAY8ADDRESS* *ppdpaNewAddress*
);

Parameters

ppdpaNewAddress
Address of a pointer to receive the **IDirectPlay8Address** pointer for the duplicate object. DirectPlay increments the reference count for this interface. You must release the interface when you no longer need it.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_GENERIC

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::SetEqual

Sets the contents of the object it is called on to match the contents of the address object passed to the method.

```
HRESULT SetEqual(
PDIRECTPLAY8ADDRESS pdpaAddress
);
```

Parameters

pdpaAddress
Pointer to a DirectPlay8Address object that this object will be set to.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDADDRESSFORMAT

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::IsEqual

Compares two addresses to see if they are equal.

```
HRESULT IsEqual(
PDIRECTPLAY8ADDRESS pdpaAddress
);
```

Parameters

pdpaAddress
Address to compare to the address contained within the object.

Return Values

If the method is successful, one of the following values is returned.

DPNSUCCESS_EQUAL
The two addresses are equal.

DPNSUCCESS_NOTEQUAL
The two addresses are not equal.

If the method fails, one of the following error values may be returned.

DPNERR_INVALIDADDRESSFORMAT

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPOINTER

Remarks

This method checks the contents of the address specified by the *pdpaAddress* parameter and compares it to the address contained within the object this method was called on. This method does not affect the contents of either address.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::Clear

Resets the address object to an empty address.

```
HRESULT Clear();
```

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetURLW

Retrieves the DirectPlay 8 address URL string represented by this object.

HRESULT GetURLW(
WCHAR* *pwszURL*,
PDWORD *pdwNumChars*
);

Parameters

pwszURL
Address of a pointer to receive the URL represented by this object. This parameter can be NULL if *pdwNumChars* points to a **DWORD** containing 0.

pdwNumChars
Pointer to a **DWORD** that contains the number of characters the specified buffer can hold, including NULL terminator. On success this value contains the number of characters written to the specified buffer, including NULL terminator. On failure this value contains the number of characters, including NULL terminator, required to hold the URL and the method returns DPNERR_BUFFERTOOSMALL.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_GENERIC

DPNERR_INVALIDURL

DPNERR_OUTOFMEMORY

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetURLA

Retrieves the Microsoft® DirectPlay® address URL string represented by this object. (ANSI Version.)

HRESULT GetURLA(
CHAR* *pszURL*,
PDWORD *pdwNumChars*
);

Parameters

pszURL
Address of a pointer to receive the URL represented by this object. This parameter can be NULL if *pdwNumChars* points to a **DWORD** containing 0.

pdwNumChars
Pointer to a **DWORD** that contains the number of characters the specified buffer can hold, including NULL terminator. On success this value contains the number of characters written to the specified buffer, including NULL terminator. On failure this value contains the number of characters, including NULL terminator, required to hold the URL and the method returns DPNERR_BUFFERTOOSMALL.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_GENERIC
DPNERR_INVALIDURL
DPNERR_OUTOFMEMORY

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetSP

Retrieves the service provider GUID in the address object. If no service provider is specified, this method returns DPNERR_DOESNOTEXIST.

```
HRESULT GetSP(
GUID* pguidSP
);
```

Parameters

pguidSP
Pointer to a GUID to receive the service provider in the address object.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_DOESNOTEXIST
DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetUserData

Retrieves the user data in the address object. If no user data exists in this address object, this method returns DPNERR_DOESNOTEXIST.

HRESULT GetUserData(
void* *pvUserData*,
PDWORD *pdwBufferSize*
);

Parameters

pvUserData
Pointer to a buffer to receive the user data from this address. To retrieve the required size, set this parameter to NULL and the **DWORD** in *pdwBufferSize* to 0.

pdwBufferSize
Size in bytes of the buffer pointed to by *pvUserData*. If *pvUserData* is NULL, this parameter must point to a **DWORD** containing 0. On output, the contained **DWORD** is set to the number of bytes written to the buffer. On failure, this contains the number of bytes required to retrieve the user data and the method returns DPNERR_BUFFERTOOSMALL.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_DOESNOTEXIST
DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::SetSP

Sets the service provider GUID in the address object. If a service provider is specified for this address, it is overwritten by this call.

HRESULT SetSP(

const GUID *const *pguidSP*
);

Parameters

pguidSP
Pointer to the service provider GUID.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPOINTER
DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::SetUserData

Sets the user data in the address object. If there is user data in this address, it is overwritten by this call.

HRESULT SetUserData(
const void *const *pvUserData*,
const DWORD *dwDataSize*
);

Parameters

pvUserData
Pointer to a buffer that contains the data to place in the user data section of the address. Set to NULL to clear the user data.

dwDataSize
Size, in bytes, of the data in *pvUserData*. If *pvUserData* is NULL, this must be 0.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPOINTER
DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetNumComponents

Retrieves the number of components in the address.

```
HRESULT GetNumComponents(
PDWORD pdwNumComponents
);
```

Parameters

pdwNumComponents
Pointer to a **DWORD** to receive the number of components in this address object.

Return Values

Returns S_OK if successful, or the following error value.

DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetComponentByName

Retrieves information on the component at the specified key. Values for the component are retrieved in their native format. If the component key is not found, DPNERR_DOESNOTEXIST is returned.

The value of the component is retrieved in its native format. Therefore, if the component's value is a **DWORD**, a **DWORD** is retrieved by this call. So buffer size = 4 and *pvBuffer* should be a recast **PDWORD**.

```
HRESULT GetComponentByName(
const WCHAR *const pwszName,
void* pvBuffer,
PDWORD pdwBufferSize,
```

PDWORD *pdwDataType*
);

Parameters

pwszName
String specifying the name of the component you want to retrieve.

pvBuffer
Buffer to retrieve the data stored in the value of the component. To retrieve the size required, specify NULL for this parameter and 0 for the **DWORD** pointed to by *pdwBufferSize*. The method returns DPNERR_BUFFERTOOSMALL in this case.

pdwBufferSize
On input, a pointer to a **DWORD** that contains the size of the buffer, in bytes, pointed to by *pvBuffer*. On output, a pointer to a **DWORD** that contains the number of bytes written to the buffer on success and on failure, the number of bytes required to store the data.

pdwDataType
DWORD pointed to by this parameter that is set to the type of data that is stored in this component. This can be one of the following:

DPNA_DATATYPE_STRING
Data is a NULL-terminated string.

DPNA_DATATYPE_DWORD
Data is a **DWORD**.

DPNA_DATATYPE_GUID
Data is a GUID.

DPNA_DATATYPE_BINARY
Data is raw binary.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_DOESNOTEXIST

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetComponentByIndex

Retrieves information on the component at the specified index. Values for the component are retrieved in their native format. If the component key is not found, the method returns DPNERR_DOESNOTEXIST.

The value of the component is retrieved in its native format. Therefore, if the component's value is a **DWORD**, a **DWORD** is retrieved by this call. So buffer size = 4 and *pvBuffer* should be a recast **PDWORD**.

```
HRESULT GetComponentByIndex(
const DWORD dwComponentID,
WCHAR* pwszName,
PDWORD pdwNameLen,
void* pvBuffer,
PDWORD pdwBufferSize,
PDWORD pdwDataType
);
```

Parameters

dwComponentID
Index of the component to retrieve. This value is zero-based and should be in the range of [0..GetNumComponents()-1].

pwszName
Buffer to retrieve the name of the component on a successful call. To retrieve the size required, specify NULL for this parameter and 0 for the **DWORD** pointed to by *pdwNameBufferSize*. The method returns DPNERR_BUFFERTOOSMAL in this case.

pdwNameLen
On input, a pointer to a **DWORD** that contains the size of the buffer, in characters including NULL terminator, pointed to by *pwszName*. On output, a pointer to a **DWORD** that contains the number of characters written to the buffer, including NULL terminator, on success and on failure, the number of characters required, including NULL terminator, to store this value.

pvBuffer
Buffer to retrieve the data stored in the value of the component. To retrieve the size required, specify NULL for this parameter and 0 for the **DWORD** pointed to by *pdwBufferSize*. The method returns DPNERR_BUFFERTOOSMALL in this case.

pdwBufferSize
On input, a pointer to a **DWORD** containing the size of the buffer, in bytes, pointed to by *pvBuffer*. On output, a pointer to a **DWORD** that contains the number of bytes written to the buffer on success and on failure, the number of bytes required to store the data.

pdwDataType

DWORD pointed to by this parameter that is set to the type of data that is stored in this component. This can be one of the following:

DPNA_DATATYPE_STRING
Data is a NULL-terminated string.

DPNA_DATATYPE_DWORD
Data is a **DWORD**.

DPNA_DATATYPE_GUID
Data is a GUID.

DPNA_DATATYPE_BINARY
Data is raw binary.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_DOESNOTEXIST

DPNERR_INVALIDPARAM

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::AddComponent

Adds a component to the address. If the component is part of the address, then it is replaced by the new value in this call.

Values are specified in native formats when making this call. Therefore, the *lpvData* parameter should be a recast pointer to a variable that holds the data in the native format. For example, if the component is a GUID, the *lpvData* parameter should be a recast pointer to a GUID.

This method validates that the predefined component types are the right format.

```
HRESULT AddComponent(
const WCHAR *const pwszName,
const void *const lpvData,
const DWORD dwDataSize,
const DWORD dwDataType
);
```

Parameters

pwszName
NULL-terminated Unicode string that contains the key for the component.

lpvData
Pointer to a buffer that contains the value associated with the specified key. Data should be specified in its native format.

dwDataSize
Size, in bytes, of the data in the buffer located at *lpvData*. The size depends on the data type. If the size is not specified correctly, the method returns DPNERR_INVALIDPARAM.

DWORD
Size = sizeof(DWORD)

GUID
Size = sizeof(GUID)

String
Size = size of the string in bytes, including NULL terminator.

dwDataType
Data type of the value associated with this key. The data type can be one of the following:

DPNA_DATATYPE_STRING
Data is a NULL-terminated string.

DPNA_DATATYPE_DWORD
Data is a **DWORD**.

DPNA_DATATYPE_GUID
Data is a GUID.

DPNA_DATATYPE_BINARY
Data is in raw binary format.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::GetDevice

Retrieves the local device GUID in the address object. If no device is specified, this method returns DPNERR_DOESNOTEXIST.

HRESULT GetDevice(

GUID* *pguidDevice*
);

Parameters

pguidDevice
Pointer to a GUID to receive the device in the address object.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_DOESNOTEXIST
DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::SetDevice

Sets the local device GUID in the address object. If a local device is specified for this address, it is overwritten by this call.

HRESULT SetDevice(
const GUID *const *pguidDevice*
);

Parameters

pguidDevice
Pointer to a GUID of the local device.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPOINTER
DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8Address::BuildFromDPADDRESS

Sets the current object's internal address to be the DirectPlay 8 equivalent of the specified DirectPlay 4 address. The purpose of this method is to allow lobby developers to launch games with the new Microsoft® DirectPlay® interface using the old lobby code.

This method enumerates the address components in the specified address and adds the corresponding element to the DirectPlay 8 address.

```
HRESULT BuildFromDPADDRESS(
LPVOID pvAddress,
DWORD dwDataSize
);
```

Parameters

pvAddress
Pointer to a DirectPlay4 address that will be converted to the DirectPlay 8 address format.

dwDataSize
Size of data contained in the *pvAddress* parameter.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDADDRESSFORMAT

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPOINTER

Remarks

This method builds a DirectPlay 8 address from a DirectPlay4 address. This method will clear the current address of all elements before building the new address.

This method has the following limitations.

- The method cannot map the DPAID_Modem address element because DirectPlay 4 used modem names, while DirectPlay 8 uses GUIDs to identify modems.
- Elements of the DirectPlay 4 address that are not part of the predefined DirectPlay 4 address elements will result in an error and a return value of DPNERR_INVALIDADDRESSFORMAT. See DirectPlay 4 documentation on DirectPlay addresses for a complete list of the DirectPlay 4 address elements.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP

The **IDirectPlay8AddressIP** interface is available through the CLSID_DirectPlayAddress COM object. This interface is used for IP provider-specific addressing services.

The **IDirectPlay8AddressIP** interface contains the following methods.

IDirectPlay8AddressIP Methods	**BuildFromSockAddr**
	BuildAddress
	BuildLocalAddress
	GetSockAddress
	GetLocalAddress
	GetAddress

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::BuildFromSockAddr

Builds a remote DirectPlay 8 IP address from a valid **SOCKADDR** structure. The **SOCKADDR** structure must specify an IP address. If the address is not in the correct format, DPNERR_INVALIDPARAM is returned.

The result of a successful call is a valid remote address with the following elements.

- DPNA_KEY_PROVIDER = CLSID_DP8SP_TCPIP
- DPNA_KEY_HOSTNAME = specified host name
- DPNA_KEY_PORT = specified port

All addressing information contained in the object before the call is erased.

```
HRESULT BuildFromSockAddr(
const SOCKADDR *const pSockAddr
);
```

Parameters

pSockAddr
Valid UDP address specified in **SOCKADDR** form.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER
DPNERR_NOTALLOWED

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::BuildAddress

Builds a remote DirectPlay 8 IP address from a host name and a port. The result of a successful call is a valid remote address with the following elements.

- DPNA_KEY_PROVIDER = CLSID_DP8SP_TCPIP
- DPNA_KEY_HOSTNAME = specified host name
- DPNA_KEY_PORT = specified port

All addressing information contained in the object before the call is erased.

```
HRESULT BuildAddress(
const WCHAR *const wszAddress,
const USHORT usPort
);
```

Parameters

wszAddress
Remote host address can be a dotted Internet address—for example, 127.0.0.1—or a valid host name—for example, example.microsoft.com.

usPort
Port on the remote host to which to connect.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER

DPNERR_NOTALLOWED

Remarks

Note

The DPNSVR is a DirectPlay feature that allows multiple processes to share a single port for enumeration. Do not use the DPNA_DPNSVR_PORT flag when constructing a device address, or when making a connection. This flag should only be used for enumerations. If you do not add a port element to the enumeration address, the port represented by the flag will be automatically added to that address. See Using the DirectPlay DPNSVR Application for a further discussion of DPNSVR.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::BuildLocalAddress

Builds a local DirectPlay 8 IP address from a device and port. The result of a successful call is a valid remote address with the following elements.

- DPNA_KEY_PROVIDER = CLSID_DP8SP_TCPIP
- DPNA_KEY_DEVICE= specified device
- DPNA_KEY_PORT = specified port

All addressing information contained in the object before the call is erased.

HRESULT BuildLocalAddress(
const GUID *const *pguidAdapter*,
const USHORT *usPort*
);

Parameters

pguidAdapter
Local device identifier to host on.

usPort
Port on the local device to host on. This value can be set to 0 to allow DirectPlay 8.0 to automatically select the port.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER
DPNERR_NOTALLOWED

Remarks

Note

The DPNSVR is a DirectPlay feature that allows multiple processes to share a single port for enumeration. Do not use the DPNA_DPNSVR_PORT flag when constructing a device address, or when making a connection. This flag should only be used for enumerations. If you do not add a port element to the enumeration address, the port represented by the flag will be automatically added to that address. See Using the DirectPlay DPNSVR Application for a further discussion of DPNSVR.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::GetSockAddress

Retrieves a list of **SOCKADDR** structures describing the addresses represented by this object. If the host name specified in the object requires a DNS lookup, it is performed. Therefore, this method may block while the DNS is queried. It is also possible for a host name to resolve to multiple addresses.

To succeed, the contained address must have at least the following elements.

- DPNA_KEY_PROVIDER
- DPNA_KEY_HOSTNAME
- DPNA_KEY_PORT = specified port

HRESULT GetSockAddress(
SOCKADDR* *psockAddress*,
PDWORD *pdwAddressBufferSize*
);

Parameters

psockAddress
Pointer to buffer to retrieve the array of **SOCKADDR** structures. There is one **SOCKADDR** structure for each address the host resolves to.

pdwAddressBufferSize
Size, in bytes, of the buffer specified in *psockAddresses*. On success, this parameter contains the number of bytes written to the specified buffer. On failure, this parameter contains the number of bytes required to retrieve the array of **SOCKADDR** structures. You can divide the value of this parameter by the

size of the SOCKADDR structure to determine the number of items present in the returned array.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::GetLocalAddress

Retrieves the local address information from a DirectPlay 8 IP address. To succeed, the contained address must have at least the following elements.

- DPNA_KEY_PROVIDER
- DPNA_KEY_DEVICE
- DPNA_KEY_PORT

HRESULT GetLocalAddress(
GUID* *pguidAdapter*,
USHORT* *pusPort*
);

Parameters

pguidAdapter
Pointer to a GUID to retrieve the GUID of the local device specified in this address.

pusPort
Pointer to a **USHORT** to contain the port specified in this local address.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

IDirectPlay8AddressIP::GetAddress

Retrieves the remote address information from a remote DirectPlay 8 IP address. To succeed, the contained address must have at least the following elements.

- DPNA_KEY_PROVIDER
- DPNA_KEY_HOSTNAME
- DPNA_KEY_PORT = specified port

HRESULT GetAddress(
WCHAR* *wszAddress*,
PDWORD *pdwAddressLength*,
USHORT* *psPort*
);

Parameters

wszAddress
Pointer to a buffer to receive the host name. This parameter can be NULL to retrieve the required size.

pdwAddressLength
Size, in characters, of the buffer specified in *wszAddress,* including NULL terminator. On success, this parameter contains the number of characters, including NULL terminator, written to the specified buffer. On failure, this parameter contains the number of characters, including NULL terminator, required to retrieve the host name.

psPort
Pointer to a **USHORT** to contain the port specified in this local address.

Return Values

Returns S_OK if successful, or one of the following error values.

DPNERR_BUFFERTOOSMALL

DPNERR_INVALIDPARAM

DPNERR_INVALIDPOINTER

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

Functions

The Microsoft® DirectPlay® functions are:

- **DirectPlay8Create**
- **DirectPlay8LobbyCreate**
- **DirectPlay8AddressCreate**
- **DirectPlayVoiceCreate**

DirectPlay8Create

The **DirectPlay8Create** function is an external creation function used to create interfaces defined in the Dplay8.h header file.

```
HRESULT WINAPI DirectPlay8Create(
  GUID* pcIID,
  void** ppvInterface,
  IUnknown* pUnknown
);
```

Parameters

pcIID
Pointer to the interface ID you want to create. You may specify the IID_IDirectPlay8Client, IID_IDirectPlay8Server or IID_IDirectPlay8Peer interface ID.

ppvInterface
Address of a variable to receive the new interface pointer.

pUnknown
Address of the controlling object's IUnknown interface for COM aggregation. Must be NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DirectPlay8LobbyCreate

The **DirectPlay8LobbyCreate** function is an external creation function used to create one of the lobby interfaces defined in the Dplobby8.h header file.

```
HRESULT WINAPI DirectPlay8LobbyCreate(
```

```
  GUID* pcIID,
  void** ppvInterface,
  IUnknown* pUnknown
);
```

Parameters

pcIID
Pointer to the interface ID you want to create. You may specify the IID_IDirectPlay8LobbiedApplication or IID_IDirectPlay8LobbyClient interface ID.

ppvInterface
Address of a variable to receive the new interface pointer.

pUnknown
Address of the controlling object's IUnknown interface for COM aggregation. Must be NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DirectPlay8AddressCreate

The **DirectPlay8AddressCreate** function is an external creation function used to create one of the addressing interfaces defined in the Dpaddr.h header file.

```
HRESULT WINAPI DirectPlay8AddressCreate(
  GUID* pcIID,
  void** ppvInterface,
  IUnknown* pUnknown
);
```

Parameters

pcIID
Pointer to the interface ID you want to create. You may specify the IID_IDirectPlay8Address or IID_IDirectPlay8AddressIP interface ID.

ppvInterface
Address of a variable to receive the new interface pointer.

pUnknown
Address of the controlling object's IUnknown interface for COM aggregation. Must be NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dpaddr.h.

DirectPlayVoiceCreate

The **DirectPlayVoiceCreate** function is an external creation function used to create one of the interfaces used in Microsoft® DirectPlay® Voice that are defined in the Dvoice.h header file.

```
HRESULT WINAPI DirectPlayVoiceCreate(
  GUID* pcIID,
  void** ppvInterface,
  IUnknown* pUnknown
);
```

Parameters

pcIID
Pointer to the interface ID you want to create. You may specify the IID_IDirectPlayVoiceTest, IID_IDirectPlayVoiceClient or IID_IDirectPlayVoiceServer interface ID.

ppvInterface
Address of a variable to receive the new interface pointer.

pUnknown
Address of the controlling object's IUnknown interface for COM aggregation. Must be NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

Callback Functions

The Microsoft® DirectPlay® callback functions are:

- **PFNDPNMESSAGEHANDLER**
- **PDVMESSAGEHANDLER**

PFNDPNMESSAGEHANDLER

PFNDPNMESSAGEHANDLER is an application-defined callback function used by the **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server** **IDirectPlay8LobbyClient** and **IDirectPlay8LobbiedApplication** interfaces to process messages.

```
typedef HRESULT (WINAPI *PFNDPNMESSAGEHANDLER)(
  PVOID pvUserContext,
  DWORD dwMessageType,
  PVOID pMessage
);
```

Parameters

pvUserContext

Pointer to the application-defined structure that will be passed to this callback function. This is defined in the *pvUserContext* parameter of the **Initialize** method.

dwMessageType

One of the following message types that are generated by the **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server** interfaces. Each interface uses a different subset of the available messages. Refer to the interface documentation for details.

DPN_MSGID_ADD_PLAYER_TO_GROUP

DPN_MSGID_ASYNC_OP_COMPLETE

DPN_MSGID_CLIENT_INFO

DPN_MSGID_CONNECT_COMPLETE

DPN_MSGID_CREATE_GROUP

DPN_MSGID_CREATE_PLAYER

DPN_MSGID_DESTROY_GROUP

DPN_MSGID_DESTROY_PLAYER

DPN_MSGID_ENUM_HOSTS_QUERY

DPN_MSGID_ENUM_HOSTS_RESPONSE

DPN_MSGID_GROUP_INFO

DPN_MSGID_HOST_MIGRATE

DPN_MSGID_INDICATE_CONNECT

DPN_MSGID_INDICATED_CONNECT_ABORTED

DPN_MSGID_PEER_INFO

DPN_MSGID_RECEIVE

DPN_MSGID_REMOVE_PLAYER_FROM_GROUP

DPN_MSGID_RETURN_BUFFER

DPN_MSGID_SEND_COMPLETE

DPN_MSGID_SERVER_INFO

DPN_MSGID_TERMINATE_SESSION

Additionally, if the application supports Microsoft® DirectPlay® lobby functionality, this parameter can specify one of the following message types that are generated by the **IDirectPlay8LobbyClient** and **IDirectPlay8LobbiedApplication** interfaces. Each interface uses a different subset of the available messages. Refer to the interface documentation for details.

DPL_MSGID_CONNECT

DPL_MSGID_CONNECTION_SETTINGS

DPL_MSGID_DISCONNECT

DPL_MSGID_RECEIVE

DPL_MSGID_SESSION_STATUS

pMessage

Structure containing message information.

Return Values

See the documentation for the individual messages for appropriate return values. Unless otherwise noted, this function should return S_OK.

Remarks

This function must be threadsafe because it might be called reentrantly through multiple threads.

Callback messages from the same player are serialized. Once you receive a message from a player, you will not receive another until you have handled the first message, and the callback function has returned.

The message structures have the same name as the message type except the "DPN_MSGID" is replaces with "DPNMSG". For example, the **DPN_MSGID_CONNECTION_TERMINATED** message type uses the **DPNMSG_CONNECTION_TERMINATED** message structure to convey the actual message information.

When implementing this callback function, first look at the message type returned in the *dwMessageType* parameter and then cast the message structure (*pMessage*) to that type to obtain message information. Some messages don't have a defined structure because they have no parameters. For these messages, the *pMessage* parameter is NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

PDVMESSAGEHANDLER

PDVMESSAGEHANDLER is an application-defined callback function used by the **IDirectPlayVoiceClient** and **IDirectPlayVoiceServer** interfaces to send messages to the user.

```
typedef HRESULT (FAR PASCAL *PDVMESSAGEHANDLER)(
  LPVOID pvUserContext,
  DWORD dwMessageType,
  LPVOID lpMessage
);
```

Parameters

pvUserContext
Pointer to the application-defined structure that will be passed to this callback function. This is defined in the *lpUserContext* parameter of the **IDirectPlayVoiceServer::Initialize** and **IDirectPlayVoiceClient::Initialize** methods.

dwMessageType
One of the following message types.

DVMSGID_CONNECTRESULT
DVMSGID_CREATEVOICEPLAYER
DVMSGID_DELETEVOICEPLAYER
DVMSGID_DISCONNECTRESULT
DVMSGID_GAINFOCUS
DVMSGID_HOSTMIGRATED
DVMSGID_INPUTLEVEL
DVMSGID_LOCALHOSTSETUP
DVMSGID_LOSTFOCUS
DVMSGID_OUTPUTLEVEL
DVMSGID_PLAYEROUTPUTLEVEL
DVMSGID_PLAYERVOICESTART
DVMSGID_PLAYERVOICESTOP
DVMSGID_RECORDSTART
DVMSGID_RECORDSTOP
DVMSGID_SESSIONLOST
DVMSGID_SETTARGETS

lpMessage
Structure containing message information.

Return Values

See the documentation for the individual messages for appropriate return values. Unless otherwise noted, this function should return DV_OK.

Remarks

When implementing this callback function, you must first look at the message type returned in the *dwMessageType* parameter and then cast the message structure (*lpMessage*) to that type to obtain message information. Some messages don't have a defined structure because they have no parameters. For these messages, the *lpMessage* parameter is NULL.

Note

This function may be called on multiple different threads at the same time. It must thus be threadsafe and reentrant.

All message structures have the same name as the corresponding message types except the prefix is DVMSG_ instead of DVMSGID_. For example, the structure for **DVMSGID_RECORDSTART** is **DVMSG_RECORDSTART**.

The structure sent to the message handler is valid only for the duration of the call. Therefore, if you want to use any of the information passed into the function after the handler function has returned you must make a copy of the data.

Callback messages from the same player are serialized. Once you receive a message from a player, you will not receive another until you have handled the first message, and the callback function has returned.

Only messages that are specified in the message mask through a call to the **IDirectPlayVoiceClient::Initialize**, **IDirectPlayVoiceServer::Initialize**, **IDirectPlayVoiceClient::SetNotifyMask** and **IDirectPlayVoiceServer::SetNotifyMask** methods are sent to this callback function.

The **DVMSGID_GAINFOCUS** and **DVMSGID_LOSTFOCUS** message structures have not been implemented in this release of Microsoft® DirectPlay®.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

System Messages

Microsoft® DirectPlay® messages are received by a DirectPlay callback message handler. DirectPlay uses these messages to convey information from the system to a DirectPlay application.

- DirectPlay Server Messages
- DirectPlay Client Messages
- DirectPlay Lobby Messages
- DirectPlay Voice Messages

DirectPlay Server Messages

The following messages should be processed by all Microsoft® DirectPlay® server callback message handlers.

- **DPN_MSGID_CREATE_PLAYER**
- **DPN_MSGID_DESTROY_PLAYER**
- **DPN_MSGID_INDICATE_CONNECT**
- **DPN_MSGID_INDICATED_CONNECT_ABORTED**
- **DPN_MSGID_RECEIVE**
- **DPN_MSGID_RETURN_BUFFER**

The following messages can be processed by DirectPlay server callback message handlers, but are not required.

- **DPN_MSGID_ADD_PLAYER_TO_GROUP**
- **DPN_MSGID_ASYNC_OP_COMPLETE**
- **DPN_MSGID_CLIENT_INFO**
- **DPN_MSGID_CREATE_GROUP**
- **DPN_MSGID_DESTROY_GROUP**
- **DPN_MSGID_GROUP_INFO**
- **DPN_MSGID_ENUM_HOSTS_QUERY**
- **DPN_MSGID_REMOVE_PLAYER_FROM_GROUP**
- **DPN_MSGID_SEND_COMPLETE**
- **DPN_MSGID_SERVER_INFO**

DirectPlay Client Messages

The following messages should be processed by all DirectPlay client callback message handlers.

- **DPN_MSGID_RECEIVE**

- **DPN_MSGID_TERMINATE_SESSION**
- **DPN_MSGID_RETURN_BUFFER**

The following messages can be processed by DirectPlay client callback message handlers, but are not required.

- **DPN_MSGID_ASYNC_OP_COMPLETE**
- **DPN_MSGID_CLIENT_INFO**
- **DPN_MSGID_CONNECT_COMPLETE**
- **DPN_MSGID_ENUM_HOSTS_RESPONSE**
- **DPN_MSGID_SEND_COMPLETE**
- **DPN_MSGID_SERVER_INFO**
- **DPN_MSGID_GROUP_INFO**

DirectPlay Lobby Messages

The following messages are handled by lobby client and lobbied application callback message handlers.

- **DPL_MSGID_CONNECT**
- **DPL_MSGID_CONNECTION_SETTINGS**
- **DPL_MSGID_DISCONNECT**
- **DPL_MSGID_RECEIVE**
- **DPL_MSGID_SESSION_STATUS**

DirectPlay Voice Messages

The following messages are handled by Microsoft® DirectPlay® voice callback message handlers.

- **DVMSGID_CONNECTRESULT**
- **DVMSGID_CREATEVOICEPLAYER**
- **DVMSGID_DELETEVOICEPLAYER**
- **DVMSGID_DISCONNECTRESULT**
- **DVMSGID_GAINFOCUS**
- **DVMSGID_HOSTMIGRATED**
- **DVMSGID_INPUTLEVEL**
- **DVMSGID_LOCALHOSTSETUP**
- **DVMSGID_LOSTFOCUS**
- **DVMSGID_OUTPUTLEVEL**
- **DVMSGID_PLAYEROUTPUTLEVEL**

- **DVMSGID_PLAYERVOICESTART**
- **DVMSGID_PLAYERVOICESTOP**
- **DVMSGID_RECORDSTART**
- **DVMSGID_RECORDSTOP**
- **DVMSGID_SESSIONLOST**
- **DVMSGID_SETTARGETS**

DPL_MSGID_CONNECT

Microsoft® DirectPlay® generates a DPL_MSGID_CONNECT message when a lobby client connects to the lobbied application through the **IDirectPlay8LobbyClient::ConnectApplication** method.

DPL_MESSAGE_CONNECT

The **DPL_MESSAGE_CONNECT** structure is passed with the DPL_MSGID_CONNECT message.

```
typedef struct _DPL_MESSAGE_CONNECT{
    DWORD     dwSize;
    DPNHANDLE hConnectId;
    PDPL_CONNECTION_SETTINGS  pdplConnectionSettings;
    PVOID     pvLobbyConnectData;
    DWORD     dwLobbyConnectDataSize;
    PVOID     pvConnectionContext;
} DPL_MESSAGE_CONNECT, *PDPL_MESSAGE_CONNECT;
```

dwSize
Size of the **DPL_MESSAGE_CONNECT** message structure. The application must set this member before it uses the structure.

hConnectId
Handle used to identify the connection. This handle is used in subsequent calls to **IDirectPlay8LobbyClient::Send** and **IDirectPlay8LobbyClient::ReleaseApplication**.

pdplConnectionSettings
Pointer to a **DPL_CONNECTION_SETTINGS** structure with connection information.

pvLobbyConnectData
Pointer to lobby connection data.

dwLobbyConnectDataSize
Variable of type DWORD specifying the size of the data contained in the **pvLobbyConnectData** member.

pvConnectionContext

Context value associated with this connection. For lobbied applications, set this parameter when this message is received in your message handler to associate the context value with the connection. This may be set to NULL to disable context values.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_MSGID_CONNECTION_SETTINGS

The DPL_MSGID_CONNECTION_SETTINGS message is sent from the lobby client to the lobby application when **IDirectPlay8LobbyClient::SetConnectionSettings** is called. It is also sent from the lobby application to the lobby client when **IDirectPlay8LobbiedApplication::SetConnectionSettings** is called. The contents of the message are valid only for the duration of the message callback. Therefore, if you want to use the data contained in the message, you must make a copy before returning. In addition, if you want to use the addressing objects you must call **AddRef** on each address to ensure you retain a reference.

DPL_MESSAGE_CONNECTION_SETTINGS

The **DPL_MESSAGE_CONNECTION_SETTINGS** structure is passed with the DPL_MSGID_CONNECTION_SETTINGS message.

```
typedef struct _DPL_MESSAGE_CONNECTION_SETTINGS{
    DWORD               dwSize;
    DPNHANDLE           hSender;
    PDPL_CONNECTION_SETTINGS   pdplConnectionSettings;
    PVOID               pvConnectionContext
} DPL_MESSAGE_CONNECTION_SETTINGS,
*PDPL_MESSAGE_CONNECTION_SETTINGS;
```

dwSize
Contains the size of the **DPL_MESSAGE_CONNECTION_SETTINGS** structure. It should be set to **sizeof**(DPL_MESSAGE_CONNECTION_SETTINGS).

hSender
Contains the handle for the connection that sent this message.

pdplConnectionSettings

Contains a pointer to a **DPL_CONNECTION_SETTINGS** structure describing the connection settings for the specified connection.

pvConnectionContext
Pointer to a context value that has been set for the connection.

Remarks

For lobbied applications, the context value is set through the **pvConnectionContext** member of the **DPL_MESSAGE_CONNECT** message structure. When your message handler receives this message, whatever you set this member to before returning will be the context value for that connection.

For lobby clients, the *pvConnectionContext* parameter in the **IDirectPlay8LobbyClient::ConnectApplication** method will be used as the connection's context value if the connection is successful.

Context values are not shared between lobby client and lobbied application. For example, if you set your context value for a lobby connection in your **IDirectPlay8LobbyClient** interface to pointer A and in your **IDirectPlay8LobbiedApplication** interface you set it to pointer B, indications in your **IDirectPlay8LobbyClient** interface will have pointer A as their context value and in your **IDirectPlay8LobbiedApplication** interface pointer B will be the context value.

You can also set your context values to NULL if you do not want to use this feature.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_MSGID_DISCONNECT

Microsoft® DirectPlay® generates a DPL_MSGID_DISCONNECT message when a lobby client disconnects from a lobbied application through the **IDirectPlay8LobbyClient::ReleaseApplication** method.

DPL_MESSAGE_DISCONNECT

The **DPL_MESSAGE_DISCONNECT** structure contains information for the DPL_MSGID_DISCONNECT system message.

```
typedef struct _DPL_MESSAGE_DISCONNECT{
    DWORD     dwSize;
    DPNHANDLE hDisconnectId;
    HRESULT   hrReason;
    PVOID     pvConnectionContext;
```

```
} DPL_MESSAGE_DISCONNECT, *PDPL_MESSAGE_DISCONNECT;
```

dwSize
Size of the **DPL_MESSAGE_DISCONNECT** message structure. The application must set this member before it uses the structure.

hDisconnectId
Handle specifying the disconnection ID.

hrReason
Reason for the disconnection.

S_OK
It was a standard disconnection.

DPNERR_CONNECTIONLOST
This will be set if the process running the client or application exited abnormally.

pvConnectionContext
Context value that has been set for the connection.

Remarks

For lobbied applications, the context value is set through the **pvConnectionContext** member of the **DPL_MESSAGE_CONNECT** message structure. When your message handler receives this message, whatever you set this member to before returning will be the context value for that connection.

For lobby clients, the *pvConnectionContext* parameter in the **IDirectPlay8LobbyClient::ConnectApplication** method will be used as the connection's context value if the connection is successful.

Context values are not shared between lobby client and lobbied application. For example, if you set your context value for a lobby connection in your **IDirectPlay8LobbyClient** interface to pointer A and in your **IDirectPlay8LobbiedApplication** interface you set it to pointer B, indications in your **IDirectPlay8LobbyClient** interface will have pointer A as their context value and in your **IDirectPlay8LobbiedApplication** interface pointer B will be the context value.

You can also set your context values to NULL if you do not want to use this feature.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_MSGID_RECEIVE

Microsoft® DirectPlay® generates the DPL_MSGID_RECEIVE message when the target receives a message sent by the **IDirectPlay8LobbyClient::Send** or **IDirectPlay8LobbiedApplication::Send** method.

DPL_MESSAGE_RECEIVE

The **DPL_MESSAGE_RECEIVE** structure contains information for the DPL_MSGID_RECEIVE system message.

```
typedef struct _DPL_MESSAGE_RECEIVE{
    DWORD     dwSize;
    DPNHANDLE hSender;
    BYTE*     pBuffer;
    DWORD     dwBufferSize;
    PVOID     pvConnectionContext;
} DPL_MESSAGE_RECEIVE, *PDPL_MESSAGE_RECEIVE;
```

dwSize
Size of the **DPL_MESSAGE_RECEIVE** message structure. The application must set this member before it uses the structure.

hSender
Handle of the client that sent the message.

pBuffer
Pointer to message data.

dwBufferSize
Size of the message data contained in the **pBuffer** member.

pvConnectionContext
Context value that has been set for the connection.

Remarks

For lobbied applications, the context value is set through the **pvConnectionContext** member of the **DPL_MESSAGE_CONNECT** message structure. When your message handler receives this message, whatever you set this member to before returning will be the context value for that connection.

For lobby clients, the *pvConnectionContext* parameter in the **IDirectPlay8LobbyClient::ConnectApplication** method will be used as the connection's context value if the connection is successful.

Context values are not shared between lobby client and lobbied application. For example, if you set your context value for a lobby connection in your **IDirectPlay8LobbyClient** interface to pointer A and in your **IDirectPlay8LobbiedApplication** interface you set it to pointer B, indications in your **IDirectPlay8LobbyClient** interface will have pointer A as their context value

and in your **IDirectPlay8LobbiedApplication** interface pointer B will be the context value.

You can also set your context values to NULL if you do not want to use this feature.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_MSGID_SESSION_STATUS

Microsoft® DirectPlay® generates the DPL_MSGID_SESSION_STATUS message when the session has been updated with a call to the **IDirectPlay8LobbiedApplication::UpdateStatus** method.

DPL_MESSAGE_SESSION_STATUS

The **DPL_MESSAGE_SESSION_STATUS** structure contains information for the DPL_MSGID_SESSION_STATUS system message.

```
typedef struct _DPL_MESSAGE_SESSION_STATUS{
    DWORD dwSize;
    DPNHANDLE hSender;
    DWORD dwStatus;
    PVOID pvConnectionContext;
} DPL_MESSAGE_SESSION_STATUS,
*PDPL_MESSAGE_SESSION_STATUS;
```

dwSize
Size of the **DPL_MESSAGE_SESSION_STATUS** message structure. The application must set this member before it uses the structure.

hSender
The handle of the application that sent the status update message.

dwStatus
Updated status of the session. This member can be set to one of the following values.

DPLSESSION_CONNECTED
The lobbied application is currently connected to a session.

DPLSESSION_COULDNOTCONNECT
The lobbied application could not connect to the session.

DPLSESSION_DISCONNECTED
The lobbied application is currently disconnected from the session.

DPLSESSION_TERMINATED

The connection between session host and the lobbied application has been terminated.

DPLSESSION_HOSTMIGRATED
The host of a peer-to-peer session has migrated. The local client is not the new host.

DPLSESSION_HOSTMIGRATEDHERE
The host of a peer-to-peer session has migrated. The local client is the new host.

pvConnectionContext
Context value that has been set for the connection.

Remarks

For lobbied applications, the context value is set through the **pvConnectionContext** member of the **DPL_MESSAGE_CONNECT** message structure. When your message handler receives this message, whatever you set this member to before returning will be the context value for that connection.

For lobby clients, the *pvConnectionContext* in the **IDirectPlay8LobbyClient::ConnectApplication** method will be used as the connection's context value if the connection is successful.

Context values are not shared between lobby client and lobbied application. For example, if you set your context value for a lobby connection in your **IDirectPlay8LobbyClient** interface to pointer A and in your **IDirectPlay8LobbiedApplication** interface you set it to pointer B, indications in your **IDirectPlay8LobbyClient** interface will have pointer A as their context value and in your **IDirectPlay8LobbiedApplication** interface pointer B will be the context value.

You can also set your context values to NULL if you do not want to use this feature.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPN_MSGID_APPLICATION_DESC

This message indicates that the application description has been changed. There is no accompanying structure. To determine the new application description, call the **GetApplicationDesc** method exposed by **IDirectPlay8Peer**, **IDirectPlay8Client**, or **IDirectPlay8Server** interfaces.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPN_MSGID_ADD_PLAYER_TO_GROUP

Microsoft® DirectPlay® generates the DPN_MSGID_ADD_PLAYER_TO_GROUP message when a player has been added to a group in a peer-to-peer or client/server session.

DPNMSG_ADD_PLAYER_TO_GROUP

The **DPNMSG_ADD_PLAYER_TO_GROUP** structure contains information for the DPN_MSGID_ADD_PLAYER_TO_GROUP system message.

```
typedef struct _DPNMSG_ADD_PLAYER_TO_GROUP{
    DWORD  dwSize;
    DPNID  dpnidGroup;
    PVOID  pvGroupContext;
    DPNID  dpnidPlayer;
    PVOID  pvPlayerContext;
} DPNMSG_ADD_PLAYER_TO_GROUP,
*PDPNMSG_ADD_PLAYER_TO_GROUP;
```

dwSize
Size of this structure.

dpnidGroup
DPNID of the group to add the player.

pvGroupContext
Group context value.

dpnidPlayer
DPNID of the player added to the group.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_ASYNC_OP_COMPLETE

Microsoft® DirectPlay® generates the DPN_MSGID_ASYNC_OP_COMPLETE message when an asynchronous request has completed.

DPNMSG_ASYNC_OP_COMPLETE

The **DPNMSG_ASYNC_OP_COMPLETE** structure contains information for the DPN_MSGID_ASYNC_OP_COMPLETE system message.

```
typedef struct _DPNMSG_ASYNC_OP_COMPLETE{
    DWORD      dwSize;
    DPNHANDLE  hAsyncOp;
    PVOID      pvUserContext;
    HRESULT    hResultCode;
} DPNMSG_ASYNC_OP_COMPLETE,
*PDPNMSG_ASYNC_OP_COMPLETE;
```

dwSize
Size of this structure.

hAsyncOp
Asynchronous operation handle.

pvUserContext
Supplied user context.

hResultCode
HRESULT indicating the result of the asynchronous operation.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_CLIENT_INFO

Microsoft® DirectPlay® generates the DPN_MSGID_CLIENT_INFO message when client data is modified during a client/server session.

DPNMSG_CLIENT_INFO

The **DPNMSG_CLIENT_INFO** structure contains information for the DPN_MSGID_CLIENT_INFO system message.

```
typedef struct _DPNMSG_CLIENT_INFO{
```

```
    DWORD  dwSize;
    DPNID  dpnidClient;
    PVOID  pvPlayerContext;
} DPNMSG_CLIENT_INFO, *PDPNMSG_CLIENT_INFO;
```

dwSize
Size of this structure.

dpnidClient
DPNID of the client for client information.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_CONNECT_COMPLETE

Microsoft® DirectPlay® generates the DPN_MSGID_CONNECT_COMPLETE message when the connection attempt has been completed in a peer-to-peer or client/server session. This message is generated whether or not the connection was successful.

DPNMSG_CONNECT_COMPLETE

The **DPNMSG_CONNECT_COMPLETE** structure contains information for the DPN_MSGID_CONNECT_COMPLETE system message.

```
typedef struct _DPNMSG_CONNECT_COMPLETE{
    DWORD      dwSize;
    DPNHANDLE  hAsyncOp;
    PVOID      pvUserContext;
    HRESULT    hResultCode;
    PVOID      pvApplicationReplyData;
    DWORD      dwApplicationReplyDataSize;
} DPNMSG_CONNECT_COMPLETE, *PDPNMSG_CONNECT_COMPLETE;
```

dwSize
Size of this structure.

hAsyncOp
Asynchronous operation handle.

pvUserContext

User context supplied when the **IDirectPlay8Peer::Connect** or **IDirectPlay8Client::Connect** methods are called.

hResultCode
HRESULT describing the result of the connection attempt. See the Return Values section in the **IDirectPlay8Peer::Connect** or **IDirectPlay8Client::Connect** method for more information. Additionally, DPNERR_PLAYERNOTREACHABLE will be returned if a player has tried to join a peer-to-peer session where at least one other existing player in the session cannot connect to the joining player.

pvApplicationReplyData
Connection reply data returned from the host or server.

dwApplicationReplyDataSize
Size of the data, in bytes, of the **pvApplicationReplyData** member.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_CREATE_GROUP

Microsoft® DirectPlay® generates the DPN_MSGID_CREATE_GROUP message when a group is created.

DPNMSG_CREATE_GROUP

The **DPNMSG_CREATE_GROUP** structure contains information for the DPN_MSGID_CREATE_GROUP system message.

```
typedef struct _DPNMSG_CREATE_GROUP{
    DWORD  dwSize;
    DPNID  dpnidGroup;
    DPNID  dpnidOwner;
    PVOID  pvGroupContext;
} DPNMSG_CREATE_GROUP, *PDPNMSG_CREATE_GROUP;
```

dwSize
Size of this structure.

dpnidGroup
DPNID of the of the created group.

dpnidOwner
DPNID of the of the group's owner. This value is only set for groups that have the DPNGROUP_AUTODESTRUCT flag set in the **dwGroupFlags** member of the **DPN_GROUP_INFO** structure.

pvGroupContext

Group context value.

Remarks

The only method of setting the group context value is through this system message. Group context values once set cannot be changed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_CREATE_PLAYER

Microsoft® DirectPlay® generates the DPN_MSGID_CREATE_PLAYER message when a player is added to a peer-to-peer or client/server session.

DPNMSG_CREATE_PLAYER

The **DPNMSG_CREATE_PLAYER** structure contains information for the DPN_MSGID_CREATE_PLAYER system message.

```
typedef struct _DPNMSG_CREATE_PLAYER{
    DWORD  dwSize;
    DPNID  dpnidPlayer;
    PVOID  pvPlayerContext;
} DPNMSG_CREATE_PLAYER, *PDPNMSG_CREATE_PLAYER;
```

dwSize
Size of this structure.

dpnidPlayer
DPNID of the player that was added to the session.

pvPlayerContext
Player context value.

Remarks

The only method of setting the player context value is through this system message. You can either set the player context value directly, through this message, or indirectly through **DPN_MSGID_INDICATE_CONNECT**. Once a player context value has been set, it cannot be changed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_DESTROY_PLAYER

Microsoft® DirectPlay® generates the DPN_MSGID_DESTROY_PLAYER message when a player leaves a peer-to-peer or client/server session.

DPNMSG_DESTROY_PLAYER

The **DPNMSG_DESTROY_PLAYER** structure contains information for the DPN_MSGID_DESTROY_PLAYER system message.

```
typedef struct _DPNMSG_DESTROY_PLAYER{
    DWORD  dwSize;
    DPNID  dpnidPlayer;
    PVOID  pvPlayerContext;
    DWORD  dwReason;
} DPNMSG_DESTROY_PLAYER, *PDPNMSG_DESTROY_PLAYER;
```

dwSize
Size of this structure.

dpnidPlayer
DPNID of the player deleted from the session.

pvPlayerContext
Player context value.

dwReason
One of the following flags indicating why the player was destroyed.

DPNDESTROYPLAYERREASON_NORMAL
The player is being deleted for normal reasons.

DPNDESTROYPLAYERREASON_CONNECTIONLOST
The player is being deleted because the connection was lost.

DPNDESTROYPLAYERREASON_SESSIONTERMINATED
The player is being deleted because the session was terminated.

DPNDESTROYPLAYERREASON_HOSTDESTROYEDPLAYER
The player is being deleted because the host called **IDirectPlay8Peer::DestroyPeer**.

Remarks

In client/server mode, this message is received only by the server. In peer-to-peer mode, all players receive this message.

You may receive **DPN_MSGID_CREATE_PLAYER** and **DPN_MSGID_DESTROY_PLAYER** messages on different threads. However, you will not receive a **DPN_MSGID_DESTROY_PLAYER** message before your callback function has returned from receiving a **DPN_MSGID_CREATE_PLAYER** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_DESTROY_GROUP

Microsoft® DirectPlay® generates the DPN_MSGID_DESTROY_GROUP message when a group is destroyed from a peer-to-peer or client/server session.

DPNMSG_DESTROY_GROUP

The **DPNMSG_DESTROY_GROUP** structure contains information for the DPN_MSGID_DESTROY_GROUP system message.

```
typedef struct _DPNMSG_DESTROY_GROUP{
    DWORD  dwSize;
    DPNID  dpnidGroup;
    PVOID  pvGroupContext;
    DWORD  dwReason;
} DPNMSG_DESTROY_GROUP, *PDPNMSG_DESTROY_GROUP;
```

dwSize
Size of this structure.

dpnidGroup
DPNID of the group deleted from the session.

pvGroupContext
Group context value.

dwReason
One of the following flags indicating why the player was destroyed.

DPNDESTROYGROUPREASON_SESSIONTERMINATED
The group is being destroyed because the session was terminated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_ENUM_HOSTS_QUERY

Microsoft® DirectPlay® sends the DPN_MSGID_ENUM_HOSTS_QUERY message to the host's message handler when a peer or client is enumerating the available hosts.

DPNMSG_ENUM_HOSTS_QUERY

The **DPNMSG_ENUM_HOSTS_QUERY** structure contains information for the DPN_MSGID_ENUM_HOSTS_QUERY system message.

```
typedef struct _DPNMSG_ENUM_HOSTS_QUERY{
    DWORD               dwSize;
    IDirectPlay8Address* pAddressSender;
    IDirectPlay8Address* pAddressDevice;
    PVOID               pvReceivedData;
    DWORD               dwReceivedDataSize;
    DWORD               dwMaxResponseDataSize;
    PVOID               pvResponseData;
    DWORD               dwResponseDataSize;
    PVOID               pvResponseContext;
} DPNMSG_ENUM_HOSTS_QUERY, *PDPNMSG_ENUM_HOSTS_QUERY;
```

dwSize
Size of this structure.

pAddressSender
Pointer an **IDirectPlay8Address** interface specifying the address of the sender. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

pAddressDevice
Pointer an **IDirectPlay8Address** interface specifying the address of the device. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

pvReceivedData
Pointer to the data received from the enumeration.

dwReceivedDataSize
Size of the data pointed to in the **pvReceivedData** member.

dwMaxResponseDataSize
Maximum allowed size for the enumeration response.

pvResponseData

Pointer to the response data from the enumeration. This data must be valid beyond the scope of the callback message handler. It cannot be stack-based. You will receive a **DPN_MSGID_RETURN_BUFFER** message when Microsoft® DirectPlay® is finished with this buffer.

dwResponseDataSize
Size of the data pointed to in the **pvResponseData** member.

pvUserContext
Pointer to a response context value. This value will be passed to the host's message handler with the **DPN_MSGID_RETURN_BUFFER** message as the **pvUserContext** member of the associated structure.

Remarks

When you respond normally to this query, DirectPlay will send you a **DPN_MSGID_RETURN_BUFFER** message once the buffer is no longer needed. You can then safely free the buffer.

You can reject the query by returning a value that is not equal to S_OK. However, when you reject a query, DirectPlay does not send a reply, does not need a reply buffer, and does not generate a **DPN_MSGID_RETURN_BUFFER** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_ENUM_HOSTS_RESPONSE

Microsoft® DirectPlay® sends the DPN_MSGID_ENUM_HOSTS_RESPONSE message to a peer or client's message handler to convey the host's response to an enumeration request.

DPNMSG_ENUM_HOSTS_RESPONSE

The **DPNMSG_ENUM_HOSTS_RESPONSE** structure contains information for the DPN_MSGID_ENUM_HOSTS_RESPONSE system message.

```
typedef struct _DPNMSG_ENUM_HOSTS_RESPONSE{
    DWORD                   dwSize;
    IDirectPlay8Address*        pAddressSender;
    IDirectPlay8Address*        pAddressDevice;
    const DPN_APPLICATION_DESC*   pApplicationDescription;
    PVOID                   pvResponseData;
    DWORD                     dwResponseDataSize;
```

```
    PVOID                   pvUserContext;
    DWORD                   dwRoundTripLatencyMS;
} DPNMSG_ENUM_HOSTS_RESPONSE,
 *PDPNMSG_ENUM_HOSTS_RESPONSE;
```

dwSize
Size of this structure.

pAddressSender
Pointer to an **IDirectPlay8Address** interface specifying the address of the host responding to the enumeration. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

pAddressDevice
Pointer an **IDirectPlay8Address** interface specifying the address of the device. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

pApplicationDescription
Pointer to a **DPN_APPLICATION_DESC** structure containing the application description.

pvResponseData
Pointer to the response data from the enumeration.

dwResponseDataSize
Size of the data pointed to in the **pvResponseData** member.

pvUserContext
Pointer to the user context value. This value is the same as the user context value passed to **IDirectPlay8Peer::EnumHosts** or **IDirectPlay8Client::EnumHosts**.

dwRoundTripLatencyMS
Latency measured in milliseconds.

Remarks

Because there is no buffer to fill, this message does not generate a **DPN_MSGID_RETURN_BUFFER** message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_GROUP_INFO

Microsoft® DirectPlay® generates the DPN_MSGID_GROUP_INFO message when group data is modified during a peer-to-peer or client/server session.

DPNMSG_GROUP_INFO

The **DPNMSG_GROUP_INFO** structure contains information for the DPN_MSGID_GROUP_INFO system message.

```
typedef struct _DPNMSG_GROUP_INFO{
    DWORD  dwSize;
    DPNID  dpnidGroup;
    PVOID  pvGroupContext;
} DPNMSG_GROUP_INFO, *PDPNMSG_GROUP_INFO;
```

dwSize
Size of this structure.

dpnidGroup
DPNID of the group for group information.

pvGroupContext
Group context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_HOST_MIGRATE

Microsoft® DirectPlay® generates the DPN_MSGID_HOST_MIGRATE message if the DPNSESSION_MIGRATE_HOST flag is set in the **DPN_APPLICATION_DESC** structure and the host has migrated.

DPNMSG_HOST_MIGRATE

The **DPNMSG_HOST_MIGRATE** structure contains information for the DPN_MSGID_HOST_MIGRATE system message.

```
typedef struct _DPNMSG_HOST_MIGRATE{
    DWORD  dwSize;
    DPNID  dpnidNewHost;
    PVOID  pvPlayerContext;
} DPNMSG_HOST_MIGRATE, *PDPNMSG_HOST_MIGRATE;
```

dwSize
Size of this structure.

dpnidNewHost
DPNID of the player that is now hosting the session.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_INDICATE_CONNECT

Microsoft® DirectPlay® generates the DPN_MSGID_INDICATE_CONNECT message when a player attempts to connect to a peer-to-peer or client/server session.

DPNMSG_INDICATE_CONNECT

The **DPNMSG_INDICATE_CONNECT** structure contains information for the DPN_MSGID_INDICATE_CONNECT system message.

This structure gives the opportunity for the host application to allow or reject the connection based on user data and provide reply information to the connecting application.

```
typedef struct {
    DWORD     dwSize;
    PVOID     pvUserConnectData;
    DWORD     dwUserConnectDataSize;
    PVOID     pvReplyData;
    DWORD     dwReplyDataSize;
    PVOID     pvReplyContext;
    PVOID     pvPlayerContext;
    IDirectPlay8Address*  pAddressPlayer;
    IDirectPlay8Address*  pAddressDevice;
} DPNMSG_INDICATE_CONNECT, *PDPNMSG_INDICATE_CONNECT;
```

dwSize
Size of this structure.

pvUserConnectData
Data of the connecting player.

dwUserConnectDataSize
Size of the data, in bytes, contained in the **pvUserConnectData** member.

pvReplyData
Connection reply data. This data must be valid beyond the scope of the callback message handler. You will receive a **DPN_MSGID_RETURN_BUFFER** message when Microsoft® DirectPlay® is finished with this buffer.

dwReplyDataSize
Size of the data, in bytes, contained in the **pvReplyData** member.

pvReplyContext

Buffer context for **pvReplyData**. This value will be passed to the host's message handler with the **DPN_MSGID_RETURN_BUFFER** message as the **pvUserContext** member of the associated structure.

pvPlayerContext
Player context preset.

pAddressPlayer
Pointer to an **IDirectPlay8Address** interface for the connecting player. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

pAddressDevice
Pointer to an **IDirectPlay8Address** interface for the device receiving the connect attempt. You must call **IDirectPlay8Address::AddRef** to increment the interface's reference count. Call **IDirectPlay8Address::Release** when you no longer need the interface.

Remarks

Return S_OK to allow the player to join the session. Any other return value will reject the requested connection. The **hResultCode** member of the structure associated with the **DPN_MSGID_CONNECT_COMPLETE** message that is sent to the player requesting a connection will be set to S_OK if the connection was successful. If the connection is rejected, **hResultCode** will be set to DPNERR_HOSTREJECTEDCONNECTION, not the value you return from this message.

When an **DPN_MSGID_INDICATE_CONNECT** notification arrives on the host player's message handler, setting **pvPlayerContext** before returning the thread will preset the player context value on the respective **DPN_MSGID_CREATE_PLAYER** notification. This feature allows you to pass a player context value to **DPN_MSGID_CREATE_PLAYER**.

If you set a player context value, that value is not frozen until the subsequent **DPN_MSGID_CREATE_PLAYER**message has been processed. You thus have the option of modifying this player context value when you process **DPN_MSGID_CREATE_PLAYER**.

If a client drops the connection after the server has processed the **DPN_MSGID_INDICATE_CONNECT** message but before it has processed **DPN_MSGID_CREATE_PLAYER**, the server will receive a **DPN_MSGID_INDICATED_CONNECT_ABORTED** message. If you receive this message, free any memory that you allocated while processing **DPN_MSGID_INDICATE_CONNECT**. Once **DPN_MSGID_CREATE_PLAYER** has been processed, this memory should be freed when you process **DPN_MSGID_DESTROY_PLAYER**.

If you specify a value for **pvUserConnectData**, you will subsequently be sent a **DPN_MSGID_RETURN_BUFFER** message to notify you that you can safely free the buffer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_INDICATED_CONNECT_ABORTED

Microsoft® DirectPlay® generates the DPN_MSGID_INDICATED_CONNECT_ABORTED message if a player's connection drops after it was indicated on the host, but prior to being added to the session though **DPN_MSGID_CREATE_PLAYER**.

DPNMSG_INDICATED_CONNECT_ABORTED

The **DPNMSG_INDICATED_CONNECT_ABORTED** structure contains information for the DPN_MSGID_INDICATED_CONNECT_ABORTED system message.

```
typedef struct {
    DWORD  dwSize;
    PVOID  pvPlayerContext;
} DPNMSG_INDICATED_CONNECT_ABORTED,
*PDPNMSG_INDICATED_CONNECT_ABORTED;
```

dwSize
Size of this structure.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_PEER_INFO

Microsoft® DirectPlay® generates the DPN_MSGID_PEER_INFO message when peer data is modified during a peer-to-peer session.

DPNMSG_PEER_INFO

The **DPNMSG_PEER_INFO** structure contains information for the DPN_MSGID_PEER_INFO system message.

```
typedef struct {
    DWORD  dwSize;
    DPNID  dpnidPeer;
    PVOID  pvPlayerContext;
} DPNMSG_PEER_INFO, *PDPNMSG_PEER_INFO;
```

dwSize
Size of this structure.

dpnidPeer
DPNID of the peer for peer information.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_RECEIVE

Microsoft® DirectPlay® generates the DPN_MSGID_RECEIVE message when a message has been processed by the receiver.

DPNMSG_RECEIVE

The **DPNMSG_RECEIVE** structure contains information for the DPN_MSGID_RECEIVE system message.

```
typedef struct {
    DWORD      dwSize;
    DPNID      dpnidSender;
    PVOID      pvPlayerContext;
    PBYTE      pReceiveData;
    DWORD      dwReceiveDataSize;
    DPNHANDLE  hBufferHandle;
} DPNMSG_RECEIVE, *PDPNMSG_RECEIVE;
```

dwSize
Size of this structure.

dpnidSender

DPNID of the player that sent the message.

pvPlayerContext
Player context value of the player that sent the message.

pReceiveData
PBYTE pointer to the message data buffer. This buffer is normally only valid while the **DPN_MSGID_RECEIVE** message is being processed by the callback message handler. Because you should not spend large amounts of time processing messages, you should copy this data, and process the message. Alternatively, you can return DPNSUCCESS_PENDING from the callback message handler. Doing so transfers ownership of the buffer to the application. If you return DPNSUCCESS_PENDING, you must call **IDirectPlay8Peer::ReturnBuffer**, **IDirectPlay8Client::ReturnBuffer**, or **IDirectPlay8Server::ReturnBuffer** when you are finished with the buffer. Pass the method the value you receive in the **hBufferHandle** member to identify the buffer. If you fail to call **ReturnBuffer**, you will create a memory leak.

dwReceiveDataSize
Size of the data, in bytes, of the **pReceiveData** member.

hBufferHandle
Buffer handle for the **pReceiveData** member. If you have returned DPNSUCCESS_PENDING , pass this value to **ReturnBuffer** to notify Microsoft® DirectPlay® to free the buffer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_REMOVE_PLAYER_FROM_GROUP

Microsoft® DirectPlay® generates the DPN_MSGID_REMOVE_PLAYER_FROM_GROUP message when a player has been deleted from a group in a peer-to-peer or client/server session.

DPNMSG_REMOVE_PLAYER_FROM_GROUP

The **DPNMSG_REMOVE_PLAYER_FROM_GROUP** structure contains information for the DPN_MSGID_REMOVE_PLAYER_FROM_GROUP system message.

```
typedef struct _DPNMSG_REMOVE_PLAYER_FROM_GROUP{
  DWORD  dwSize;
```

```
    DPNID  dpnidGroup;
    PVOID  pvGroupContext;
    DPNID  dpnidPlayer;
    PVOID  pvPlayerContext;
  } DPNMSG_REMOVE_PLAYER_FROM_GROUP,
  *PDPNMSG_REMOVE_PLAYER_FROM_GROUP;
```

dwSize
Size of this structure.

dpnidGroup
DPNID of the group that the player was deleted from.

pvGroupContext
Group context value.

dpnidPlayer
DPNID of the player deleted from the group.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_RETURN_BUFFER

Microsoft® DirectPlay® generates the DPN_MSGID_RETURN_BUFFER message when DirectPlay is done with a user buffer.

DPNMSG_RETURN_BUFFER

The **DPNMSG_RETURN_BUFFER** structure contains information for the DPN_MSGID_RETURN_BUFFER system message.

```
typedef struct {
    DWORD      dwSize;
    HRESULT    hResultCode;
    PVOID      pvBuffer;
    PVOID      pUserContext;
} DPNMSG_RETURN_BUFFER, *PDPNMSG_RETURN_BUFFER;
```

dwSize
Size of this structure.

hResultCode
Return value of the operation. This will be set to DPNERR_ENUMRESPONSETOOLARGE if the response to a **DPN_MSGID_ENUM_HOSTS_QUERY** message is too large.

pvBuffer
Pointer to the buffer being returned.

pUserContext
Context value associated with the buffer.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_SEND_COMPLETE

Microsoft® DirectPlay® generates the DPN_MSGID_SEND_COMPLETE message when a send message request has completed.

DPNMSG_SEND_COMPLETE

The **DPNMSG_SEND_COMPLETE** structure contains information for the DPN_MSGID_SEND_COMPLETE system message.

```
typedef struct {
    DWORD      dwSize;
    DPNHANDLE  hAsyncOp;
    PVOID      pvUserContext;
    HRESULT    hResultCode;
    DWORD      dwSendTime;
} DPNMSG_SEND_COMPLETE, *PDPNMSG_SEND_COMPLETE;
```

dwSize
Size of this structure.

hAsyncOp
Asynchronous operation handle.

pvUserContext
User context supplied in the **IDirectPlay8Client::Send**, **IDirectPlay8Peer::SendTo** and **IDirectPlay8Server::SendTo** methods.

hResultCode
HRESULT indicating the result of the send message request.

dwSendTime
Total time, in milliseconds, between send call and completion.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_SERVER_INFO

Microsoft® DirectPlay® generates the DPN_MSGID_SERVER_INFO message when server data is modified during a client/server session.

DPNMSG_SERVER_INFO

The **DPNMSG_SERVER_INFO** structure contains information for the DPN_MSGID_SERVER_INFO system message.

```
typedef struct {
    DWORD  dwSize;
    DPNID  dpnidServer;
    PVOID  pvPlayerContext;
} DPNMSG_SERVER_INFO, *PDPNMSG_SERVER_INFO;
```

dwSize
Size of this structure.

dpnidServer
DPNID of the server for server information.

pvPlayerContext
Player context value.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_MSGID_TERMINATE_SESSION

Microsoft® DirectPlay® generates the DPN_MSGID_TERMINATE_SESSION message when a session is terminated by the host.

DPNMSG_TERMINATE_SESSION

The **DPNMSG_TERMINATE_SESSION** structure contains information for the DPN_MSGID_TERMINATE_SESSION system message.

```
typedef struct {
    DWORD   dwSize;
    HRESULT hResultCode;
    PVOID   pvTerminateData;
    DWORD   dwTerminateDataSize;
} DPNMSG_TERMINATE_SESSION, *PDPNMSG_TERMINATE_SESSION;
```

dwSize
Size of this structure.

hResultCode
Specifies how the session was terminated. This member is set to DPNERR_HOSTTERMINATEDSESSION if the session was peer-to-peer, and the host called **IDirectPlay8Peer::TerminateSession**. If the session was ended by the host calling **Close**, or if the host stops responding, **hResultCode** is set to DPNERR_CONNECTIONLOST.

pvTerminateData
Termination data. If **hResultCode** is set to DPNERR_HOSTTERMINATEDSESSION, **pvTerminateData** points to the data block that the host passed through the *pvTerminateData* parameter of **IDirectPlay8Peer::TerminateSession**.

dwTerminateDataSize
Size of the data block pointed to by **pvTerminateData**. This member will be zero if **pvTerminateData** is set to NULL.

Remarks

In a peer-peer game that permits host-migration, if the current host calls **Close** or stops responding, the session does not terminate. Instead, the host migrates and all nonhost players receive a **DPN_MSGID_DESTROY_PLAYER** message for the host's players, and a **DPN_MSGID_HOST_MIGRATE** message for the new host. To prevent host migration, the host must shut down the session by calling **IDirectPlay8Peer::TerminateSession**. When the host terminates a session this way, all players receive a **DPN_MSGID_TERMINATE_SESSION** message with **hResultCode** set to DPNERR_HOSTTERMINATEDSESSION. The session will terminate, generating **DPN_MSGID_DESTROY_PLAYER** messages for every player.

In a peer-peer game that does not permit host-migration, the session is terminated if the host calls **IDirectPlay8Peer::Close**, or stops responding. In that case, **DPN_MSGID_TERMINATE_SESSION** is sent to all players with **hResultCode** set to DPNERR_CONNECTIONLOST. The session will terminate, generating **DPN_MSGID_DESTROY_PLAYER** messages for every player.

In a client/server game, the session is also terminated if the host calls **IDirectPlay8Server::Close** or stops responding. In that case, **DPN_MSGID_TERMINATE_SESSION** is sent to all connected clients with **hResultCode** set to DPNERR_CONNECTIONLOST. The **DPN_MSGID_DESTROY_PLAYER** message not sent to clients. If the server disconnected by calling **IDirectPlay8Server::Close**, it will receive **DPN_MSGID_DESTROY_PLAYER** messages for all players, including its own. Otherwise, the server will only receive **DPN_MSGID_DESTROY_PLAYER** for the clients' players.

Note

The **DPN_MSGID_TERMINATE_SESSION** message typically arrives before any **DPN_MSGID_DESTROY_PLAYER** messages. However, the order of arrival is not guaranteed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DVMSGID_CONNECTRESULT

Microsoft® DirectPlay® Voice generates the DVMSGID_CONNECTRESULT message when the connect request generated through a call to the **IDirectPlayVoiceClient::Connect** method has completed. This message is sent only if the **Connect** method is called asynchronously.

DVMSG_CONNECTRESULT

The **DVMSG_CONNECTRESULT** structure contains information for the DVMSGID_CONNECTRESULT system message.

```
typedef struct {
    DWORD    dwSize;
    HRESULT  hrResult;
} DVMSG_CONNECTRESULT, *LPDVMSG_CONNECTRESULT,
*PDVMSG_CONNECTRESULT;
```

dwSize
Size of the **DVMSG_CONNECTRESULT** message structure.

hrResult
Result of the connection attempt.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_CREATEVOICEPLAYER

Microsoft® DirectPlay® Voice generates the DVMSGID_CREATEVOICEPLAYER message when a new player joins the voice session.

Upon connecting to a voice session, clients will receive one of these messages for each player in the voice session. These messages are sent only to clients in peer-to-peer voice sessions.

The host receives these messages when players join the voice session.

Players do not join the voice session until they have called **IDirectPlayVoiceClient::Connect**. Therefore, it is possible for a player to be in the transport session but not part of the voice session.

DVMSG_CREATEVOICEPLAYER

The **DVMSG_CREATEVOICEPLAYER** structure contains information for the DVMSGID_CREATEVOICEPLAYER system message.

```
typedef struct {
    DWORD  dwSize;
    DVID   dvidPlayer;
    DWORD  dwFlags;
    PVOID  pvPlayerContext;
} DVMSG_CREATEVOICEPLAYER, *LPDVMSG_CREATEVOICEPLAYER,
*PDVMSG_CREATEVOICEPLAYER;
```

dwSize
Size of the this message structure.

dvidPlayer
DVID of the player who connected.

dwFlags
Flag specifying information about the player:

DVPLAYERCAPS_HALFDUPLEX
The specified player is running in half duplex mode. The player will only be able to receive voice, not transmit it.

DVPLAYERCAPS_LOCAL
The player is the local player.

pvPlayerContext
Player context value for the player in the voice session. This value is set through this parameter when this message is received.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_DELETEVOICEPLAYER

For clients, Microsoft® DirectPlay® Voice generates the DVMSGID_DELETEVOICEPLAYER message when a player quits the voice session. This message is available only to clients in peer-to-peer voice sessions.

For the host, Microsoft® DirectPlay® Voice generates the DVMSGID_DELETEVOICEPLAYER message when a player quits the voice session.

Players do not leave the voice session until they have called **IDirectPlayVoiceClient::Disconnect** or they have disconnected from the transport session. Therefore, a client might be part of the transport session but not part of the voice session.

DVMSG_DELETEVOICEPLAYER

The **DVMSG_DELETEVOICEPLAYER** structure contains information for the DVMSGID_DELETEVOICEPLAYER system message.

```
typedef struct {
    DWORD  dwSize;
    DVID   dvidPlayer;
    PVOID  pvPlayerContext;
} DVMSG_DELETEVOICEPLAYER, *LPDVMSG_DELETEVOICEPLAYER,
*PDVMSG_DELETEVOICEPLAYER;
```

dwSize
Size of the **DVMSG_DELETEVOICEPLAYER** message structure.

dvidPlayer
DVID of player who disconnected.

pvPlayerContext
Pointer to the context value set for the player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_DISCONNECTRESULT

Microsoft® DirectPlay® Voice generates the DVMSGID_DISCONNECTRESULT message when the disconnect request generated through a call to the **IDirectPlayVoiceClient::Disconnect** method has completed. This message is sent only if the **Disconnect** method is called asynchronously.

DVMSG_DISCONNECTRESULT

The **DVMSG_DISCONNECTRESULT** structure contains information for the DVMSGID_DISCONNECTRESULT system message.

```
typedef struct {
    DWORD   dwSize;
    HRESULT hrResult;
```

```
} DVMSG_DISCONNECTRESULT, *LPDVMSG_DISCONNECTRESULT,
*PDVMSG_DISCONNECTRESULT;
```

dwSize
Size of the **DVMSG_DISCONNECTRESULT** message structure.

hrResult
Result of the disconnect request.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_GAINFOCUS

The DVMSGID_GAINFOCUS message is sent to notify you that you have begun capturing audio. It is sent when an application that has lost capture focus recovers it. There is no data associated with this message. Refer to the Microsoft® DirectSound® documentation for more information on capturing audio.

DVMSGID_HOSTMIGRATED

Microsoft® DirectPlay® Voice generates the DVMSGID_HOSTMIGRATED message when the voice host has changed.

DVMSG_HOSTMIGRATED

The **DVMSG_HOSTMIGRATED** structure contains information for the DVMSGID_HOSTMIGRATED system message.

```
typedef struct {
    DWORD               dwSize;
    DVID                dvidNewHostID;
    LPDIRECTPLAYVOICESERVER pdvServerInterface;
} DVMSG_HOSTMIGRATED, *LPDVMSG_HOSTMIGRATED,
*PDVMSG_HOSTMIGRATED;
```

dwSize
Size of the **DVMSG_HOSTMIGRATED** message structure.

dvidNewHostID
DVID of the new host.

pdvServerInterface
If the local client has become the new voice session host, this member will point to a newly created **IDirectPlayVoiceServer** object that can be used by the local client for providing host services. If the local client is not the new host, then this member will be NULL. If this parameter points to an **IDirectPlayVoiceServer**

interface, you must call **IDirectPlayVoiceServer::AddRef** to increment the interface's reference count. Call **IDirectPlayVoiceServer::Release** when you no longer need the interface.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_INPUTLEVEL

Microsoft® DirectPlay® Voice generates the DVMSGID_INPUTLEVEL message periodically to notify the user of the input level from the microphone. The period of notification is set by the **dwNotifyPeriod** member of the **DVCLIENTCONFIG** structure. If the notification period is set to 0, this message will not be sent. In addition, if the client is running in half duplex mode, this message is not available.

DVMSG_INPUTLEVEL

The **DVMSG_INPUTLEVEL** structure contains information for the DVMSGID_INPUTLEVEL system message.

```
typedef struct {
    DWORD dwSize;
    DWORD dwPeakLevel;
    LONG  lRecordVolume;
    PVOID pvLocalPlayerContext;
} DVMSG_INPUTLEVEL, *LPDVMSG_INPUTLEVEL,
*PDVMSG_INPUTLEVEL;
```

dwSize
Size of the **DVMSG_INPUTLEVEL** message structure.

dwPeakLevel
Integer representing peak level across the current frame, which corresponds to approximately 1/10 second of audio stream. The current frame typically lags 50-200 ms behind real-time. This value can range from 0 through 99, with 0 being completely silent and 99 being the highest possible input level.

lRecordVolume
Current recording volume for the client. The value can range from -10,000 to 0. This member is available even when automatic gain control is active.

pvLocalPlayerContext
Pointer to the context value set for the local player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_LOCALHOSTSETUP

The DVMSGID_LOCALHOSTSETUP message is sent when the local client is elected to become the new voice host during host migration. The message is sent before the DVMSGID_HOSTMIGRATED message and gives you the chance to set the callback function and context value that will be used when creating the new host object. If you do not set either of the values, then the new server interface will have no callback function. Once the application returns from handling this message it will receive the DVMSGID_HOSTMIGRATED message. The new message has the following associated structure, which is passed in the void * field of the message handler.

DVMSG_LOCALHOSTSETUP

The **DVMSG_LOCALHOSTSETUP** structure contains information for the DVMSGID_LOCALHOSTSETUP system message.

```
typedef struct {
    DWORD dwSize;
    PVOID pvContext;
    PDVMESSAGEHANDLER pMessageHandler;
} DVMSG_LOCALHOSTSETUP, *LPDVMSG_LOCALHOSTSETUP,
*PDVMSG_LOCALHOSTSETUP;
```

dwSize
Size of the **DVMSG_LOCALHOSTSETUP** message structure.

pvContext
Set to the context value you want to set for the new server.

pMessageHandler
Set to the callback function to be used for the new server.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_LOSTFOCUS

The DVMSGID_LOSTFOCUS message is sent to notify you that you have stopped capturing audio. It is sent when an application that has capture focus loses it to another application. There is no data associated with this message. Refer to the Microsoft® DirectSound® documentation for more information on capturing audio.

DVMSGID_OUTPUTLEVEL

Microsoft® DirectPlay® Voice generates the DVMSGID_OUTPUTLEVEL message periodically to notify the user of the output level of playback. The period of notification is set by the **dwNotifyPeriod** member of the **DVCLIENTCONFIG** structure. If the notification period is set to 0, this message will not be sent.

DVMSG_OUTPUTLEVEL

The **DVMSG_OUTPUTLEVEL** structure contains information for the DVMSGID_OUTPUTLEVEL system message.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwPeakLevel;
    LONG   lOutputVolume;
    PVOID  pvLocalPlayerContext;
} DVMSG_OUTPUTLEVEL, *LPDVMSG_OUTPUTLEVEL,
*PDVMSG_OUTPUTLEVEL;
```

dwSize
Size of the **DVMSG_OUTPUTLEVEL** message structure.

dwPeakLevel
Integer representing the current output level of playback. This value is in the range from 0 through 99, with 0 being completely silent and 99 being the highest possible output level.

lOutputVolume
Current playback volume for the client.

pvLocalPlayerContext
Pointer to the context value set for the local player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_PLAYEROUTPUTLEVEL

Microsoft® DirectPlay® Voice generates the DVMSGID_PLAYEROUTPUTLEVEL message periodically to notify the user of the output level of an individual player's voice stream. It is generated while voice is being played back for an individual player. If multiple player voices are being played, one message for each player speaking will be sent each notification period.

The period of notification is set by the **dwNotifyPeriod** member of the **DVCLIENTCONFIG** structure. If the notification period is set to 0, this message will not be sent.

DVMSG_PLAYEROUTPUTLEVEL

The **DVMSG_PLAYEROUTPUTLEVEL** structure contains information for the DVMSGID_PLAYEROUTPUTLEVEL system message.

```
typedef struct {
    DWORD  dwSize;
    DVID   dvidSourcePlayerID;
    DWORD  dwPeakLevel;
    PVOID  pvPlayerContext;
} DVMSG_PLAYEROUTPUTLEVEL, *LPDVMSG_PLAYEROUTPUTLEVEL,
*PDVMSG_PLAYEROUTPUTLEVEL;
```

dwSize
Size of the **DVMSG_PLAYEROUTPUTLEVEL** message structure.

dvidSourcePlayerID
DVID of the player whose voice is being played back.

dwPeakLevel
Integer representing the current output level of the player's voice stream. This value is in the range from 0 through 99, with 0 being completely silent and 99 being the highest possible output level.

pvPlayerContext
Pointer to the context value set for the player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_PLAYERVOICESTART

Microsoft® DirectPlay® Voice generates the DVMSGID_PLAYERVOICESTART message when an incoming audio stream begins playing back.

DVMSG_PLAYERVOICESTART

The **DVMSG_PLAYERVOICESTART** structure contains information for the DVMSGID_PLAYERVOICESTART system message.

```
typedef struct {
    DWORD  dwSize;
    DVID   dvidSourcePlayerID;
    PVOID  pvPlayerContext;
} DVMSG_PLAYERVOICESTART, *LPDVMSG_PLAYERVOICESTART,
*PDVMSG_PLAYERVOICESTART;
```

dwSize
Size of the **DVMSG_PLAYERVOICESTART** message structure.

dvidSourcePlayerID
DVID of the player where the voice transmission originated.

pvPlayerContext
Pointer to the context value set for the player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_PLAYERVOICESTOP

Microsoft® DirectPlay® Voice generates the DVMSGID_PLAYERVOICESTOP message when an incoming audio stream stops.

DVMSG_PLAYERVOICESTOP

The **DVMSG_PLAYERVOICESTOP** structure contains information for the DVMSGID_PLAYERVOICESTOP system message.

```
typedef struct {
    DWORD  dwSize;
    DVID   dvidSourcePlayerID;
    PVOID  pvPlayerContext;
```

```
} DVMSG_PLAYERVOICESTOP, *LPDVMSG_PLAYERVOICESTOP,
*PDVMSG_PLAYERVOICESTOP;
```

dwSize
Size of the **DVMSG_PLAYERVOICESTOP** message structure.

dvidSourcePlayerID
DVID of the player where the voice transmission originated.

pvPlayerContext
Pointer to the context value set for the player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_RECORDSTART

Microsoft® DirectPlay® Voice generates the DVMSGID_RECORDSTART message when audio input on the local client begins. This can be caused by the voice activation sensitivity level being exceeded or when a valid target is specified in push-to-talk mode.

DVMSG_RECORDSTART

The **DVMSG_RECORDSTART** structure contains information for the DVMSGID_RECORDSTART system message.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwPeakLevel;
    PVOID pvLocalPlayerContext;
} DVMSG_RECORDSTART, *LPDVMSG_RECORDSTART,
*PDVMSG_RECORDSTART;
```

dwSize
Size of the **DVMSG_RECORDSTART** message structure.

dwPeakLevel
Voice activation level that caused the transmission to begin. In push-to-talk mode, this value is 0.

pvLocalPlayerContext
Pointer to the context value set for the local player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_RECORDSTOP

Microsoft® DirectPlay® Voice generates the DVMSGID_RECORDSTOP message when audio input on the local client stops. This can be caused by the voice activation sensitivity level not being reached or when a target is deselected in push-to-talk mode.

DVMSG_RECORDSTOP

The **DVMSG_RECORDSTOP** structure contains information for the DVMSGID_RECORDSTOP system message.

```
typedef struct {
    DWORD dwSize;
    DWORD dwPeakLevel;
    PVOID pvLocalPlayerContext;
} DVMSG_RECORDSTOP, *LPDVMSG_RECORDSTOP,
*PDVMSG_RECORDSTOP;
```

dwSize
Size of the **DVMSG_RECORDSTOP** message structure.

dwPeakLevel
Voice activation level that caused the transmission to stop. In push-to-talk mode, this value is 0.

pvLocalPlayerContext
Pointer to the context value set for the local player. This value is set through the **pvPlayerContext** member of the **DVMSG_CREATEVOICEPLAYER** message structure.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_SESSIONLOST

Microsoft® DirectPlay® Voice generates the DVMSGID_SESSIONLOST message when the voice session terminates.

DVMSG_SESSIONLOST

The **DVMSG_SESSIONLOST** structure contains information for the DVMSGID_SESSIONLOST system message.

```
typedef struct {
    DWORD   dwSize;
    HRESULT hrResult;
} DVMSG_SESSIONLOST, *LPDVMSG_SESSIONLOST,
*PDVMSG_SESSIONLOST;
```

dwSize
Size of the **DVMSG_SESSIONLOST** message structure.

hrResult
HRESULT indicating why the session was terminated.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

DVMSGID_SETTARGETS

Microsoft® DirectPlay® Voice generates the DVMSGID_SETTARGETS message when the **IDirectPlayVoiceClient::SetTransmitTargets** or **IDirectPlayVoiceServer::SetTransmitTargets** methods are called.

DVMSG_SETTARGETS

The **DVMSG_SETTARGETS** structure contains information for the DVMSGID_SETTARGETS system message.

```
typedef struct {
    DWORD dwSize;
    DWORD dwNumTargets;
    PDVID pdvidTargets;
} DVMSG_SETTARGETS, *LPDVMSG_SETTARGETS,
*PDVMSG_SETTARGETS;
```

dwSize
Size of the **DVMSG_SETTARGETS** message structure.

dwNumTargets
Number of DVIDs contained in the **pdvidTargets** member.

pdvidTargets
Array of DVIDs specifying the set targets. This can also be set to NULL if there are no targets.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice.h.

Structures

The Microsoft® DirectPlay® structures are:

- **DPL_APPLICATION_INFO**
- **DPL_CONNECT_INFO**
- **DPL_CONNECTION_SETTINGS**
- **DPL_PROGRAM_DESC**
- **DPN_APPLICATION_DESC**
- **DPN_BUFFER_DESC**
- **DPN_CAPS**
- **DPN_CONNECTION_INFO**
- **DPN_GROUP_INFO**
- **DPN_PLAYER_INFO**
- **DPN_SECURITY_CREDENTIALS**
- **DPN_SECURITY_DESC**
- **DPN_SERVICE_PROVIDER_INFO**
- **DPN_SP_CAPS**
- **DVCAPS**
- **DVCLIENTCONFIG**
- **DVCOMPRESSIONINFO**
- **DVSESSIONDESC**
- **DVSOUNDDEVICECONFIG**

DPL_APPLICATION_INFO

Used in the *pEnumData* parameter of the **IDirectPlay8LobbyClient::EnumLocalPrograms** method to describe the lobbied application.

```
typedef struct _DPL_APPLICATION_INFO {
    GUID  guidApplication;
    PWSTR pwszApplicationName;
```

```
    DWORD dwNumRunning;
    DWORD dwNumWaiting;
    DWORD dwFlags;
} DPL_APPLICATION_INFO,  *PDPL_APPLICATION_INFO;
```

Members

guidApplication
Variable of type GUID specifying the lobbied application.

pwszApplicationName
Pointer to a variable of type **WSTR** containing the name of the lobbied application.

dwNumRunning
Number of instances of the application.

dwNumWaiting
Number of clients waiting to connect to the lobbied application.

dwFlags
Reserved. Must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_CONNECT_INFO

Used to specify connection information for a lobby client when connecting to the lobby application in the **IDirectPlay8LobbyClient::ConnectApplication** method.

```
typedef struct _DPL_CONNECT_INFO {
    DWORD dwSize;
    DWORD dwFlags;
    GUID  guidApplication;
    PDPL_CONNECTION_SETTINGS pdplConnectionSettings;
    PVOID pvLobbyConnectData;
    DWORD dwLobbyConnectDataSize;
} DPL_CONNECT_INFO,  *PDPL_CONNECT_INFO;
```

Members

dwSize
Size of the **DPL_CONNECT_INFO** structure. The application must set this member before it uses the structure.

dwFlags
One of the following flags, which determine connection behavior.

DPLCONNECT_LAUNCHNEW

Launches a new instance of the application.

DPLCONNECT_LAUNCHNOTFOUND
Launches a new instance of the application only if there is currently no application running that can supply launch settings.

guidApplication
Variable of type GUID specifying the application.

pdplConnectionSettings
Contains the connection settings you want to associate with the connection when it is established.

pvLobbyConnectData
Pointer to connection data passed to the lobbied application.

dwLobbyConnectDataSize
Variable of type **DWORD** specifying the size of the data buffer in the **pvLobbyConnectData** member.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_CONNECTION_SETTINGS

Used to specify the settings you want to associate with a connection. These settings contain all the information required to create, initialize and connect/host a Microsoft® DirectPlay® object.

```
typedef struct _DPL_CONNECTION_SETTINGS {
    DWORD                dwSize;
    DWORD                dwFlags;
    DPN_APPLICATION_DESC    dpnAppDesc;
    IDirectPlay8Address*    pdp8HostAddress;
    IDirectPlay8Address**   ppdp8DeviceAddresses;
    DWORD                cNumDeviceAddresses;
    PWSTR                pwszPlayerName
} DPL_CONNECTION_SETTINGS, *PDPL_CONNECTION_SETTINGS;
```

Members

dwSize
Size of the **DPL_CONNECTION_SETTINGS** structure. The application must set this to sizeof(DPL_CONNECTION_SETTINGS) before using this structure.

dwFlags
Combination of the following flags.

DPLCONNECTSETTINGS_CLIENTSERVER

The application should be launched with a client/server session.

DPLCONNECTSETTINGS_HOST
The application should host the session.

dpnAppDesc
Pointer to the application description that should be passed to the **Connect** or **Host** call when DirectPlay initialized.

pdp8HostAddress
If DPLCONNECTSETTINGS_HOST is not specified, this is the address of the session the client should connect to. If DPLCONNECTSETTINGS_HOST is specified, this member must be NULL.

ppdp8DeviceAddresses
This structure contains an array of pointers to device addresses. If DPLCONNECTSETTINGS_HOST is specified, this member will contain the addresses the host should listen on. If DPLCONNECTSETTINGS_HOST is not specified, this member will contain the address of the devices the client should use when connecting.

cNumDeviceAddresses
Number of addresses specified in the **ppdp8DeviceAddresses** member.

pwszPlayerName
Can be used to pass the player name you want the DirectPlay object to use when launching. This member can be NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPL_PROGRAM_DESC

Describes a Microsoft® DirectPlay® lobby-aware application.

```
typedef struct _DPL_PROGRAM_DESC {
    DWORD dwSize;
    DWORD dwFlags;
    GUID  guidApplication;
    PWSTR pwszApplicationName;
    PWSTR pwszCommandLine;
    PWSTR pwszCurrentDirectory;
    PWSTR pwszDescription;
    PWSTR pwszExecutableFilename;
    PWSTR pwszExecutablePath;
    PWSTR pwszLauncherFilename;
    PWSTR pwszLauncherPath;
} DPL_PROGRAM_DESC, *PDPL_PROGRAM_DESC;
```

Members

dwSize
Size of the **DPL_PROGRAM_DESC** structure. The application must set this member before it uses the structure.

dwFlags
Reserved. Must be 0.

guidApplication
Variable of type GUID specifying the application.

pwszApplicationName
Pointer to the application name.

pwszCommandLine
Pointer to the command-line arguments.

pwszCurrentDirectory
Pointer to the directory that should be set as the application's working directory..

pwszDescription
Pointer to the application description.

pwszExecutableFilename
Pointer to the file name of the application executable.

pwszExecutablePath
Pointer to the path of the application executable.

pwszLauncherFilename
Pointer to the file name of the launcher executable.

pwszLauncherPath
Pointer to the path of the launcher executable.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplobby8.h.

DPN_APPLICATION_DESC

Describes the settings for a Microsoft® DirectPlay® application.

```
typedef struct _DPN_APPLICATION_DESC{
    DWORD  dwSize;
    DWORD  dwFlags;
    GUID   guidInstance;
    GUID   guidApplication;
    DWORD  dwMaxPlayers;
    DWORD  dwCurrentPlayers;
    WCHAR* pwszSessionName;
```

```
    WCHAR* pwszPassword;
    PVOID  pvReservedData;
    DWORD  dwReservedDataSize;
    PVOID  pvApplicationReservedData;
    DWORD  dwApplicationReservedDataSize;
} DPN_APPLICATION_DESC, *PDPN_APPLICATION_DESC;
```

Members

dwSize
Size of the **DPN_APPLICATION_DESC** structure. The application must set this member before it uses the structure.

dwFlags
One of the following flags describing application behavior.

DPNSESSION_CLIENT_SERVER
This type of session is client/server. This flag cannot be combined with DPNSESSION_MIGRATE_HOST.

DPNSESSION_MIGRATE_HOST
Used in peer-to-peer sessions, enables host migration. This flag cannot be combined with DPNSESSION_CLIENT_SERVER.

DPNSESSION_NODPNSVR
You do not want enumerations forwarded to your host from DPNSVR. See Using the DirectPlay DPNSVR Application.

DPNSESSION_REQUIREPASSWORD
The session is password protected. If this flag is set, *pwszPassword* must be a valid string.

guidInstance
Globally unique identifier (GUID) that is generated by DirectPlay at startup representing the instance of this application. This member is an [out] parameter when calling the **GetApplicationDesc** method exposed by the **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server** interfaces. It is an optional [in] parameter when calling the **Connect** method exposed by the **IDirectPlay8Peer** and **IDirectPlay8Client** interfaces. It must be set to NULL when you call the **SetApplicationDesc** method exposed by the **IDirectPlay8Server** and **IDirectPlay8Peer** interfaces. You can not obtain this GUID by calling the **IDirectPlay8Server::Host** or **IDirectPlay8Peer::Host** methods. You must obtain the GUID by calling a **GetApplicationDesc** method.

guidApplication
Application GUID.

dwMaxPlayers
Variable of type **DWORD** specifying the maximum number of players allowed in the session. Set this member to 0 to specify an unlimited number of players.

dwCurrentPlayers
Variable of type **DWORD** specifying the number of players currently connected to the session. This member is an [out] parameter that is set only by the

GetApplicationDescription method exposed by **IDirectPlay8Peer**, **IDirectPlay8Client**, and **IDirectPlay8Server**.

pwszSessionName
Pointer to a variable of type **WCHAR** specifying the Unicode™ name of the session. This member is set by the host or server only for informational purposes. A client cannot use this name to connect to a host or server.

pwszPassword
Pointer to a variable of type **WCHAR** specifying the Unicode password that is required to connect to the session. This must be NULL if the DPNSESSION_REQUIREPASSWORD is not set in the **dwFlags** member.

pvReservedData
Pointer to DirectPlay reserved data. An application should never modify this value.

dwReservedDataSize
Variable of type **DWORD** specifying the size of data contained in the **pvReservedData** member. An application should never modify this value.

pvApplicationReservedData
Pointer to application-specific reserved data. This value is optional and may be set to NULL.

dwApplicationReservedDataSize
Variable of type **DWORD** specifying the size of the data in the **pvApplicationReservedData** member. This value is optional and may be set to 0.

Remarks

The **dwMaxPlayers**, **pvApplicationReservedData**, **dwApplicationReservedDataSize**, **pwszPassword**, and **pwszSessionName** members can be set when calling the **Host** or **SetApplicationDesc** methods exposed by the **IDirectPlay8Server** and **IDirectPlay8Peer** interfaces.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_BUFFER_DESC

Used by Microsoft® DirectPlay® for generic buffer information.

```
typedef struct _BUFFERDESC{
    DWORD  dwBufferSize;
    BYTE*  pBufferData;
} BUFFERDESC, DPN_BUFFER_DESC;
```

Members

dwBufferSize
Variable of type **DWORD** that specifies the size of the data buffer in the **pBufferData** member.

pBufferData
Pointer to a variable of type **BYTE** that contains the buffer data.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_CAPS

Used to set and retrieve general parameters for DirectPlay.

```
typedef struct _DPN_CAPS{
    DWORD dwSize;
    DWORD dwFlags;
    DWORD dwConnectTimeout;
    DWORD dwConnectRetries;
    DWORD dwTimeoutUntilKeepAlive;
} DPN_CAPS, *PDPN_CAPS;
```

Members

dwSize
This value must be set to the size of the structure.

dwFlags
Reserved, this must be 0.

dwConnectTimeout
Number of milliseconds DirectPlay should wait before it retries a connection request.

dwConnectRetries
Number of connection retries DirectPlay should make during the connection process.

dwTimeoutUntilKeepAlive
Number of milliseconds DirectPlay should wait since the last time it received a packet from an endpoint, before it sends a keep-alive message.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_CONNECTION_INFO

Used to retrieve statistics for the connection between you and a remote computer that you are connected to.

```
typedef struct _DPN_CONNECTION_INFO{
    DWORD   dwSize;
    DWORD   dwRoundTripLatencyMS;
    DWORD   dwThroughputBPS;
    DWORD   dwPeakThroughputBPS;
    DWORD   dwBytesSentGuaranteed;
    DWORD   dwPacketsSentGuaranteed;
    DWORD   dwBytesSentNonGuaranteed;
    DWORD   dwPacketsSentNonGuaranteed;
    DWORD   dwBytesRetried;
    DWORD   dwPacketsRetried;
    DWORD   dwBytesDropped;
    DWORD   dwPacketsDropped;
    DWORD   dwMessagesTransmittedHighPriority;
    DWORD   dwMessagesTimedOutHighPriority;
    DWORD   dwMessagesTransmittedNormalPriority;
    DWORD   dwMessagesTimedOutNormalPriority;
    DWORD   dwMessagesTransmittedLowPriority;
    DWORD   dwMessagesTimedOutLowPriority;
    DWORD   dwBytesReceivedGuaranteed;
    DWORD   dwPacketsReceivedGuaranteed;
    DWORD   dwBytesReceivedNonGuaranteed;
    DWORD   dwPacketsReceivedNonGuaranteed;
    DWORD   dwMessagesReceived;
} DPN_CONNECTION_INFO, *PDPN_CONNECTION_INFO;
```

Members

dwSize
Size of the structure.

dwRoundTripLatencyMS
Approximate time, in milliseconds (ms), it takes a packet to reach the remote computer and be returned to the local computer. This number will change throughout the session as link conditions change.

dwThroughputBPS

Approximate throughput, in bytes per second (Bps), for the link. This number will change throughout the session as link conditions change. This value is approximate, and you may want to calculate your own value for greater accuracy.

dwPeakThroughputBPS
Peak throughput, in bytes per second (Bps) for the link. This number will change throughout the session as link conditions change. This value is approximate, and you may want to calculate your own value for greater accuracy.

dwBytesSentGuaranteed
Amount, in bytes, of guaranteed messages that have been sent.

dwPacketsSentGuaranteed
Number of packets of guaranteed messages that have been sent.

dwBytesSentNonGuaranteed
Amount, in bytes, of nonguaranteed messages that have been sent.

dwPacketsSentNonGuaranteed
Number of packets of nonguaranteed messages that have been sent.

dwBytesRetried
Amount, in bytes, of messages that have been retried.

dwPacketsRetried
Amount of packets that have been retried.

dwBytesDropped
Amount, in bytes, of messages that have been dropped.

dwPacketsDropped
Number of packets that have been dropped.

dwMessagesTransmittedHighPriority
Number of high-priority messages that have been transmitted.

dwMessagesTimedOutHighPriority
Number of high-priority messages that have timed out.

dwMessagesTransmittedNormalPriority
Number of normal-priority messages that have been transmitted.

dwMessagesTimedOutNormalPriority
Number of normal-priority messages that have timed out.

dwMessagesTransmittedLowPriority
Number of low-priority messages that have been transmitted.

dwMessagesTimedOutLowPriority
Number of low priority messages that have timed out.

dwBytesReceivedGuaranteed
Amount, in bytes, of guaranteed messages that have been received.

dwPacketsReceivedGuaranteed
Number of packets of guaranteed messages that have been received.

dwBytesReceivedNonGuaranteed
Amount, in bytes, of nonguaranteed messages that have been received.

dwPacketsReceivedNonGuaranteed
Number of packets of nonguaranteed messages that have been received.

dwMessagesReceived
Number of messages that have been received.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_GROUP_INFO

Describes static group information.

```
typedef struct _DPN_GROUP_INFO{
    DWORD  dwSize;
    DWORD  dwInfoFlags;
    PWSTR  pwszName;
    PVOID  pvData;
    DWORD  dwDataSize;
    DWORD  dwGroupFlags;
} DPN_GROUP_INFO, *PDPN_GROUP_INFO;
```

Members

dwSize
Variable of type **DWORD** describing the size of this structure.

dwInfoFlags
Variable of type **DWORD** containing flags that specify the type of information contained in this structure. When a **GetGroupInfo** method returns, the **dwInfoFlags** member of the **DPN_GROUP_INFO** will always have both flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Peer::SetGroupInfo,** to notify Microsoft® DirectPlay® of which values have changed.

DPNINFO_NAME
The **pwszName** member contains valid data.

DPNINFO_DATA
The **pvData** member contains valid data.

pwszName
Pointer to a variable of type **PWSTR** specifying the Unicode name of the group.

pvData
Pointer to the data describing the group.

dwDataSize
Variable of type **DWORD** that specifies the size of the data contained in the **pvData** member.

dwGroupFlags

Variable of type **DWORD** that can be set to the following description flag.

DPNGROUP_AUTODESTRUCT
Causes the group to be automatically destroyed when the group creator leaves the group.

Remarks

When using this structure in the **IDirectPlay8Peer::GetGroupInfo** and **IDirectPlay8Server::GetGroupInfo** methods, **dwInfoFlags** must be set to 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_PLAYER_INFO

Describes static player information.

```
typedef struct _DPN_PLAYER_INFO{
    DWORD  dwSize;
    DWORD  dwInfoFlags;
    PWSTR  pwszName;
    PVOID  pvData;
    DWORD  dwDataSize;
    DWORD  dwPlayerFlags;
} DPN_PLAYER_INFO, *PDPN_PLAYER_INFO;
```

Members

dwSize
Variable of type **DWORD** describing the size of this structure.

dwInfoFlags
Variable of type **DWORD** containing flags that specify the type of information contained in this structure. When a **GetPlayerInfo** method returns, the **dwInfoFlags** member of the **DPN_PLAYER_INFO** will always have both flags set, even if the corresponding pointers are set to NULL. These flags are used when calling **IDirectPlay8Peer::SetPeerInfo,** to notify Microsoft® DirectPlay® which values have changed.

DPNINFO_NAME
The **pwszName** member contains valid data.

DPNINFO_DATA
The **pvData** member contains valid data.

pwszName

Pointer to a variable of type **PWSTR** specifying the Unicode name of the player.

pvData
Pointer to the data describing the player.

dwDataSize
Variable of type **DWORD** that specifies the size of the data contained in the **pvData** member.

dwPlayerFlags
Variable of type **DWORD** that may contain one of the following flags.

DPNPLAYER_LOCAL
This information is for the local player.

DPNPLAYER_HOST
This player is the host for the application.

Remarks

When using this structure in the **IDirectPlay8Peer::GetPeerInfo** and **IDirectPlay8Server::GetClientInfo** methods, **dwInfoFlags** must be set to 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_SECURITY_CREDENTIALS

Not implemented.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_SECURITY_DESC

Not implemented.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_SERVICE_PROVIDER_INFO

Used when enumerating information for a specific service provider.

```
typedef struct _DPN_SERVICE_PROVIDER_INFO{
    DWORD  dwFlags;
    GUID  Guid;
    WCHAR* pwszName;
    PVOID  pvReserved;
    DWORD  dwReserved;
} DPN_SERVICE_PROVIDER_INFO, *PDPN_SERVICE_PROVIDER_INFO;
```

Members

dwFlags
Reserved. Must be 0.

pGuid
GUID for the service provider.

pwszName
Name of the service provider.

pvReserved
Reserved. Must be 0.

dwReserved
Reserved. Must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DPN_SP_CAPS

Used to set and retrieve parameters for service providers.

```
typedef struct _DPN_SP_CAPS{
    DWORD  dwSize;
    DWORD  dwFlags;
    DWORD  dwNumThreads;
    DWORD  dwDefaultEnumCount;
    DWORD  dwDefaultEnumRetryInterval;
    DWORD  dwDefaultEnumTimeout;
    DWORD  dwMaxEnumPayloadSize;
    DWORD  dwBuffersPerThread;
    DWORD  dwSystemBufferSize;
```

} DPN_SP_CAPS, *PDPN_SP_CAPS;

Members

dwSize
This value must be set to the size of the structure.

dwFlags
This can be a combination of the following flags.

DPNSPCAPS_SUPPORTSDPNSRV
DPNSVR.EXE will provide port sharing for the given SP. Currently this flag is available on IP and IPX only. See Using the DirectPlay DPNSVR Application for a further discussion of DPNSVR.

DPNSPCAPS_SUPPORTSBROADCAST
On IP and IPX applications, the service provider has the ability to broadcast to find games if not enough addressing information is passed.

DPNSPCAPS_SUPPORTSALLADAPTERS
The service provider will use all devices on the system. There is no need to specify a device element.

dwNumThreads
Number of threads the service provider will use for servicing network requests. The default value for this is based on an algorithm that takes into account the number of processors on the system. Most applications will not need to modify this value.

After a service provider is active in your process, you may only increase this value. Decreasing the value will have no effect. The setting is process wide, which means it will affect your current DirectPlay object and any other DirectPlay objects in your process.

You may specify a lower value than the default if you call the **SetSPCaps** method before you call an **EnumHosts, Connect**, or **Host** method.

dwDefaultEnumCount
Default enumeration count.

dwDefaultEnumRetryInterval
Default retry interval, in milliseconds.

dwDefaultEnumTimeout
Default enumeration timeout value, in milliseconds.

dwMaxEnumPayloadSize
Maximum size of the payload information that can be sent in the **pvResponseData** member of the structures that accompany the **DPN_MSGID_ENUM_HOST_QUERY** and **DPN_MSGID_ENUM_HOST_RESPONSE** messages..

dwBuffersPerThread
The number of outstanding receive buffers allocated for each DirectPlay thread. If you increase the number of receive buffers, DirectPlay can pull more data out of the operating system buffers. However, you may also increase latency if data is arriving faster than your application can process it.

dwSystemBufferSize
The size of the operating system buffer. This buffer holds data from the communications device when your application cannot process data as fast as it arrives. The purpose of this buffer is to prevent data loss if you receive a sudden burst of data, or if the receive threads are momentarily stalled. Increasing **dwSystemBufferSize** may increase latency if your application cannot process the received data fast enough. You can eliminate the operating system buffer by setting **dwSystemBufferSize** to 0. However, if you do so, you run the risk of losing data if you cannot process the received data as fast as it arrives.

Remarks

For DirectX 8.0, the **dwBuffersPerThread** and **dwSystemBufferSize** members are used only by IP and IPX service providers. The default values for these members are set by the service provider. To determine the default value, call the appropriate **GetSPCaps** method. Most applications should use the default values for these two members. They are intended primarily for use by developers writing server applications for massively-multiplayer games.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dplay8.h.

DVCAPS

Describes the capabilities of the Microsoft® DirectPlay® VoiceClient object.

```
typedef struct{
    DWORD dwSize;
    DWORD dwFlags;
} DVCAPS, *LPDVCAPS, *PDVCAPS;
```

Members

dwSize
Must be set the to size of this structure, in bytes, before using this structure.

dwFlags
Reserved. Must be 0.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice8.h.

DVCLIENTCONFIG

Controls the run-time parameters for the client. The structure is first used in the call to **IDirectPlayVoiceClient::Connect**, where it sets the initial state of these parameters. The structure can be retrieved after a connection has been made by calling **IDirectPlayVoiceClient::GetClientConfig**, and set using **IDirectPlayVoiceClient::SetClientConfig**.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwFlags;
    LONG   lRecordVolume;
    LONG   lPlaybackVolume;
    DWORD  dwThreshold;
    DWORD  dwBufferQuality;
    DWORD  dwBufferAggressiveness;
    DWORD  dwNotifyPeriod;
} DVCLIENTCONFIG, *LPDVCLIENTCONFIG, *PDVCLIENTCONFIG;
```

Members

dwSize
Must be set the to size of this structure, in bytes, before using this structure.

dwFlags
Combination of the following flags.

DVCLIENTCONFIG_AUTORECORDVOLUME
Activates automatic gain control. With automatic gain control, Microsoft® DirectPlay® Voice adjusts the hardware input volume on your sound card automatically to get the best input level possible. You can determine the current input volume by looking at the **lRecordVolume** member after a call to **IDirectPlayVoiceClient::GetClientConfig**, or by looking at the **lRecordVolume** member of **DVMSG_INPUTLEVEL** messages.

DVCLIENTCONFIG_ECHOSUPPRESSION
Activates the echo suppression mode. This mode reduces echo introduced by configurations with external speakers and extremely sensitive microphones. While remote players' voices are being played back on the local speaker, the microphone is automatically muted. If the local player is transmitting, the playback of remote player voices is buffered until local input stops. After local input stops, playback resumes.

DVCLIENTCONFIG_MUTEGLOBAL
Mutes playback of the main sound buffer. Only sound buffers created through calls to **IDirectPlayVoiceClient::Create3DSoundBuffer** will be heard.

DVCLIENTCONFIG_PLAYBACKMUTE
Mutes playback of all DirectPlay Voice output and stops playback. This also stops decompression of incoming packets so CPU usage is reduced. Packets are effectively discarded while this flag is specified.

DVCLIENTCONFIG_RECORDMUTE
Mutes input from the microphone and stops recording. This also stops compression so CPU usage is reduced.

In addition to the preceding flags, the method of transmission is controlled by setting only one of the following flags or by not specifying either flag.

DVCLIENTCONFIG_AUTOVOICEACTIVATED
Places the transmission control system into automatic voice activation mode. In this mode, the sensitivity of voice activation is determined automatically by the system. The input level is adaptive, adjusting itself automatically to the input signal. For most applications this should be the setting used. This flag is mutually exclusive with the DVCLIENTCONFIG_MANUALVOICEACTIVATED flag.

DVCLIENTCONFIG_MANUALVOICEACTIVATED
Places the transmission control system into manual voice activation mode. In this mode, transmission of voice begins when the input level passes the level specified by the **dwThreshold** member. When input levels drop below the specified level, transmission stops. This flag is mutually exclusive with the DVCLIENTCONFIG_AUTOVOICEACTIVATED flag.

If you do not specify either DVCLIENTCONFIG_MANUALVOICEACTIVATED or DVCLIENTCONFIG_AUTOVOICEACTIVATED, the system will operate in push-to-talk mode. In push-to-talk mode, as long as there is a valid target specified the input from the microphone will be transmitted. Voice transmission stops when a NULL target is set or the current target leaves the session or is destroyed.

lRecordVolume
Value indicating what the volume of the recording should be set to. See the **IDirectSoundBuffer8::SetVolume** method for valid values.

If automatic gain control is enabled, this value can be set to DVRECORDVOLUME_LAST, which tells the system to use the current volume as determined by the automatic gain control algorithm. If a value other than DVRECORDVOLUME_LAST is specified in combination with automatic gain control, this value will be used to restart the algorithm at the specified value.

On return from a call to **IDirectPlayVoiceClient::GetClientConfig**, this value will contain the current recording volume. When adjusting the recording volume, DirectPlay Voice will adjust the volume for the microphone (if a microphone volume is present for the card) and the master recording volume (if one is present on the card). If neither a microphone volume nor a master record volume is present, DirectPlay Voice will be unable to adjust the recording volume.

lPlaybackVolume
Value indicating what the volume of the playback should be set to. Adjusting this volume adjusts both the main buffer and all 3-D sound buffers. See the **IDirectSoundBuffer8::SetVolume** method for valid values. You can specify DVPLAYBACKVOLUME_DEFAULT to use a default value that is appropriate for most situations (full volume).

dwThreshold

Input level used to trigger voice transmission if the DVCLIENTCONFIG_MANUALVOICEACTIVATED flag is specified in the **dwFlags** member. When the flag is specified, this value can be set to anywhere in the range of DVTHRESHOLD_MIN to DVTHRESHOLD_MAX. Additionally, DVTHRESHOLD_DEFAULT can be set to use a default value.

If DVCLIENTCONFIG_MANUALVOICEACTIVATED or DVCLIENTCONFIG_AUTOVOICEACTIVATED is not specified in the **dwFlags** member of this structure (indicating push-to-talk mode) this value must be set to DVTHRESHOLD_UNUSED.

dwBufferQuality

Buffer quality setting for the adaptive buffering algorithm. For most applications, this should be set to DVBUFFERQUALITY_DEFAULT. It can be set to anything in the range of DVBUFFERQUALITY_MIN to DVBUFFERQUALITY_MAX. In general, the higher the value, the higher the quality of the voice but the higher the latency. The lower the value, the lower the latency but the lower the quality.

dwBufferAggressiveness

Buffer aggressiveness setting for the adaptive buffer algorithm. For most applications, this can be set to DVBUFFERAGGRESSIVENESS_DEFAULT. It can also be set to anything in the range of DVBUFFERAGGRESSIVENESS_MIN and DVBUFFERAGGRESSIVENESS_MAX. In general, the higher the value, the quicker the adaptive buffering adjusts to changing conditions. The lower the value, the slower the adaptive buffering adjusts to changing conditions.

dwNotifyPeriod

Value indicating how often you want to receive DVMSGID_OUTPUTLEVEL and DVMSGID_INPUTLEVEL (if session is full duplex) messages. If this value is set to 0, these messages are disabled. The value specifies the number of milliseconds between these messages. DVNOTIFYPERIOD_MINPERIOD specifies the minimum period between messages that is allowed.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice8.h.

DVCOMPRESSIONINFO

Describes the attributes of a specific Microsoft® DirectPlay® Voice compression type.

```
typedef struct{
    DWORD       dwSize;
    GUID        guidType;
    LPWSTR      lpszName;
```

```
    LPWSTR      lpszDescription;
    DWORD       dwFlags;
    DWORD       dwMaxBitsPerSecond;
} DVCOMPRESSIONINFO, *LPDVCOMPRESSIONINFO,
*PDVCOMPRESSIONINFO;
```

Members

dwSize
Must be set the to size of this structure, in bytes, before using this structure.

guidType
GUID used to identify this compression type by DirectPlay Voice.

lpszName
Pointer to a name describing the codec.

lpszDescription
Pointer to a longer name of the codec.

dwFlags
Reserved; must be 0.

dwMaxBitsPerSecond
Maximum number of bits per second claimed by the codec.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice8.h.

DVSESSIONDESC

Describes the desired or current session settings for the Microsoft® DirectPlay® Voice server. This structure is used by the voice session host to configure the session, and by the session host and clients to retrieve information about the current session. The **dwFlags**, **dwSessionType**, and **guidCT** members can only be set when the host starts the voice session. The host can change the buffer settings at any time.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwFlags;
    DWORD  dwSessionType;
    GUID   guidCT;
    DWORD  dwBufferQuality;
    DWORD  dwBufferAggresiveness;
} DVSESSIONDESC, *LPDVSESSIONDESC, *PDVSESSIONDESC;
```

Members

dwSize

Must be set the to size of this structure, in bytes, before using this structure.

dwFlags

Combination of the following flags.

DVSESSION_NOHOSTMIGRATION

The voice host will not migrate regardless of the transport settings. If this flag is not specified, the voice host will migrate if the transport supports it.

DVSESSION_SERVERCONTROLTARGET

The clients are unable to control the target of their speech. Only the server player can control the target of their speech. If the server does not specify this flag, only the clients can control the target of their speech. This flag can be specified only in multicast and mixing sessions.

dwSessionType

The type of DirectPlay Voice session to run. The DVSESSIONTYPE_PEER flag is not available in client/server sessions; all other flags are valid for all session types. This member can be one of the following values.

DVSESSIONTYPE_PEER

Voice messages will be sent directly between players.

DVSESSIONTYPE_MIXING

Voice session will use a mixing server. In this mode of operation, all voice messages are sent to the server, which mixes them and then forwards a single, premixed stream to each client. This reduces the bandwidth and CPU usage on clients significantly at the cost of increased bandwidth and CPU usage on the server.

DVSESSIONTYPE_FOWARDING

Voice messages will be routed through the session host. This will save bandwidth on the clients at the expense of bandwidth usage on the server. This option is only useful if the session host has a high-speed connection.

guidCT

GUID specifying the compression type of the session.

dwBufferQuality

The buffer quality setting. This member is unused for all session types except mixing sessions. For all sessions except mixing sessions, set this member to DVBUFFERQUALITY_DEFAULT.

Allowable values are between DVBUFFERQUALITY_MIN and DVBUFFERQUALITY_MAX. Additionally, this member can be set to the following value.

DVBUFFERQUALITY_DEFAULT

Specifying this value tells DirectPlay Voice to use the system default for this value, which is adjustable through a registry entry that can also be set through Sounds and Multimedia in Control Panel.

dwBufferAggresiveness

Buffer aggressiveness setting. This member is unused for all session types except mixing sessions. For all sessions except mixing sessions, set this member to DVBUFFERAGGRESIVENESS_DEFAULT.

Allowable values are between DVBUFFERAGGRESIVENESS _MIN and DVBUFFERAGGRESIVENESS _MAX. Additionally, this member can be set to the following value.

DVBUFFERAGGRESIVENESS_DEFAULT
Specifying this value tells DirectPlay Voice to use the system default for this value, which is adjustable through a registry entry that can also be set through Control Panel.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice8.h.

DVSOUNDDEVICECONFIG

Used to set and retrieve information about the sound device configuration and cannot be changed once a connection has been made. After a connection is made, you can retrieve the current sound device configuration by calling **IDirectPlayVoiceClient::GetSoundDeviceConfig**.

```
typedef struct {
    DWORD               dwSize;
    DWORD               dwFlags;
    GUID                guidPlaybackDevice;
    LPDIRECTSOUND       lpdsPlaybackDevice;
    GUID                guidCaptureDevice;
    LPDIRECTSOUNDCAPTURE lpdsCaptureDevice;
    HWND                hwndAppWindow;
    LPDIRECTSOUNDBUFFER  lpdsMainBuffer;
    DWORD               dwMainBufferFlags;
    DWORD               dwMainBufferPriority;
} DVSOUNDDEVICECONFIG, *LPDVSOUNDDEVICECONFIG,
*PDVSOUNDDEVICECONFIG;
```

Members

dwSize
Must be set the to size of this structure, in bytes, before using this structure.

dwFlags
A combination of the following flags.

DVSOUNDCONFIG_AUTOSELECT

Tells Microsoft® DirectPlay® Voice to attempt to automatically select (or unmute) the microphone line in the mixer for the specified recording device.

DVSOUNDCONFIG_HALFDUPLEX
Tells DirectPlay Voice to initialize itself in half-duplex mode. In half-duplex mode no recording takes place. If the initialization of the sound system fails in full-duplex mode, this flag will be set by the system.

DVSOUNDCONFIG_NORMALMODE
Tells DirectPlay Voice to use Microsoft® DirectSound® Normal Mode when initializing the DirectSound object. If this flag is not specified, the DirectSound object is initialized with DirectSound Priority Mode. See documentation for **IDirectSound8::SetCooperativeLevel** for more information. If a valid DirectSound object is specified in the **lpdsPlaybackDevice** member, this flag is ignored.

DVSOUNDCONFIG_SETCONVERSIONQUALITY
Enables better quality audio at the expense of higher CPU usage.

DVSOUNDCONFIG_NORECVOLAVAILABLE
Set by DirectPlay Voice if there are no volume controls available on the recording device you specified. You cannot set this flag.

DVSOUNDCONFIG_NOFOCUS
The voice application will never go out of focus. In other words, the application will never release the sound capture device. Use of this flag is not recommended.

DVSOUNDCONFIG_STRICTFOCUS
The voice application will lose focus whenever its window is not the foreground window.

Note

Applications should set the DVSOUNDCONFIG_NOFOCUS or DVSOUNDCONFIG_STRICTFOCUS flags only when strictly necessary. Instead, you should normally use the default behavior that results when neither flag is set.

guidPlaybackDevice

When this structure is used in the **IDirectPlayVoiceClient::Connect** method, this member specifies the GUID of the device used for playback. This must be specified even if the **lpdsPlaybackDevice** member is used. You can also specify the following default GUIDs provided by DirectSound.

DSDEVID_DefaultPlayback
The system default playback device.

DSDEVID_DefaultVoicePlayback
The default voice playback device.

When this structure is used in the **IDirectPlayVoiceClient::GetSoundDeviceConfig** method, this member contains the actual device GUID used for playback.

lpdsPlaybackDevice

When this structure is used in the **IDirectPlayVoiceClient::Connect** method, this member specifies the DirectSound object you want DirectPlay Voice to use for playback. The GUID specified in **guidPlaybackDevice** must match the one used to create the device specified by this parameter. If you used NULL when specifying the device when you created your DirectSound object, pass DSDEVID_DefaultPlayback for this member.

When this structure is used in the **IDirectPlayVoiceClient::GetSoundDeviceConfig** method, this member contains a pointer to the DirectSound object being used by DirectPlay Voice. This will either be a pointer to the object specified when **Connect** was called or a pointer to a newly created and initialized DirectSound object. If you want to use this DirectSound object, you must store the pointer and increment the reference count by calling **AddRef** on the DirectSound interface.

guidCaptureDevice

When this structure is used in **IDirectPlayVoiceClient::Connect** method, this member specifies the GUID of the device used for capture. This must be specified even if the **lpdsCaptureDevice** member is used. If you used NULL when specifying the device when you created your DirectSoundCapture object, pass DSDEVID_DefaultCapture for this member.

When this structure is used in the **IDirectPlayVoiceClient::GetSoundDeviceConfig** method, this member will contain the actual device GUID used for capture.

lpdsCaptureDevice

When this structure is used in the **IDirectPlayVoiceClient::Connect** method, this member specifies the DirectSound object you want DirectPlay Voice to use for capture. The GUID specified in **guidCaptureDevice** must match the one used to create the device specified by this parameter. If you want to have DirectPlay Voice create the DirectSoundCapture object for you, specify NULL for this member.

When this structure is used in the **IDirectPlayVoiceClient::GetSoundDeviceConfig** method, this member contains a pointer to the DirectSoundCapture object being used by DirectPlay Voice. This will either be a pointer to the object specified when **Connect** was called or a pointer to a newly created and initialized DirectSoundCapture object. If you want to use this DirectSoundCapture object, you must store the pointer and increment the reference count by calling **AddRef** on the **IDirectSoundCapture8** interface. If the DirectPlay Voice object is operating in half duplex mode, this member will be NULL.

hwndAppWindow

Must be set to the handle of the window that will be used to determine focus for sound playback. See **IDirectSound8::SetCooperativeLevel** for information on DirectSound focus. If you do not have a window to use for focus, use **GetDesktopWindow** to use the desktop window.

lpdsMainBuffer

Pointer to an **IDirectSoundBuffer8** interface, which is used to create the DirectPlay Voice main buffer. This can be either NULL or a user-created

DirectSound buffer. If this member is set to NULL, DirectPlay Voice will create a buffer for the main voice buffer. If users specify a buffer here, DirectPlay Voice will use their buffer for the main voice buffer. User-created buffers have the following restrictions.

- The buffer must be 22 kilohertz, 16-bit, Mono format.
- The buffer must be at least 1 second in length.
- The buffer must have been created with the DSBCAPS_GETCURRENTPOSITION2 and DSBCAPS_CTRL3D flags.
- The buffer must not be a primary buffer.
- The buffer must not be playing when it is passed to the DirectPlay Voice software.
- The buffer must not be locked when it is passed to the DirectPlay Voice software.

dwMainBufferFlags
Passed directly to the *dwFlags* parameter of the **IDirectSoundBuffer8::Play** method when **Play** is called for the main buffer. The DSBPLAY_LOOPING flag is automatically added to this field. See the documentation on **IDirectSoundBuffer8::Play** for details. This parameter must be 0 if the **lpdsMainBufferDesc** member of this structure is NULL.

dwMainBufferPriority
Passed directly to the *dwPriority* parameter of the **IDirectSoundBuffer8::Play** method when **Play** is called on the main buffer. See documentation for **IDirectSoundBuffer8::Play** for more information. This member must be set to 0 if **lpdsMainBufferDesc** is NULL.

Requirements

Windows NT/2000: Available as a redistributable for Windows 2000 and later.
Windows 95/98: Available as a redistributable for Windows 95 and later.
Header: Declared in Dvoice8.h.

Return Values

Errors are represented by negative values and cannot be combined.

The following table lists the interfaces to which the error values listed below apply.

Interface	Interface
IDirectPlay8Address	**IDirectPlay8Peer**

IDirectPlay8AddressIP
IDirectPlay8Client
IDirectPlay8LobbiedApplication
IDirectPlay8LobbyClient
IDirectPlay8Server
IDirectPlayVoiceClient
IDirectPlayVoiceServer
IDirectPlayVoiceTest

For a list of the error values each method can return, see the individual method descriptions.

Many of the Microsoft® DirectPlay® samples include a **GetDirectPlayErrStr** function that converts HRESULT values to string names for the DirectPlay errors. You can copy this code into your own applications for diagnostic traces or error reports.

Success Codes

DPNSUCCESS_PENDING
An asynchronous operation has reached the point where it is successfully queued.

S_OK
The operation completed successfully.

Error Codes

DPNERR_ABORTED
The operation was canceled before it could be completed.

DPNERR_ADDRESSING
The address specified is invalid.

DPNERR_ALREADYCONNECTED
The object is already connected to the session.

DPNERR_ALREADYCLOSING
An attempt to call the **Close** method on a session has been made more than once.

DPNERR_ALREADYDISCONNECTING
The client is already disconnecting from the session.

DPNERR_ALREADYINITIALIZED
The object has already been initialized.

DPNERR_BUFFERTOOSMALL
The supplied buffer is not large enough to contain the requested data.

DPNERR_CANNOTCANCEL
The operation could not be canceled.

DPNERR_CANTCREATEGROUP
A new group cannot be created.

DPNERR_CANTCREATEPLAYER
A new player cannot be created.

DPNERR_CANTLAUNCHAPPLICATION
The lobby cannot launch the specified application.

DPNERR_CONNECTING
The method is in the process of connecting to the network.

DPNERR_CONNECTIONLOST
The service provider connection was reset while data was being sent.

DPNERR_DATATOOLARGE
The application data is too large for the service provider's Maximum Transmission Unit.

DPNERR_DOESNOTEXIST
Requested element is not part of the address.

DPNERR_ENUMQUERYTOOLARGE
The query data specified is too large.

DPNERR_ENUMRESPONSETOOLARGE
The response to an enumeration query is too large.

DPNERR_EXCEPTION
An exception occurred when processing the request.

DPNERR_GENERIC
An undefined error condition occurred.

DPNERR_GROUPNOTEMPTY
The specified group is not empty.

DPNERR_HOSTREJECTEDCONNECTION
The **DPN_MSGID_INDICATE_CONNECT** system message returned something other than S_OK in response to a connect request.

DPNERR_HOSTTERMINATEDSESSION
The host in a peer session (with host migration enabled) terminated the session.

DPNERR_INCOMPLETEADDRESS
The address specified is not complete.

DPNERR_INVALIDADDRESSFORMAT
Address format is invalid.

DPNERR_INVALIDAPPLICATION
The GUID supplied for the application is invalid.

DPNERR_INVALIDCOMMAND
The command specified is invalid.

DPNERR_INVALIDDEVICEADDRESS
The address for the local computer or adapter is invalid.

DPNERR_INVALIDFLAGS
The flags passed to this method are invalid.

DPNERR_INVALIDGROUP
The group ID is not recognized as a valid group ID for this game session.

DPNERR_INVALIDHANDLE
The handle specified is invalid.

DPNERR_INVALIDHOSTADDRESS
The specified remote address is invalid.

DPNERR_INVALIDINSTANCE
The GUID for the application instance is invalid.

DPNERR_INVALIDINTERFACE
The interface parameter is invalid. This value will be returned in a connect request if the connecting player was not a client in a client/server game or a peer in a peer-to-peer game.

DPNERR_INVALIDOBJECT
The DirectPlay object pointer is invalid.

DPNERR_INVALIDPARAM
One or more of the parameters passed to the method are invalid.

DPNERR_INVALIDPASSWORD
An invalid password was supplied when attempting to join a session that requires a password.

DPNERR_INVALIDPLAYER
The player ID is not recognized as a valid player ID for this game session.

DPNERR_INVALIDPOINTER
Pointer specified as a parameter is invalid.

DPNERR_INVALIDPRIORITY
The specified priority is not within the range of allowed priorities, which is inclusively from 0 through 65535.

DPNERR_INVALIDSTRING
String specified as a parameter is invalid.

DPNERR_INVALIDURL
Specified string is not a valid DirectPlay URL.

DPNERR_INVALIDVERSION
There was an attempt to connect to an invalid version of DirectPlay.

DPNERR_NOCAPS
The communication link that DirectPlay is attempting to use is not capable of this function.

DPNERR_NOCONNECTION
No communication link was established.

DPNERR_NOHOSTPLAYER
There is currently no player acting as the host of the session.

DPNERR_NOINTERFACE
The interface is not supported.

DPNERR_NORESPONSE
There was no response from the specified target.

DPNERR_NOTALLOWED
Object is read-only; this function is not allowed on this object.

DPNERR_NOTHOST
An attempt by the client to connect to a nonhost computer. Additionally, this error value may be returned by a nonhost that tries to set the application description.

DPNERR_OUTOFMEMORY
There is insufficient memory to perform the requested operation.

DPNERR_PENDING
Not an error, this return indicates that an asynchronous operation has reached the point where it is successfully queued. **SUCCEEDED**(DPNERR_PENDING) will return TRUE. This error value has been superseded by DPNERR_SUCCESS, which should be used by all new applications. DPNERR_PENDING is only included for backward compatibility.

DPNERR_PLAYERLOST
A player has lost the connection to the session.

DPNERR_PLAYERNOTREACHABLE
A player has tried to join a peer-peer session where at least one other existing player in the session cannot connect to the joining player.

DPNERR_SESSIONFULL
The request to connect to the host or server failed because the maximum number of players allotted for the session has been reached.

DPNERR_TIMEDOUT
The operation could not complete because it has timed out.

DPNERR_UNINITIALIZED
The requested object has not been initialized.

DPNERR_UNSUPPORTED
The function or feature is not available in this implementation or on this service provider.

DPNERR_USERCANCEL
The user canceled the operation.

DV_OK
The request completed successfully.

DV_FULLDUPLEX
The sound card is capable of full-duplex operation.

DV_HALFDUPLEX
The sound card can only be run in half-duplex mode.

DVERR_BUFFERTOOSMALL
The supplied buffer is not large enough to contain the requested data.

DVERR_EXCEPTION
An exception occurred when processing the request.

DVERR_GENERIC
An undefined error condition occurred.

DVERR_INVALIDFLAGS
The flags passed to this method are invalid.

DVERR_INVALIDOBJECT
The DirectPlay object pointer is invalid.

DVERR_INVALIDPARAM

One or more of the parameters passed to the method are invalid.

DVERR_INVALIDPLAYER
The player ID is not recognized as a valid player ID for this game session.

DVERR_INVALIDGROUP
The group ID is not recognized as a valid group ID for this game session.

DVERR_INVALIDHANDLE
The handle specified is invalid.

DVERR_OUTOFMEMORY
There is insufficient memory to perform the requested operation.

DVERR_PENDING
Not an error, this return indicates that an asynchronous operation has reached the point where it is successfully queued.

DVERR_NOTSUPPORTED
The operation is not supported.

DVERR_NOINTERFACE
The specified interface is not supported. Could indicate using the wrong version of DirectPlay.

DVERR_SESSIONLOST
The transport has lost the connection to the session.

DVERR_NOVOICESESSION
The session specified is not a voice session.

DVERR_CONNECTIONLOST
The connection to the voice session has been lost.

DVERR_NOTINITIALIZED
The **IDirectPlayVoiceClient::Initialize** or **IDirectPlayVoiceServer::Initialize** method must be called before calling this method.

DVERR_CONNECTED
The DirectPlayVoice object is connected.

DVERR_NOTCONNECTED
The DirectPlayVoice object is not connected.

DVERR_CONNECTABORTING
The connection is being disconnected.

DVERR_NOTALLOWED
The object does not have the permission to perform this operation.

DVERR_INVALIDTARGET
The specified target is not a valid player ID or group ID for this voice session.

DVERR_TRANSPORTNOTHOST
The object is not the host of the voice session.

DVERR_COMPRESSIONNOTSUPPORTED
The specified compression type is not supported on the local computer.

DVERR_ALREADYPENDING
An asynchronous call of this type is already pending.

DVERR_ALREADYINITIALIZED
The object has already been initialized.

DVERR_SOUNDINITFAILURE
A failure was encountered initializing the sound card.

DVERR_TIMEOUT
The operation could not be performed in the specified time.

DVERR_CONNECTABORTED
The connect operation was canceled before it could be completed.

DVERR_NO3DSOUND
The local computer does not support 3-D sound.

DVERR_ALREADYBUFFERED
There is already a user buffer for the specified ID.

DVERR_NOTBUFFERED
There is no user buffer for the specified ID.

DVERR_HOSTING
The object is the host of the session.

DVERR_NOTHOSTING
The object is not the host of the session.

DVERR_INVALIDDEVICE
The specified device is invalid.

DVERR_RECORDSYSTEMERROR
An error in the recording system occurred.

DVERR_PLAYBACKSYSTEMERROR
An error in the playback system occurred.

DVERR_SENDERROR
An error occurred while sending data.

DVERR_USERCANCEL
The user canceled the operation.

DVERR_UNKNOWN
An unknown error occurred.

DVERR_RUNSETUP
The specified audio configuration has not been tested. Call the **IDirectPlayVoiceTest::CheckAudioSetup** method.

DVERR_INCOMPATIBLEVERSION
The client connected to a voice session that is incompatible with the host.

DVERR_INITIALIZED
The **Initialize** method failed because the object has already been initialized.

DVERR_INVALIDPOINTER
The pointer specified is invalid.

DVERR_NOTRANSPORT
The specified object is not a valid transport.

DVERR_NOCALLBACK

This operation cannot be performed because no callback function was specified.

DVERR_TRANSPORTNOTINIT
Specified transport is not yet initialized.

DVERR_TRANSPORTNOSESSION
Specified transport is valid but is not connected/hosting.

DVERR_TRANSPORTNOPLAYER
Specified transport is connected/hosting but no local player exists.

DirectPlay Visual Basic Reference

This section contains reference information for the API elements of Microsoft® DirectPlay® for Microsoft® Visual Basic®. Reference material is divided into the following categories.

- Classes
- Functions
- Types
- Enumerations
- Error Codes

Classes

This section contains references for methods of the following Microsoft® DirectPlay® classes.

- **DirectPlay8Address**
- **DirectPlay8Client**
- **DirectPlay8Event**
- **DirectPlay8LobbiedApplication**
- **DirectPlay8LobbyClient**
- **DirectPlay8LobbyEvent**
- **DirectPlay8Peer**
- **DirectPlay8Server**
- **DirectPlayVoiceClient8**
- **DirectPlayVoiceEvent8**

- **DirectPlayVoiceServer8**
- **DirectPlayVoiceTest8**

DirectPlay8Address

#The **DirectPlay8Address** class provides methods for creating and managing **DirectPlay** addresses.

The methods of the **DirectPlay8Address** class are:

DirectPlay8Address Methods	**AddComponentLong**
	AddComponentString
	BuildFromURL
	Clear
	Duplicate
	GetComponentLong
	GetComponentString
	GetDevice
	GetNumComponents
	GetSP
	GetURL
	GetUserData
	SetDevice
	SetEqual
	SetSP
	SetUserData

DirectPlay8Address.AddComponentLong

#Adds a component of type **Long** to the address. If the component is part of the address, then it is replaced by the new value in this call.

AddComponentLong(*sComponent* **As String,** *lValue* **As Long)**

Parts

sComponent
String that contains the key for the component.

lValue
Long value to be added to the component.

IDH_DirectPlay8Address_dplay_vb
IDH_DirectPlay8Address.AddComponentLong_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDPARAM
DPNERR_NOTALLOWED

DirectPlay8Address.AddComponentStrin g

#Adds a component of type **String** to the address. If the component is part of the address, then it is replaced by the new value in this call.

AddComponentString(*sComponent* **As String,** *sValue* **As String)**

Parts

sComponent
String that contains the key for the component.

sValue
String value to be added to the component.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDPARAM
DPNERR_NOTALLOWED

DirectPlay8Address.BuildFromURL

#Sets the object equal to the address of a DirectPlay URL. It erases the contents of the object.

BuildFromURL(*SourceURL* **As String)**

Parts

SourceURL
String that contains a properly formatted DirectPlay address.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

IDH_DirectPlay8Address.AddComponentString_dplay_vb
IDH_DirectPlay8Address.BuildFromURL_dplay_vb

DPNERR_INVALIDURL
DPNERR_NOTALLOWED

DirectPlay8Address.Clear

#Resets the address object to an empty address.

Clear()

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_NOTALLOWED

DirectPlay8Address.Duplicate

#Creates a **DirectPlay8Address** object that duplicates the address in this object.

Duplicate() As DirectPlay8Address

Return Values

Returns a **DirectPlay8Address** object.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_GENERIC

DirectPlay8Address.GetComponentLong

#Retrieves one of the integer components in the address URL string.

GetComponentLong(*sComponent* **As String) As Long**

Parts

sComponent
Name of the component to retrieve.

Return Values

Returns the value of the requested component.

IDH_DirectPlay8Address.Clear_dplay_vb
IDH_DirectPlay8Address.Duplicate_dplay_vb
IDH_DirectPlay8Address.GetComponentLong_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_DOESNOTEXIST
DPNERR_INVALIDPARAM
DPNERR_BUFFERTOOSMALL

Remarks

For a discussion of the URL string, see DirectPlay Addressing.

DirectPlay8Address.GetComponentStrin g

#Retrieves one of the integer components in the address URL string.

GetComponentString(*sComponent* **As String) As String**

Parts

sComponent
Name of the component to retrieve.

Return Values

Returns the requested string.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_DOESNOTEXIST
DPNERR_INVALIDPARAM
DPNERR_BUFFERTOOSMALL

Remarks

For a discussion of the URL string, see DirectPlay Addressing.

DirectPlay8Address.GetDevice

#Retrieves the local device GUID in the address object. If no device is specified, this method raises DPNERR_DOESNOTEXIST.

IDH_DirectPlay8Address.GetComponentString_dplay_vb
IDH_DirectPlay8Address.GetDevice_dplay_vb

GetDevice() As String

Return Values

Returns a **String** value representing the device in the address object.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_DOESNOTEXIST

DirectPlay8Address.GetNumComponents

#Retrieves the number of components in the address.

GetNumComponents() As Long

Return Values

Returns the number of components in this address object.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Address.GetSP

#Retrieves the service provider GUID in the address object. If no service provider is specified, this method raises DPNERR_DOESNOTEXIST.

GetSP() As String

Return Values

Returns the service provider in the address object.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_DOESNOTEXIST

IDH_DirectPlay8Address.GetNumComponents_dplay_vb
IDH_DirectPlay8Address.GetSP_dplay_vb

DirectPlay8Address.GetURL

#Retrieves the DirectPlay address URL represented by this object.

GetURL() As String

Return Values

Returns the URL represented by this object.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_OUTOFMEMORY
DPNERR_INVALIDURL
DPNERR_GENERIC

DirectPlay8Address.GetUserData

#Retrieves the user data in the address object. If no user data exists in this address object, DPNERR_DOESNOTEXIST is raised.

GetUserData(*UserData* **As Long)**

Parts

UserData
User data from this address.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_DOESNOTEXIST

DirectPlay8Address.SetDevice

#Sets the local device GUID in the address object. If a local device is specified for this address, it is overwritten by this call.

SetDevice(*guidDevice* **As String)**

Parts

guidDevice

IDH_DirectPlay8Address.GetURL_dplay_vb
IDH_DirectPlay8Address.GetUserData_dplay_vb
IDH_DirectPlay8Address.SetDevice_dplay_vb

GUID of the local device.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_NOTALLOWED

DirectPlay8Address.SetEqual

#Sets this **DirectPlay8Address** object to the address specified.

SetEqual(*Address* **As DirectPlay8Address)**

Parts

Address
DirectPlay8Address object that this object will be set to.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_NOTALLOWED

DirectPlay8Address.SetSP

#Sets the service provider GUID in the address object. If a service provider is specified for this address, it is overwritten by this call.

SetSP(*guidSP* **As String)**

Parts

guidSP
Service provider GUID to set.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_NOTALLOWED

IDH_DirectPlay8Address.SetEqual_dplay_vb
IDH_DirectPlay8Address.SetSP_dplay_vb

DirectPlay8Address.SetUserData

#Sets the user data in the address object. If there is user data in this address, it is overwritten by this call.

SetUserData(*UserData* **As Long,** *lDataSize* **As Long)**

Parts

UserData
Data to place in the user data section of the address. Set to 0 to clear the user data.

lDataSize
Size, in bytes, of the data in *UserData*. If *UserData* is 0, this must be 0.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_NOTALLOWED

DirectPlay8Client

#Applications use the methods of the **DirectPlay8Client** class to create and manage client applications for client/server sessions.

The methods of the **DirectPlay8Client** class can be organized into the following groups.

Session Management	**Close**
	Connect
	EnumHosts
	GetApplicationDesc
	GetCaps
	GetCountServiceProviders
	GetServiceProvider
	GetSPCaps
	SetCaps
	SetSPCaps
Message Management	**GetSendQueueInfo**
	RegisterMessageHandler

IDH_DirectPlay8Address.SetUserData_dplay_vb
IDH_DirectPlay8Client_dplay_vb

	UnRegisterMessageHandler
	Send
Client Information	**SetClientInfo**
Server Information	**GetServerAddress**
	GetServerInfo
Miscellaneous	**CancelAsyncOperation**
	GetConnectionInfo
	RegisterLobby

DirectPlay8Client.CancelAsyncOperation

[#]Cancels asynchronous requests. Many methods of the **DirectPlay8Client** class run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this class that can be run asynchronously return an *lAsyncHandle* parameter.

Specific requests are canceled by passing the *lAsyncHandle* of the request in this method's *lAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, passing 0 in the *lAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *lFlags* parameter. If a specific handle is provided to this method, you must pass 0 in the *lFlags* parameter.

CancelAsyncOperation(*lAsyncHandle* **As Long, _**
[*lFlags* **As CONST_DPNCANCELFLAGS])**

Parts

lAsyncHandle
: Handle of the asynchronous operation to stop. This value can be 0 to stop all requests or a particular type of asynchronous request. If a specific handle for the request to cancel is specified, the *lFlags* parameter must be 0.

lFlags
: Flag that specifies which asynchronous request to cancel. You can set this parameter to one of the flags of the **CONST_DPNCANCELFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_PENDING

DPNERR_INVALIDFLAGS

DPNERR_CANNOTCANCEL

IDH_DirectPlay8Client.CancelAsyncOperation_dplay_vb

DPNERR_INVALIDHANDLE

Remarks

You can cancel a send by providing the handle returned from **DirectPlay8Client.Send** method. The **DirectPlay8Event.SendComplete** method will still be called unless the message was sent with the DPNSEND_NOCOMPLETE flag set. If you cancel a send operation by calling **DirectPlay8Peer.CancelAsyncOperation** the **hResultCode** member of the **DPNMSG_SEND_COMPLETE** type that is passed to the **DirectPlay8Event.SendComplete** method will be set to DPNERR_CANCELLED.

DirectPlay8Client.Close

#Closes the open connection with a session.

Close([*lFlags* **As Long]);**

Parts

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_UNINITIALIZED

DirectPlay8Client.Connect

#Establishes the connection to the server. Once a connection is established, the communication channel on the interface is open and active and the application should expect messages to arrive immediately. No messages can be sent by means of the **DirectPlay8Client.Send** method until the connection has completed.

Before this method is called, you can obtain an application description by calling **DirectPlay8Client.EnumHosts**. The **EnumHosts** method returns a **DPN_APPLICATION_DESC** type for each hosted application. The type describes the application, including the GUID of the application.

When the connection to the host is requested, the **DirectPlay8Event.IndicateConnect** method is called in the host's message handler. The host may either accept or reject the connection. In either case, once the host has acted, the client message handler's **DirectPlay8Event.ConnectComplete** method will be called to convey the response.

IDH_DirectPlay8Client.Close_dplay_vb
IDH_DirectPlay8Client.Connect_dplay_vb

Connect(*AppDesc* **As DPN_APPLICATION_DESC, _**
Address **As DirectPlay8Address, _**
DeviceInfo **As DirectPlay8Address, _**
lFlags **As CONST_DPNOPERATIONS, _**
UserData **As Any, _**
UserDataSize **As Long) As Long**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application. The only member of this type that you must set is the *guidApplication* member. Only some of the members of this type are used by this method. The only member that you must set is *guidApplication*. You can also set *guidInstance*, *pwszPassword*, *dwFlags*, and *dwSize***.**

Address
Optional **DirectPlay8Address** object that specifies the addressing information to use to connect to the computer that is hosting.

DeviceInfo
DirectPlay8Address object that specifies the network adapter (that is, NIC, modem, and so on) to use to connect to the server.

lFlags
Flag from the **CONST_DPNOPERATIONS** enumeration that describes the connection mode. You can set the following flag.

DPNOP_SYNC
Process the connection request synchronously. Setting this flag does not generate a **DirectPlay8Event.ConnectComplete** method call.

UserData
Application-specific user data.

UserDataSize
Size of the data contained in the *UserData* parameter.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Client.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDPASSWORD
DPNERR_INVALIDFLAGS

DPNERR_INVALIDINTERFACE
DPNERR_INVALIDAPPLICATION
DPNERR_NOTHOST
DPNERR_SESSIONFULL
DPNERR_HOSTREJECTEDCONNECTION
DPNERR_INVALIDINSTANCE

Remarks

Although multiple enumerations can be run concurrently, and can be run across the duration of a connection, only one connection is allowed per object. To establish a connection to more than one application, you must create another object. That is, only one running application per object is allowed. If **DirectPlay8Client.Connect** is called while another connection is in progress, the method raises an error.

DirectPlay8Client.EnumHosts

#Enumerates applications that host Microsoft® DirectPlay® games. When an application is found meeting the enumeration criteria, the **DirectPlay8Event.EnumHostsResponse** method is called in the application's message handler. This method contains a **DPNMSG_ENUM_HOSTS_RESPONSE** message type that contains a **DPN_APPLICATION_DESC** type that describes the applications found.

EnumHosts(*ApplicationDesc* **As DPN_APPLICATION_DESC, _**
AddrHost **As DirectPlay8Address, _**
DeviceInfo **As DirectPlay8Address, _**
lRetryCount **As Long, _**
lRetryInterval **As Long, _**
lTimeOut **As Long, _**
lFlags **As CONST_DPNOPERATIONS, _**
UserData **As Any, _**
UserDataSize **As Long) As Long**

Parts

ApplicationDesc
DPN_APPLICATION_DESC structure that specifies which application hosts to enumerate. You can specify the following fields to reduce the number of responses to the enumeration.

guidApplication
GUID of the application to find; if not specified, all are searched for.

Password

IDH_DirectPlay8Client.EnumHosts_dplay_vb

Password to provide; secure sessions will not respond without a password.

AddrHost
DirectPlay8Address object that specifies the address of the computer that is hosting the application.

DeviceInfo
DirectPlay8Address object that specifies the service provider and settings to enumerate.

lRetryCount
Value that specifies how many times the enumeration data will be sent. You can set this parameter to zero to specify the default value. If you set this value to INFINITE, the enumeration will continue until canceled.

lRetryInterval
Value that specifies the time, in milliseconds, between successive enumeration attempts. Set this parameter to 0 to use the default value.

lTimeOut
Number of milliseconds for the enumeration to run. If 0 is specified, a default value is used. If INFINITE is specified, the enumeration continues until it is canceled.

lEnumPeriod
Value specifying how often to re-enumerate. If 0 is specified, a default value is used.

lFlags
Flag from the **CONST_DPNOPERATIONS** enumeration that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

UserData
Block of data that is sent in the enumeration request to the host. The size of the data can be limited depending on the network type, but 512 bytes is supported at a minimum.

UserDataSize
Size of the data in the *UserData* parameter.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Client.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_TIMEDOUT

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

Because of the variety of ways enumeration can happen, it is not recommended that an application specify *lEnumPeriod* unless the application has some specific media knowledge.

DirectPlay8Client.GetApplicationDesc

#Retrieves the full application description for the connected application.

GetApplicationDesc([*lFlags* **As Long**]**)** _
As DPN_APPLICATION_DESC

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns a **DPN_APPLICATION_DESC** type describing the application.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDFLAGS
DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDPARAM

DirectPlay8Client.GetCaps

#Retrieves the **DPN_CAPS** type for the current object.

GetCaps([*lFlags* **As Long**]**) As DPN_CAPS**

Parts

lFlags

IDH_DirectPlay8Client.GetApplicationDesc_dplay_vb
IDH_DirectPlay8Client.GetCaps_dplay_vb

Reserved. Must be 0.

Return Values

Returns a **DPN_CAPS** type filled with session parameters.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Client.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Client.GetConnectionInfo

#Retrieves statistical information about the connection between the local endpoint and the server.

GetConnectionInfo([*lFlags* **As Long**]**) As DPN_CONNECTION_INFO**

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns a **DPN_CONNECTION_INFO** type to retrieve information about the specified connection.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM

IDH_DirectPlay8Client.GetConnectionInfo_dplay_vb

DPNERR_UNINITIALIZED

Remarks

This method can be called only after a successful **Connect** call has completed.

DirectPlay8Client.GetCountServiceProviders

#Retrieves the number of registered service providers available to the application.

GetCountServiceProviders([*lFlags* **As Long**]**) As Long**

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns the number of registered service providers available to the application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

DirectPlay8Client.GetSendQueueInfo

#Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® does not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should decrease the rate at which messages are sent.

GetSendQueueInfo(*lNumMsgs* **As Long,** _
lNumBytes **As Long,** _
[*lFlags* **As CONST_DPNGETSENDQUEUEINFO**]**)**

IDH_DirectPlay8Client.GetCountServiceProviders_dplay_vb
IDH_DirectPlay8Client.GetSendQueueInfo_dplay_vb

Parts

lNumMsgs
Number of messages currently queued.

lNumBytes
Amount of data, in bytes, of the messages currently queued.

lFlags
Set this parameter to one of the **CONST_DPNGETSENDQUEUEINFO** values to get the send-queue information for a particular priority level. Set this flag to zero to get the combined send-queue information.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Client.GetServerAddress

[#]Retrieves the address for the server for the session.

GetServerAddress([*lFlags* **As Long**]**) As DirectPlay8Address**

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns a **DirectPlay8Address** object specifying the address of the server.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

IDH_DirectPlay8Client.GetServerAddress_dplay_vb

DirectPlay8Client.GetServerInfo

#Retrieves the data set for the server set by the call to the **DirectPlay8Server.SetServerInfo** method.

GetServerInfo([*lFlags* **As Long]) As DPN_PLAYER_INFO**

Parts

lFlags
Set this parameter to one of the **CONST_DPNINFO** flags to indicate whether the method should return the name or the data set.

Return Values

Returns a **DPN_PLAYER_INFO** type containing the name or the data set for the server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_UNINITIALIZED

DirectPlay8Client.GetServiceProvider

#Retrieves information for the specified service provider. Before calling this method, call the **DirectPlay8Client.GetCountServiceProviders** method to obtain the number of registered service providers available to the application.

GetServiceProvider(*lIndex* **As Long) As DPN_SERVICE_PROVIDER_INFO**

Parts

lIndex
Index value specifying the specific server provider.

Return Values

Returns a **DPN_SERVICE_PROVIDER_INFO** type that describes the service provider.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

IDH_DirectPlay8Client.GetServerInfo_dplay_vb
IDH_DirectPlay8Client.GetServiceProvider_dplay_vb

DirectPlay8Client.GetSPCaps

#Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

GetSPCaps(*guidSP* **As String, _**
[*lFlags* **As Long]) As DPN_SP_CAPS**

Parts

guidSP
String specifying the GUID of the service provider you want to get information about.

lFlags
Reserved. Must be 0.

Return Values

Returns a **DPN_SP_CAPS** type to receive the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Client.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Client.RegisterLobby

#Registers or unregisters an application with a lobby.

RegisterLobby(*dpnHandle* **As Long, _**
***LobbyApp* As DirectPlay8LobbiedApplication, _**
[*lFlags* **As Long])**

IDH_DirectPlay8Client.GetSPCaps_dplay_vb
IDH_DirectPlay8Client.RegisterLobby_dplay_vb

Parts

dpnHandle
Connection handle to be used when making the calls to **DirectPlay8LobbiedApplication.UpdateStatus**.

LobbyApp
DirectPlay8LobbiedApplication object that specifies the application.

lFlags
Set this parameter to one of the two **CONST_DPNLOBBY** enumeration values to indicate whether the application is to be registered or unregistered.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

Remarks

When an application is registered with a lobby, the lobby client automatically receives notifications of changes in game status.

DirectPlay8Client.RegisterMessageHandler

#Registers an entry point in the client's code that receives the messages from the **DirectPlay8Client** object and from the server. This method must be called before calling any other methods of this class.

RegisterMessageHandler(*event* **As DirectPlay8Event)**

Parts

event
DirectPlay8Event object that will receive the messages generated by the client and server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_ALREADYREGISTERED

IDH_DirectPlay8Client.RegisterMessageHandler_dplay_vb

Remarks

If you want to register a new message handler, you must first call **DirectPlay8Client.UnRegisterMessageHandler** to unregister the current message handler.

DirectPlay8Client.Send

#Transmits data to the server. The message can be sent synchronously or asynchronously.

Send(*buffer()* **As BYTE, _**
lTimeOut **As Long, _**
[*lFlags* **As CONST_DPNSENDFLAGS]) As Long**

Parts

buffer()
Array of type **BYTE** that describes the data to send.

lTimeOut
Number of milliseconds to wait for the message to be sent. If the message has not been sent by the *lTimeOut* value, the message is not sent. If you do not want a time out for message sends, set this parameter to 0.

lFlags
Flags that describe send behavior. You can set one or more of the following flags defined in the **CONST_DPNSENDFLAGS** enumeration.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Client.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS
DPNERR_TIMEDOUT

Remarks

This method will call the **DirectPlay8Event.Receive** method in the server's message handler. When the **Send** request is completed, the **DirectPlay8Event.SendComplete** method is called in the client's message handler. The **SendComplete** method contains

---

IDH_DirectPlay8Client.Send_dplay_vb

a **DPNMSG_SEND_COMPLETE** message type. The success or failure of the request is contained in the **hResultCode** member of this message type.

DirectPlay8Client.SetCaps

#Sets the capabilities for the session.

SetCaps(*caps* **As DPN_CAPS, _**
[*lFlags* **As Long])**

Parts

caps
DPN_CAPS type used to set the information about the current session.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT

DPNERR_INVALIDPOINTER

DPNERR_INVALIDPARAM

DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Client.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Client.SetClientInfo

#Sets the static settings of a client with an application. Call this method before connecting to relay basic player information with the application. After the client successfully connects with the application, information obtained through this method can be retrieved by the server by calling the **DirectPlay8Server.GetClientInfo** method.

SetClientInfo(*PlayerInfo* **As DPN_PLAYER_INFO, _**
[*lFlags* **As CONST_DPNOPERATIONS]) As Long**

Parts

PlayerInfo

IDH_DirectPlay8Client.SetCaps_dplay_vb
IDH_DirectPlay8Client.SetClientInfo_dplay_vb

DPN_PLAYER_INFO type that contains the peer information to set.

lFlags

Set the DPNSETCLIENTINFO_SYNC flag from the CONST_DPNOPERATIONS enumeration to have the method process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Client.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

Remarks

Transmission of nonstatic information should be handled with the **DirectPlay8Client.Send** method because of the high cost of using the **DirectPlay8Client.SetClientInfo** method.

You can modify the client information with this method after connecting to the application. Calling this method after connection will call the **DirectPlay8Event.InfoNotify** method in the server's message handler.

DirectPlay8Client.SetSPCaps

[#]Sets the capabilities for the specified service provider.

SetSPCaps(*guidSP* **As String, _**
spCaps **As DPN_SP_CAPS, _**
[*lFlags* **As Long**]**)**

Parts

guidSP

String specifying the GUID of the service provider you want to set information about.

spCaps

DPN_SP_CAPS type to set the information about the specified service provider.

lFlags

IDH_DirectPlay8Client.SetSPCaps_dplay_vb

Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Client.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Client.UnregisterMessageHandler

#Unregisters the current **DirectPlay8Client** message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlay8Event

#Applications use the methods of the **DirectPlay8Event** class to capture Microsoft® DirectPlay® events generated by the **DirectPlay8Peer**, **DirectPlay8Client**, and **DirectPlay8Server** classes.

DirectPlay8Event is associated with the **DirectPlay8Peer**, **DirectPlay8Client**, and **DirectPlay8Server** objects when calling **RegisterMessageHandler** for each object.

Note

When implementing the **DirectPlay8Event** object, Microsoft® Visual Basic® requires that every method of this object is implemented in your form.

The methods of the **DirectPlay8Event** class are:

DirectPlay8Event Methods	**AddRemovePlayerGroup**
	AppDesc

IDH_DirectPlay8Client.UnregisterMessageHandler_dplay_vb
IDH_DirectPlay8Event_dplay_vb

AsyncOpComplete
ConnectComplete
CreateGroup
CreatePlayer
DestroyGroup
DestroyPlayer
EnumHostsQuery
EnumHostsResponse
HostMigrate
IndicateConnect
IndicateConnectAborted
InfoNotify
Receive
SendComplete
TerminateSession

DirectPlay8Event.AddRemovePlayerGroup

#Called by the associated **DirectPlay8Peer**, **DirectPlay8Client**, or **DirectPlay8Server** object when a player or group has been added to or removed from a group.

AddRemovePlayerGroup(_
lMsgID **As Long, _**
lPlayerID **As Long, _**
lGroupID **As Long, _**
fRejectMsg **As Boolean)**

Parts

lMsgID
One of the following flags, which indicate whether a player or group has joined or has left the session:
DPN_MSGID_ADD_PLAYER_TO_GROUP
A player has joined the group.
DPN_MSGID_REMOVE_PLAYER_FROM_GROUP
A player has left the group.

lPlayerID

IDH_DirectPlay8Event.AddRemovePlayerGroup_dplay_vb

Long value set to the ID of the player or group being added or removed.

lGroupID

Long value set to the ID of the group this player or group is being added to or removed from.

fRejectMsg

Parameter is not supported in this release.

DirectPlay8Event.AppDesc

#Called when the application description has been changed by a call to the **DirectPlay8Peer.SetApplicationDesc** or **DirectPlay8Server.SetApplicationDesc** method.

AppDesc(*fRejectMsg* **As Boolean)**

Parts

fRejectMsg

Parameter is not supported in this release.

DirectPlay8Event.AsyncOpComplete

#Called when an asynchronous operation has completed.

AsyncOpComplete(_
dpnotify **As DPNMSG_ASYNC_OP_COMPLETE, _**
fRejectMsg **As Boolean)**

Parts

dpnotify

DPNMSG_ASYNC_OP_COMPLETE message type containing the handle and the result of the asynchronous operation.

fRejectMsg

Parameter is not supported in this release.

DirectPlay8Event.ConnectComplete

#Called when the attempt to connect to the host or server has completed.

ConnectComplete(_

IDH_DirectPlay8Event.AppDesc_dplay_vb
IDH_DirectPlay8Event.AsyncOpComplete_dplay_vb
IDH_DirectPlay8Event.ConnectComplete_dplay_vb

dpnotify **As DPNMSG_CONNECT_COMPLETE,**
fRejectMsg **As Boolean)**

Parts

dpnotify
DPNMSG_CONNECT_COMPLETE message type containing the async handle and the result of the connection attempt.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.CreateGroup

#Called when a group is created.

CreateGroup (_
lGroupID ***As Long,*** **_**
lOwnerID ***As Long,*** **_**
fRejectMsg **As Boolean)**

Parts

lGroupID
Long value set to the group's ID.

lOwnerID
Long value set to the owner's ID.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.CreatePlayer

#Called when a player is created.

CreatePlayer (_
lPlayerID **As Long, _**
fRejectMsg **As Boolean)**

Parts

lPlayerID
Long value set to the player's ID.

fRejectMsg
Parameter is not supported in this release.

IDH_DirectPlay8Event.CreateGroup_dplay_vb
IDH_DirectPlay8Event.CreatePlayer_dplay_vb

DirectPlay8Event.DestroyGroup

#Called when a group is destroyed.

```
DestroyGroup( _
    lGroupID As Long, _
    lReason As Long, _
    fRejectMsg As Boolean)
```

Parts

lGroupID
Long value set to the group's ID.

lReason
Long value that indicates the reason that the group was destroyed.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.DestroyPlayer

#Called when a player is destroyed.

```
DestroyPlayer( _
    lPlayerID As Long, _
    lReason As Long, _
    fRejectMsg As Boolean)
```

Parts

lPlayerID
Long value set to the player's ID.

lReason
Long value that indicates the reason that the group was destroyed.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.EnumHostsQuery

#Called when host enumeration is requested through the **DirectPlay8Peer.EnumHosts** or **DirectPlay8Client.EnumHosts** method.

```
EnumHostsQuery( _
    dpnotify As DPNMSG_ENUM_HOSTS_QUERY, _
    fRejectMsg As Boolean)
```

IDH_DirectPlay8Event.DestroyGroup_dplay_vb
IDH_DirectPlay8Event.DestroyPlayer_dplay_vb
IDH_DirectPlay8Event.EnumHostsQuery_dplay_vb

Parts

dpnotify
DPNMSG_ENUM_HOSTS_QUERY message type containing received data.

fRejectMsg
Boolean value that is set to True to reject the enumeration request. However, if you reject the query, Microsoft® DirectPlay® will not call the client's **DirectPlay8Event.EnumHostsResponse**, so they will not be aware that you are hosting a session.

DirectPlay8Event.EnumHostsResponse

#Contains the host information resulting from a call to the **DirectPlay8Peer.EnumHosts** or **DirectPlay8Client.EnumHosts** method.

```
EnumHostsResponse( _
    dpnotify As DPNMSG_ENUM_HOSTS_RESPONSE, _
    fRejectMsg As Boolean)
```

Parts

dpnotify
DPNMSG_ENUM_HOSTS_RESPONSE message type containing host information.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.HostMigrate

#Called when the host has changed in a session by a call to the **DirectPlay8Peer.Host** or **DirectPlay8Server.Host** method.

```
HostMigrate(NewHostID As Long, _
    fRejectMsg As Boolean)
```

Parts

NewHostID
Long value set to the ID of the new host.

fRejectMsg
Parameter is not supported in this release.

IDH_DirectPlay8Event.EnumHostsResponse_dplay_vb
IDH_DirectPlay8Event.HostMigrate_dplay_vb

DirectPlay8Event.IndicateConnect

#Called on the host's message handler when a player has requested a connection to the session.

```
IndicateConnect( _
    dpnotify As DPNMSG_INDICATE_CONNECT, _
    fRejectMsg As Boolean)
```

Parts

dpnotify
DPNMSG_INDICATE_CONNECT message type.

fRejectMsg
Boolean value that is set to False to enable the player to join the session and True to reject the player's request.

Remarks

After this method returns, the player's **DirectPlay8Event.ConnectComplete** method will be called with the response to their request to join the session. If the connection request was successful, the **hResultCode** member of the **DPNMSG_CONNECT_COMPLETE** type will be set to 0. If the request was rejected or failed, **hResultCode** will be set to an error code.

DirectPlay8Event.IndicateConnectAborted

#Called if a player's connection is lost after it was indicated on the host, but prior to being added to the session.

```
IndicateConnectAborted( _
    fRejectMsg As Boolean)
```

Parts

fRejectMsg
Parameter is not supported in this release.

IDH_DirectPlay8Event.IndicateConnect_dplay_vb
IDH_DirectPlay8Event.IndicateConnectAborted_dplay_vb

DirectPlay8Event.InfoNotify

#Called when the information for a client, server, peer, or group has been updated.

InfoNotify(*lMsgID* **As Long, _**
lNotifyID **As Long, _**
fRejectMsg **As Boolean)**

Parts

lMsgID
This will be set to one of the following constants.
DPN_MSGID_CLIENT_INFO
The client information has changed.
DPN_MSGID_GROUP_INFO
The group information has changed.
DPN_MSGID_PEER_INFO
The peer information has changed.
DPN_MSGID_SERVER_INFO
The server information has changed.

lNotifyID
Notification ID.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.Receive

#Called in the receiver's message handler when a message is received from a player or host.

Receive(_
dpnotify **As DPNMSG_RECEIVE,**
fRejectMsg **As Boolean)**

Parts

dpnotify
DPNMSG_RECEIVE message type containing the data sent.

fRejectMsg
Parameter is not supported in this release.

IDH_DirectPlay8Event.InfoNotify_dplay_vb
IDH_DirectPlay8Event.Receive_dplay_vb

DirectPlay8Event.SendComplete

#Called in the sender's message handler when the message is received by the recipient.

```
SendComplete( _
   dpnotify As DPNMSG_SEND_COMPLETE, _
   fRejectMsg As Boolean)
```

Parts

dpnotify
DPNMSG_SEND_COMPLETE message type containing the result of the message send and the time it took to send the message.

fRejectMsg
Parameter is not supported in this release.

DirectPlay8Event.TerminateSession

#Called in the message handler for each session player after a call to **DirectPlay8Peer.TerminateSession**.

```
TerminateSession( _
dpnotify As DPNMSG_TERMINATE_SESSION, _
fRejectMsg As Boolean)
```

Parts

dpnotify
DPNMSG_TERMINATE_SESSION message type containing the result of the message send.

fRejectMsg
Parameter is not supported in this release.

Remarks

In a peer-peer session that permits host-migration, the host migrates if the current host calls **DirectPlay8Peer.Close** or stops responding. When this event occurs:

- Microsoft® DirectPlay® notifies all the other players that the host has left the session by calling their **DirectPlay8Event.DestroyPlayer** method.
- DirectPlay then notifies the remaining players of the new host by calling their **DirectPlay8Event.HostMigrate** method

---

IDH_DirectPlay8Event.SendComplete_dplay_vb
IDH_DirectPlay8Event.TerminateSession_dplay_vb

To prevent host-migration, the current host must terminate the session by calling **DirectPlay8Peer.TerminateSession**. When the host terminates a session this way:

- DirectPlay calls each players' **DirectPlay8Event.TerminateSession** method. The **hResultCode** of the associated **DPNMSG_TERMINATE_SESSION** type will be set to DPNERR_HOSTTERMINATEDSESSION.
- DirectPlay will then generate **DirectPlay8Event.DestroyPlayer** calls for each player.

In a peer-peer game that does not permit host-migration, the session is terminated if the host calls **DirectPlay8Peer.Close**, or stops responding. When this event occurs:

- DirectPlay calls each player's **DirectPlay8Event.TerminateSession** method. The **hResultCode** of the associated **DPNMSG_TERMINATE_SESSION** type will be set to set to DPNERR_HOSTTERMINATEDSESSION.
- DirectPlay will then generate **DirectPlay8Event.DestroyPlayer** calls for each player.

In a client/server game, the session is terminated if the host calls **DirectPlay8Server.Close** or stops responding. When this event occurs:

- DirectPlay calls each player's **DirectPlay8Event.TerminateSession** method. The **hResultCode** of the associated **DPNMSG_TERMINATE_SESSION** type will be set to DPNERR_CONNECTIONLOST.
- If the server disconnected itself from the session by calling **DirectPlay8Server.Close**, its **DirectPlay8Event.DestroyPlayer** method will be called once for each player in the session, including its own player.
- Otherwise, the server's **DirectPlay8Event.DestroyPlayer** method is called for each client's player, but not for the server's player.

DirectPlay8LobbiedApplication

[#]The **DirectPlay8LobbiedApplication** class is used by an application that supports lobbying. This class enables the application to register with the system so that it can be lobby launched. Additionally, it also enables the application to get the connection information necessary to launch a game without querying the user. Lastly, this class enables the lobbied application to send messages and notifications to the lobby client that launched the application.

The methods of the **DirectPlay8LobbiedApplication** class are:

DirectPlay8LobbiedApplication Methods	**Close**
	GetConnectionSettings
	RegisterMessageHandler

[#] IDH_DirectPlay8LobbiedApplication_dplay_vb

RegisterProgram
Send
SetAppAvailable
SetConnectionSettings
UnRegisterMessageHandler
UnRegisterProgram
UpdateStatus

DirectPlay8LobbiedApplication.Close

#Deletes the lobbied application.

Close()

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

DirectPlay8LobbiedApplication.GetConnectionSettings

#Retrieves the settings for the specified connection.

GetConnectionSettings(*LobbyClient* **As Long,** *lFlags* **As Long)**

Parts

LobbyClient
Handle to the connection to retrieve the settings for.

lFlags
Reserved, must be 0.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM
DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDOBJECT
DPNERR_INVALIDFLAGS

IDH_DirectPlay8LobbiedApplication.Close_dplay_vb
IDH_DirectPlay8LobbiedApplication.GetConnectionSettings_dplay_vb

DirectPlay8LobbiedApplication.Register MessageHandler

#Registers a message handler function that receives notifications about changes in the state of the lobbied application and receives messages from the lobby client.

RegisterMessageHandler(*lobbyEvent* **As DirectPlay8LobbyEvent)**

Parts

lobbyEvent
DirectPlay8LobbyEvent object that is used to receive notifications.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_ALREADYREGISTERED
DPNERR_INVALIDPARAM

Remarks

If you want to register a new message handler, you must first call **DirectPlay8LobbiedApplication.UnRegisterMessageHandler** to unregister the current message handler.

DirectPlay8LobbiedApplication.Register Program

#Registers a lobby-aware application with Microsoft® DirectPlay®. Applications must be registered to enable lobby launching.

RegisterProgram(_
ProgramDesc **As DPL_PROGRAM_DESC, _**
lFlags **As Long)**

Parts

ProgramDesc
DPL_PROGRAM_DESC type that describes the lobby-aware application to register.

lFlags
Reserved. Must be 0.

IDH_DirectPlay8LobbiedApplication.RegisterMessageHandler_dplay_vb
IDH_DirectPlay8LobbiedApplication.RegisterProgram_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

The application needs to register only once. It should be unregistered with a call to the **DirectPlay8LobbiedApplication.UnRegisterProgram** method when it is uninstalled. If a lobby client tries to launch an application that no longer exists on the system, DirectPlay automatically unregisters the application.

In DirectX® 8.0, **RegisterProgram** must be used. You cannot manually place application information into the registry. Failure to use this class makes your application nonportable and incompatible with future versions of DirectPlay.

DirectPlay8LobbiedApplication.Send

[#]Sends a message from the lobbied application to the lobby client.

Send(*Target* **As Long, _**
buffer() **As Byte, _**
lBufferSize **As Long, _**
lTimeOut **As Long, _**
lFlags **As Long)**

Parts

Target
Long value specifying the handle of the lobby client to receive the message.

buffer()
Variable of type **BYTE** that contains the message buffer.

lBufferSize
Long value that specifies the size of the message buffer in the *buffer()* parameter, in bytes. This value must be at least 1 byte and no more than 64 KB.

lTimeOut
Long value that specifies the number of milliseconds to wait for the **Send** request to process.

lFlags
Reserved. Must be 0.

IDH_DirectPlay8LobbiedApplication.Send_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_SENDTOOLARGE

Remarks

If the buffer size is larger than 64 KB, **Err.Number** is set to DPNERR_SENDTOOLARGE. If the buffer size is set to 0, **Err.Number** is set to DPNERR_INVALIDPARAM.

DirectPlay8LobbiedApplication.SetAppAvailable

#Makes an application available or unavailable for a lobby client to connect to. This method is typically called if a lobbied application is independently launched, that is, not launched by a lobby client. Additionally, this method should be called if a game has ended and the lobbied application needs to be available to connect to a lobby client at the start of another game.

SetAppAvailable(*fAvailable* **As Boolean)**

Parts

fAvailable

Boolean value to set to TRUE if the application is available, or FALSE to make it unavailable.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

DirectPlay8LobbiedApplication.SetConnectionSettings

#Specifies the settings for the connection.

SetConnectionSettings(*hTarget* **As Long**, _

IDH_DirectPlay8LobbiedApplication.SetAppAvailable_dplay_vb
IDH_DirectPlay8LobbiedApplication.SetConnectionSettings_dplay_vb

lFlags **As Long**, _
ConnectionSettings As **DPL_CONNECTION_SETTINGS**, _
HostAddress **As DirectPlay8Address**, _
Device **As DirectPlay8Address)**

Parts

hTarget
Long value that is set to the connection handle.

lFlags
Reserved, must be 0.

ConnectionSettings
DPL_CONNECTION_SETTINGS type with the connection settings.

HostAddress
DirectPlay8Address object with the host address.

Device
DirectPlay8Address object with the device address.

DirectPlay8LobbiedApplication.UnRegisterMessageHandler

#Unregisters the current **DirectPlay8LobbiedApplication** message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlay8LobbiedApplication.UnRegisterProgram

#Unregisters a lobby-aware application that was registered through the **DirectPlay8LobbiedApplication.RegisterProgram** method.

UnRegisterProgram(*guidApplication* **As String,** *lFlags* **As Long)**

Parts

guidApplication
GUID of the application to unregister.

lFlags
Reserved. Must be 0.

IDH_DirectPlay8LobbiedApplication.UnRegisterMessageHandler_dplay_vb
IDH_DirectPlay8LobbiedApplication.UnRegisterProgram_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8LobbiedApplication.UpdateStatus

[#]Updates the status of a connected lobby client.

UpdateStatus(*LobbyClient* **As Long, _**
***lStatus* As CONST_DPLSESSION)**

Parts

LobbyClient
Long value that specifies the connection handle for the lobby client.

lStatus
One of the constants of the **CONST_DPLSESSION** enumeration that specifies the status between the lobby client and the lobbied application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

Remarks

You define the connection handle when you register the application with the lobby by calling **DirectPlay8Client.RegisterLobby** or **DirectPlay8Peer.RegisterLobby**.

DirectPlay8LobbyClient

[#]The **DirectPlay8LobbyClient** class is used by a lobby client application and is responsible for enumerating and launching lobby-enabled game applications on the local computer, and communicating with them after they are running. The lobby client must register a message handler routine to process messages from the lobby and the lobbied game application.

The methods of the **DirectPlay8LobbyClient** class are:

IDH_DirectPlay8LobbiedApplication.UpdateStatus_dplay_vb
IDH_DirectPlay8LobbyClient_dplay_vb

DirectPlay8LobbyClient Methods

Close

ConnectApplication
GetCountLocalPrograms
GetLocalProgram
RegisterMessageHandler
ReleaseApplication
Send
SetConnectionSettings
UnRegisterMessageHandler

DirectPlay8LobbyClient.Close

#Deletes the lobby client.

```
Close()
```

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

DirectPlay8LobbyClient.ConnectApplication

#Connects a lobby-enabled application to the session specified in the **DPL_CONNECT_INFO** type. If the application is not running, this method can be used to launch the application.

When the connection is successfully established, Microsoft® DirectPlay® calls the message handler's **DirectPlay8LobbyEvent.Connect** method.

```
ConnectApplication( _
   ConnectionInfo As DPL_CONNECT_INFO,
   lTimeOut As Long, _
   lFlags As CONST_DPLCONNECT) As Long
```

Parts

ConnectionInfo
DPL_CONNECT_INFO type that describes the connection parameters, including the GUID of the application to connect to.

IDH_DirectPlay8LobbyClient.Close_dplay_vb
IDH_DirectPlay8LobbyClient.ConnectApplication_dplay_vb

lTimeOut
Long value that specifies the number of milliseconds to wait for the connection to process.

lFlags
One of the constants of the **CONST_DPLCONNECT** enumeration that determines connection behavior.

Return Values

Returns a **Long** value that is set to the application connection handle . This handle is used for further communication with the application. Additionally, this handle is used in the *Application* parameter in the **DirectPlay8LobbyClient.ReleaseApplication** method.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_CANTLAUNCHAPPLICATION
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM
DPNERR_TIMEDOUT

DirectPlay8LobbyClient.GetCountLocalPrograms

[#]Retrieves the number of lobbied applications that are registered on the system.

GetCountLocalPrograms(*guidApplication* **As String) As Long**

Parts

guidApplication
String value specifying the **GUID** of the lobbied application to enumerate.

Return Values

Returns the number of lobbied applications that are registered.

DirectPlay8LobbyClient.GetLocalProgram

IDH_DirectPlay8LobbyClient.GetCountLocalPrograms_dplay_vb

#Enumerates the specified lobbied application that is registered on the system. Before calling this method, call **DirectPlay8LobbyClient.GetCountLocalPrograms** to see how many lobbied applications are registered.

GetLocalProgram(*lProgID* **As Long) As DPL_APPLICATION_INFO**

Parts

lProgID
Long value specifying the ID of the lobbied application to enumerate.

Return Values

Returns a **DPL_APPLICATION_INFO** structure that describes the lobbied application.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

This method is generally called twice—once to obtain the size of the required buffer, and then with the correct buffer size.

DirectPlay8LobbyClient.RegisterMessageHandler

#Registers a message handler function that receives notifications about changes in the state of the lobby client and receives messages from the lobbied application.

RegisterMessageHandler(*lobbyEvent* **As DirectPlay8LobbyEvent)**

Parts

lobbyEvent
DirectPlay8LobbyEvent object that is used to receive notifications.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_ALREADYREGISTERED

IDH_DirectPlay8LobbyClient.GetLocalProgram_dplay_vb
IDH_DirectPlay8LobbyClient.RegisterMessageHandler_dplay_vb

DPNERR_INVALIDPARAM

Remarks

If you want to register a new message handler, you must first call **DirectPlay8LobbyClient.UnRegisterMessageHandler** to unregister the current message handler.

DirectPlay8LobbyClient.ReleaseApplication

#Releases a lobbied application and closes the connection between the lobby client and the application. This method should be called whenever a lobby client has finished its session with an application.

ReleaseApplication(*Application* **As Long)**

Parts

Application
Long value specifying the GUID of the lobbied application to release. This value is returned by the **DirectPlay8LobbyClient.ConnectApplication** method.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDHANDLE
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

DirectPlay8LobbyClient.Send

#Sends a message to a lobbied application that was launched by this lobby client or was connected by this lobby client.

Microsoft® DirectPlay® transmits the data to the target by calling the target message handler's **DirectPlay8LobbyEvent.Receive** method.

Send(*Target* **As Long, _**
buffer() **As Byte, _**
lBufferSize **As Long, _**
lFlags **As Long)**

IDH_DirectPlay8LobbyClient.ReleaseApplication_dplay_vb
IDH_DirectPlay8LobbyClient.Send_dplay_vb

Parts

Target
Long value that specifies the target for the message transmission.

buffer()
Buffer that contains the message.

lBufferSize
Size of the message buffer in the *buffer()* parameter, in bytes. This value must be at least 1 byte and no more than 64 KB.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDHANDLE

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPARAM

DPNERR_SENDTOOLARGE

Remarks

If the buffer size is larger than 64 KB, **Err.Number** is set to DPNERR_SENDTOOLARGE. If the buffer size is set to 0, **Err.Number** is set to DPNERR_INVALIDPARAM.

DirectPlay8LobbyClient.SetConnectionSettings

[#]Sets the connection settings to be associated with the specified connection.

SetConnectionSettings(*hTarget* **As Long, _**
lFlags **As Long, _**
ConnectionSettings **As DPL_CONNECTION_SETTINGS), _**
HostAddress **As DirectPlay8Address, _**
Device **As DirectPlay8Address)**

Parts

hTarget

IDH_DirectPlay8LobbyClient.SetConnectionSettings_dplay_vb

Long value that is set to the connection handle.

lFlags
Reserved. Set to 0.

ConnectionSettings
DPL_CONNECTION_SETTINGS structure that contains the connections.

HostAddress
DirectPlay8Address object containing the host address.

Device
DirectPlay8Address object containing the device address.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDPARAM
DPNERR_INVALIDOBJECT
DPNERR_INVALIDFLAGS

DirectPlay8LobbyClient.UnRegisterMessageHandler

#Unregisters the current **DirectPlay8LobbyClient** message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlay8LobbyEvent

#The **DirectPlay8LobbyEvent** class is used to capture Microsoft® DirectPlay® events generated when using the **DirectPlay8LobbiedApplication** and **DirectPlay8LobbyClient** classes.

Note

When implementing the **DirectPlay8LobbyEvent** object, Microsoft® Visual Basic® requires that every method of this object is implemented in your form.

The methods of the **DirectPlay8LobbyEvent** class are:

DirectPlay8LobbyEvent Methods **Connect**

IDH_DirectPlay8LobbyClient.UnRegisterMessageHandler_dplay_vb
IDH_DirectPlay8LobbyEvent_dplay_vb

ConnectionSettings
Disconnect
Receive
SessionStatus

DirectPlay8LobbyEvent.Connect

#Called when the **DirectPlay8LobbyClient.ConnectApplication** method has been called by the lobby client.

Connect(_
dlNotify **As DPL_MESSAGE_CONNECT, _**
fRejectMsg **As Boolean)**

Parts

dlNotify
DPL_MESSAGE_CONNECT message type with the connection information.

fRejectMsg
Boolean value indicating whether to connect the application.

DirectPlay8LobbyEvent.ConnectionSettings

#Called when the **DirectPlay8LobbyClient.SetConnectionSettings** or **DirectPlay8LobbiedApplication.SetConnectionSettings** method is called.

ConnectionSettings(*_ConnectionSettings* **As DPL_MESSAGE_CONNECTION_SETTINGS)**

Parts

ConnectionSettings
DPL_MESSAGE_CONNECTION_SETTINGS message type with the connection settings.

DirectPlay8LobbyEvent.Disconnect

#Called when the **DirectPlay8LobbyClient.ReleaseApplication** method has been called by the lobby client.

IDH_DirectPlay8LobbyEvent.Connect_dplay_vb
IDH_DirectPlay8LobbyEvent.ConnectionSettings_dplay_vb
IDH_DirectPlay8LobbyEvent.Disconnect_dplay_vb

Disconnect(*DisconnectID* **As Long, _**
lReason **As Long)**

Parts

DisconnectID
Long value specifying the identifier of the lobby application that requested to disconnect.

lReason
Long value that is set to the reason for the disconnection.

DirectPlay8LobbyEvent.Receive

[#]Called when the **DirectPlay8LobbyClient.Send** or **DirectPlay8LobbiedApplication.Send** method has been called.

Receive(*dlNotify* **As DPL_MESSAGE_RECEIVE, _**
fRejectMsg **As Boolean)**

Parts

dlNotify
DPL_MESSAGE_RECEIVE message type containing the message sent.

fRejectMsg
Boolean value indicating whether to accept or reject the message.

DirectPlay8LobbyEvent.SessionStatus

[#]Called when the **DirectPlay8LobbiedApplication.UpdateStatus** method has been called by the lobbied application.

LobbySessionStatus(*status* **As Long, _**
lHandle **As Long)**

Parts

status
Long value specifying the updated status of the lobby session. It will be set to one of the constants of the **CONST_DPLSESSION** enumeration that specifies the status between the lobby client and the lobbied application.

lHandle

IDH_DirectPlay8LobbyEvent.Receive_dplay_vb
IDH_DirectPlay8LobbyEvent.SessionStatus_dplay_vb

Long value set to the handle of the application that sent the status update message.

DirectPlay8Peer

[#]Applications use the methods of the **DirectPlay8Peer** class to create a peer-to-peer Microsoft® DirectPlay® session.

The methods of the **DirectPlay8Peer** class can be organized into the following groups.

Session Management	**Close**
	Connect
	EnumHosts
	GetApplicationDesc
	GetCaps
	GetConnectionInfo
	GetCountServiceProviders
	GetEnumHostResponseAddress
	GetServiceProvider
	GetSPCaps
	Host
	SetApplicationDesc
	SetCaps
	SetSPCaps
	TerminateSession
Message Management	**GetSendQueueInfo**
	RegisterMessageHandler
	SendTo
	UnregisterMessageHandler
Player Management	**DestroyPeer**
	GetPlayerOrGroup
	GetCountClientsAndGroups
	GetPeerAddress
	GetPeerInfo
	RemovePlayerFromGroup
	SetPeerInfo
Group Management	**AddPlayerToGroup**

[#] IDH_DirectPlay8Peer_dplay_vb

	CreateGroup
	DestroyGroup
	GetCountGroupMembers
	GetGroupInfo
	GetGroupMember
	SetGroupInfo
Miscellaneous	**CancelAsyncOperation**
	RegisterLobby

DirectPlay8Peer.AddPlayerToGroup

[#]Adds a peer to a group.

When this method is called, DirectPlay calls each player's **DirectPlay8Event.AddRemovePlayerGroup** method to notify him or her of the new group member.

AddPlayerToGroup(*idGroup* **As Long, _**
idClient **As Long,**
lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group to add the peer to.

idClient
Long value that specifies the identifier of the peer that is added to the group.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

IDH_DirectPlay8Peer.AddPlayerToGroup_dplay_vb

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

Remarks

Any peer can add itself or another peer to an existing group. After the peer is successfully added to the group, all messages sent to the group are also sent to the peer.

For a peer to add itself to the group, pass DPNID_ME in the *idClient* parameter.

DirectPlay8Peer.CancelAsyncOperation

[#]Cancels asynchronous requests. Many methods of the **DirectPlay8Peer** class run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this class that can be run asynchronously return an *lAsyncHandle* parameter.

Specific requests are canceled by passing the *lAsyncHandle* of the request in this method's *lAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, passing 0 in the *lAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *lFlags* parameter. If a specific handle is provided to this method, you must pass 0 in the *lFlags* parameter.

CancelAsyncOperation(*lAsyncHandle* **As Long, _**
***lFlags* As CONST_DPNCANCELFLAGS)**

Parts

lAsyncHandle
: Handle of the asynchronous operation to stop. This value can be 0 to stop all requests or a particular type of asynchronous request. If a specific handle for the request to cancel is specified, the *lFlags* parameter must be 0.

lFlags
: Flag that specifies which asynchronous request to canceled. You can set this parameter to one of the flags of the **CONST_DPNCANCELFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_PENDING
DPNERR_INVALIDFLAGS
DPNERR_CANNOTCANCEL

IDH_DirectPlay8Peer.CancelAsyncOperation_dplay_vb

DPNERR_INVALIDHANDLE

Remarks

You can cancel a send by providing the handle returned from **DirectPlay8Peer.SendTo** method. The **DirectPlay8Event.SendComplete** method will still be called unless the message was sent with the DPNSEND_NOCOMPLETE flag set. If you cancel a send operation by calling **DirectPlay8Peer.CancelAsyncOperation** the **hResultCode** member of the **DPNMSG_SEND_COMPLETE** type that is passed to the **DirectPlay8Event.SendComplete** method will be set to DPNERR_CANCELLED.

DirectPlay8Peer.Close

#Closes the open connection with a session.

```
Close();
```

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_UNINITIALIZED

DirectPlay8Peer.Connect

#Establishes the connection to the server. After a connection is established, the communication channel on the object is open and active and the application should expect messages to arrive immediately. No messages can be sent by means of the **DirectPlay8Peer.SendTo** method until the connection has completed.

Before this method is called, it must obtain an application description by calling **DirectPlay8Peer.EnumHosts**. The **EnumHosts** method returns a **DPN_APPLICATION_DESC** type for each hosted application. The type describes the application, including the GUID of the application.

Although the application GUID is not required, Microsoft® DirectPlay® performs verification if one is specified.

When the connection to the host is requested, the **DirectPlay8Event.IndicateConnect** method is called in the host's message handler. The host may either accept or reject the connection. In either case, after the host has acted, the client message handler's **DirectPlay8Event.ConnectComplete** method will be called to convey the response.

```
Connect(AppDesc As DPN_APPLICATION_DESC, _
   Address As DirectPlay8Address, _
```

IDH_DirectPlay8Peer.Close_dplay_vb
IDH_DirectPlay8Peer.Connect_dplay_vb

DeviceInfo **As DirectPlay8Address, _**
lFlags **As Long, _**
UserData **As Any, _**
UserDataSize **As Long) As Long**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application. Only some of the members of this type are used by this method. The only member of this type that you must set is the *guidApplication* member. You can also set *guidInstance*, *Password*, and *lFlags***.**

Address
Optional **DirectPlay8Address** object that specifies the addressing information to use to connect to the computer that is hosting.

DeviceInfo
DirectPlay8Address object that specifies what service provider to enumerate and what settings to use.

lFlags
Flag that describes the connection mode. You can set the following flag.

DPNOP_SYNC
Process the connection request synchronously. If this flag is set, DirectPlay will not call the **DirectPlay8Event.ConnectComplete** method.

UserData
Application-specific user data.

UserDataSize
Size of the data contained in the *UserData* parameter.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDPASSWORD
DPNERR_INVALIDFLAGS
DPNERR_INVALIDINTERFACE
DPNERR_INVALIDAPPLICATION
DPNERR_NOTHOST

DPNERR_SESSIONFULL
DPNERR_HOSTREJECTEDCONNECTION
DPNERR_INVALIDINSTANCE

Remarks

Although multiple enumerations can be run concurrently, and can be run across the duration of a connection, only one connection is allowed per object. To establish a connection to more than one application, you must create another object. That is, only one running application per object is allowed. If **DirectPlay8Peer.Connect** is called while another connection is in progress, the function raises an error.

DirectPlay8Peer.CreateGroup

#Creates a group in the current session. A group is a logical collection of players.

When this method is called, DirectPlay calls each player's **DirectPlay8Event.CreateGroup** method to notify him or her of the new group.

CreateGroup(*GroupInfo* **As DPN_GROUP_INFO, _**
***lFlags* As Long) As Long**

Parts

GroupInfo
DPN_GROUP_INFO structure that contains the group description.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

IDH_DirectPlay8Peer.CreateGroup_dplay_vb

Remarks

Microsoft® DirectPlay® does not maintain hierarchical groups because they can easily be implemented with flat groups and expeditious use of the group data.

DirectPlay8Peer.DestroyPeer

#Deletes a peer from the session.

DestroyPeer(*idClient* **As Long, _**
lFlags **As Long, _**
UserData **As Any, _**
UserDataSize **As Long)**

Parts

idClient
Long value that specifies the identifier of the peer to delete.

lFlags
Reserved. Must be 0.

UserData
Pointer that describes additional delete data information that is sent to the peer.

UserDataSize
Long value that specifies the size of the data contained in the *UserData* parameter.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOTHOST

DPNERR_INVALIDPLAYER

DPNERR_INVALIDPARAM

Remarks

This method calls the **DirectPlay8Event.DestroyPlayer** method in the destroyed player's message handler.

DirectPlay8Peer.DestroyGroup

#Deletes a group created by the **DirectPlay8Peer.CreateGroup** method. This method can be called by any peer in the group.

IDH_DirectPlay8Peer.DestroyPeer_dplay_vb
IDH_DirectPlay8Peer.DestroyGroup_dplay_vb

This method will call the **DirectPlay8Event.AddRemovePlayerGroup** method for each peer in the group.

DestroyGroup(*idGroup* **As Long,** *lFlags* **As Long) As Long**

Parts

idGroup
Long value that specifies of the identifier of the group to delete.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDGROUP

DirectPlay8Peer.EnumHosts

#Enumerates applications that host Microsoft® DirectPlay® games. When an application is found meeting the enumeration criteria, the **DirectPlay8Event.EnumHostsResponse** method is called in the application's message handler. This method contains a **DPNMSG_ENUM_HOSTS_RESPONSE** message type that contains a **DPN_APPLICATION_DESC** type that describes the applications found.

EnumHosts(*ApplicationDesc* **As DPN_APPLICATION_DESC, _**
AddrHost **As DirectPlay8Address, _**
DeviceInfo **As DirectPlay8Address, _**
lRetryCount **As Long, _**
lRetryInterval **As Long, _**
lTimeOut **As Long, _**
lFlags **As Long, _**

IDH_DirectPlay8Peer.EnumHosts_dplay_vb

lFlags **As CONST_DPNOPERATIONS, _**
UserData **As Any, _**
UserDataSize **As Long) As Long**

Parts

ApplicationDesc
DPN_APPLICATION_DESC structure that specifies which application hosts to enumerate. You can specify the following fields to reduce the number of responses to the enumeration.

guidApplication
GUID of the application to find; if not specified, all are searched for.

Password
Password to provide; secure sessions will not respond without a password.

AddrHost
DirectPlay8Address object that specifies the address of the computer that is hosting the application.

DeviceInfo
DirectPlay8Address object that specifies the service provider and settings to enumerate.

lRetryCount
Value that specifies how many times the enumeration data will be sent. You can set this parameter to zero to specify the default value. If you set this value to INFINITE, the enumeration will continue until canceled.

lRetryInterval
Value that specifies the time, in milliseconds, between successive enumeration attempts. Set this parameter to 0 to use the default value.

lTimeOut
Number of milliseconds for the enumeration to run. If 0 is specified, a default value is used. If INFINITE is specified, the enumeration continues until it is canceled.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

UserData
Block of data that is sent in the enumeration request to the host. The size of the data can be limited depending on the network type, but 512 bytes is supported at a minimum.

UserDataSize
Size of the data in the *UserData* parameter.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_TIMEDOUT
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

Because of the variety of ways enumeration can happen, it is not recommended that an application specify *lRetryInterval*, *lRetryCount , orlTimeOut* unless the application has some specific media knowledge.

DirectPlay8Peer.GetApplicationDesc

[#]Retrieves the full application description for the connected application.

GetApplicationDesc(*lFlags* **As Long) _**
As DPN_APPLICATION_DESC

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns a **DPN_APPLICATION_DESC** type describing the application.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDFLAGS
DPNERR_BUFFERTOOSMALL

IDH_DirectPlay8Peer.GetApplicationDesc_dplay_vb

DPNERR_INVALIDPARAM

DirectPlay8Peer.GetCaps

#Retrieves the **DPN_CAPS** type for the current object.

GetCaps() As DPN_CAPS

Return Values

Returns a **DPN_SP_CAPS** type to receive the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Peer.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Peer.GetPlayerOrGroup

#Retrieves the client or group identifier for the specified client or group. The number of clients or groups for the application can be determined by calling **DirectPlay8Peer.GetCountClientsAndGroups**.

GetPlayerOrGroup(*lIndex* **As Long) As Long**

Parts

lIndex
Index of the client or group.

Return Values

Returns the identification of the client or group.

IDH_DirectPlay8Peer.GetCaps_dplay_vb
IDH_DirectPlay8Peer.GetPlayerOrGroup_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM

DirectPlay8Peer.GetConnectionInfo

#Retrieves statistical information about the connection between the local endpoint and the host.

```
GetConnectionInfo() As DPN_CONNECTION_INFO
```

Return Values

Returns a **DPN_CONNECTION_INFO** type to retrieve information about the specified connection.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

This method can be called only after a successful **Connect** call has completed.

DirectPlay8Peer.GetCountClientsAndGroups

#Retrieves the number of peers or groups for the session.

```
GetCountClientsAndGroups( _
   lFlags As CONST_DPNENUMCLIENTGROUPFLAGS) As Long
```

IDH_DirectPlay8Peer.GetConnectionInfo_dplay_vb
IDH_DirectPlay8Peer.GetCountClientsAndGroups_dplay_vb

Parts

lFlags
Flag that describes enumeration behavior. You can set one of the following flags of the **CONST_DPNENUMCLIENTGROUPFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

DirectPlay8Peer.GetCountGroupMembers

[#]Retrieves the number of players in a group.

GetCountGroupMembers(*dpid* **As Long) As Long**

Parts

dpid
Long value that specifies the identification of the group.

Return Values

Returns the number of players in the group.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

Remarks

Because player information changes frequently, the required buffer size returned may change between subsequent calls. Check and reallocate the buffer until the method succeeds.

IDH_DirectPlay8Peer.GetCountGroupMembers_dplay_vb

DirectPlay8Peer.GetCountServiceProviders

#Retrieves the number of registered service providers available to the application.

```
GetCountServiceProviders() As Long
```

Return Values

Returns the number of registered service providers available to the application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

DirectPlay8Peer.GetEnumHostResponseAddress

#Retrieves the address of the host.

```
GetEnumHostResponseAddress( _
   AppDesc As DPN_APPLICATION_DESC) _
   As DirectPlay8Address
```

Parts

AppDesc
DPN_APPLICATION_DESC type describing the application.

Return Values

Returns a **DirectPlay8Address** object specifying the address of the host.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

IDH_DirectPlay8Peer.GetCountServiceProviders_dplay_vb
IDH_DirectPlay8Peer.GetEnumHostResponseAddress_dplay_vb

DirectPlay8Peer.GetGroupInfo

#Retrieves a block of data associated with a group, including the group name.

This method is typically called after DirectPlay calls the **DirectPlay8Event.InfoNotify** method, indicating that the group data has been modified.

GetGroupInfo(*idGroup* **As Long) As DPN_GROUP_INFO**

Parts

idGroup
Long value that specifies the identifier of the group whose data block will be retrieved.

Return Values

Returns a **DPN_GROUP_INFO** type that describes the group data.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

DirectPlay8Peer.GetGroupMember

#Retrieves the identifier of the specified member of a group. The number of players in a group is determined by calling **DirectPlay8Peer.GetCountGroupMembers**.

GetGroupMember(*lIndex* **As Long,** *dpid* **As Long) As Long**

Parts

lIndex
Index value of the player in a group.
dpid
Long value that specifies the identification of the group.

Return Values

Returns a **Long** value that specifies the identification of the player.

IDH_DirectPlay8Peer.GetGroupInfo_dplay_vb
IDH_DirectPlay8Peer.GetGroupMember_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDPARAM

DirectPlay8Peer.GetPeerAddress

#Retrieves the address for the specified player in the session.

GetPeerAddress(*idPlayer* **As Long) As DirectPlay8Address**

Parts

idPlayer
Long value specifying the identification of the player.

Return Values

Returns a **DirectPlay8Address** object that specifies the address of the player.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_UNINITIALIZED

DirectPlay8Peer.GetPeerInfo

#Retrieves player information set for the specified peer.

GetPeerInfo(*idPeer* **As Long) As DPN_PLAYER_INFO**

Parts

idPeer
Long value that specifies the identifier of the peer whose information will be retrieved.

Return Values

Returns a **DPN_PLAYER_INFO** type describing player information.

IDH_DirectPlay8Peer.GetPeerAddress_dplay_vb
IDH_DirectPlay8Peer.GetPeerInfo_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDPLAYER
DPNERR_INVALIDPARAM

Remarks

Call this method after DirectPlay calls the **DirectPlay8Event.InfoNotify** method, indicating that the group data has been modified.

DirectPlay8Peer.GetSendQueueInfo

#Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® does not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should decrease the rate that messages are sent.

GetSendQueueInfo(*lNumMsgs* **As Long,** *lNumBytes* **As Long)**

Parts

lNumMsgs
Number of messages currently queued.

lNumBytes
Amount of data, in bytes, of the messages currently queued.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Peer.GetServiceProvider

#Retrieves information for the specified service provider. Before calling this method, call the **DirectPlay8Peer.GetCountServiceProviders** method to obtain the number of registered service providers available to the application.

GetServiceProvider(*lIndex* **As Long) As DPN_SERVICE_PROVIDER_INFO**

IDH_DirectPlay8Peer.GetSendQueueInfo_dplay_vb
IDH_DirectPlay8Peer.GetServiceProvider_dplay_vb

Parts

lIndex
Index value specifying the specific server provider.

Return Values

Returns a **DPN_SERVICE_PROVIDER_INFO** type that describes the service provider.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Peer.GetSPCaps

#Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

GetSPCaps(*guidSP* **As String) As DPN_SP_CAPS**

Parts

guidSP
String specifying the GUID of the service provider you want to get information about.

Return Values

Returns a **DPN_SP_CAPS** type to receive the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

IDH_DirectPlay8Peer.GetSPCaps_dplay_vb

Remarks

A successful call to **DirectPlay8Peer.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Peer.Host

#Specifies the host for the peer-to-peer session.

Host(*AppDesc* **As DPN_APPLICATION_DESC, _**
Address **As DirectPlay8Address)**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application.

Address
DirectPlay8Address object that specifies an addresses of a service-provider device or a service provider to host the application on.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Peer.RegisterLobby

#Registers or unregisters an application with a lobby.

RegisterLobby(*dpnHandle* **As Long, _**
LobbyApp **As DirectPlay8LobbiedApplication, _**
lFlags **As Long)**

Parts

dpnHandle
Connection handle to be used when making the calls to **DirectPlay8LobbiedApplication.UpdateStatus**.

LobbyApp
DirectPlay8LobbiedApplication object that specifies the application.

lFlags
Set this parameter to one of the two CONST_DPNLOBBY enumeration values to indicate whether the application is to be registered or unregistered.

IDH_DirectPlay8Peer.Host_dplay_vb
IDH_DirectPlay8Peer.RegisterLobby_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

Remarks

When an application is registered with a lobby, the lobby client automatically receives notifications of changes in game status.

DirectPlay8Peer.RegisterMessageHandler

#Registers an entry point in the client's code that receives the messages from the **DirectPlay8Peer** class. Call this method before calling any other methods of this class.

RegisterMessageHandler(*event* **As DirectPlay8Event)**

Parts

event
DirectPlay8Event object that will receive the messages generated by the client and server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_ALREADYREGISTERED

Remarks

If you want to register a new message handler, you must first call **DirectPlay8Peer.UnRegisterMessageHandler** to unregister the current message handler.

DirectPlay8Peer.RemovePlayerFromGroup

#Removes a peer from a group.

IDH_DirectPlay8Peer.RegisterMessageHandler_dplay_vb
IDH_DirectPlay8Peer.RemovePlayerFromGroup_dplay_vb

When this method is called, DirectPlay calls each player's **DirectPlay8Event.AddRemovePlayerGroup** method to notify him or her that a member has been removed from the group.

RemovePlayerFromGroup(*idGroup* **As Long, _**
idClient **As Long, _**
lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group that the peer will be removed from.

idClient
Long value that specifies the identifier of the peer that will be removed from the group.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPLAYER

DirectPlay8Peer.SendTo

#Transmits data to the specified player. The message can be sent synchronously or asynchronously.

SendTo(*idSend* **As Long, _**
buffer() **As BYTE, _**
lPriority **As Long, _**
lTimeOut **As Long, _**

IDH_DirectPlay8Peer.SendTo_dplay_vb

lFlags **As CONST_DPNSENDFLAGS) As Long**

Parts

idSend
Long value specifying the identifier of the player to receive data. Set this parameter to DPNID_ALL_PLAYERS_GROUP value from the CONST_DPNPLAYERGROUPFLAGS enumeration to send a message to all players in the session.

buffer()
Array of type **BYTE** that describes the data to send.

lPriority
Priority of the message.

lTimeOut
Number of milliseconds to wait for the message to be sent. If the message has not been sent by the *lTimeOut* value, the message is not sent. If you do not want a time-out for message sends, set this parameter to 0.

lFlags
Flags that describe send behavior. You can set one or more of the following flags defined in the **CONST_DPNSENDFLAGS** enumeration.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS

DPNERR_TIMEDOUT

Remarks

This method will call the **DirectPlay8Event.Receive** method in the recipient's message handler. When the **SendTo** request is completed, the **DirectPlay8Event.SendComplete** method is called in the sender's message handler. The **SendComplete** method contains a **DPNMSG_SEND_COMPLETE** message type. The success or failure of the request is contained in the **hResultCode** member of this message type.

DirectPlay8Peer.SetApplicationDesc

#Changes the settings for the application that is being hosted. Only some settings can be changed.

SetApplicationDesc(*AppDesc* **As DPN_APPLICATION_DESC, _**
***lFlags* As Long)**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application settings to modify.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

Remarks

You can use this method to modify only the following members of the **DPN_APPLICATION_DESC** type.

- **lMaxPlayers**
- **SessionName**
- **Password**

DirectPlay8Peer.SetCaps

#Sets the capabilities for the session.

SetCaps(*caps* **As DPN_CAPS,** *lFlags* **As Long)**

Parts

caps
DPN_CAPS type used to set the information about the current session.

lFlags
Reserved. Must be 0.

IDH_DirectPlay8Peer.SetApplicationDesc_dplay_vb
IDH_DirectPlay8Peer.SetCaps_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Peer.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Peer.SetGroupInfo

#Sets a block of data associated with a group, including the name of the group.

DirectPlay calls each player's **DirectPlay8Event.InfoNotify** method to notify him or her that the group information has changed.

SetGroupInfo(*idGroup* **As Long, _**
PlayerInfo **As DPN_GROUP_INFO, _**
lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group whose data block will be modified.

PlayerInfo
DPN_GROUP_INFO type that describes the group data to set.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method:

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

IDH_DirectPlay8Peer.SetGroupInfo_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

DirectPlay8Peer.SetPeerInfo

#Sets the static settings of a peer with an application. Call this method before connecting to relay basic player information with the application. After the peer successfully connects with the application, information obtained through this method can be retrieved by other players by calling the **DirectPlay8Peer.GetPeerInfo** method.

SetPeerInfo(*PlayerInfo* **As DPN_PLAYER_INFO) As Long**

Parts

PlayerInfo
DPN_PLAYER_INFO type that contains the player information to set.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in the *lAsyncHandle* parameter of the **DirectPlay8Peer.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

Transmission of nonstatic information should be handled with the **DirectPlay8Peer.SendTo** method because of the high cost of using the **DirectPlay8Peer.SetPeerInfo** method.

You can modify the peer information with this method after connecting to the application. If this method is called after a connection, DirectPlay will call each player's **DirectPlay8Event.InfoNotify** method to notify him or her that the data has been updated.

IDH_DirectPlay8Peer.SetPeerInfo_dplay_vb

DirectPlay8Peer.SetSPCaps

#Sets the capabilities for the specified service provider.

SetSPCaps(*guidSP* **As String,** *spCaps* **As DPN_SP_CAPS)**

Parts

guidSP
String specifying the GUID of the service provider you want to set information about.

spCaps
DPN_SP_CAPS type to set the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Peer.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Peer.TerminateSession

#Terminates the current Microsoft® DirectPlay® session.

TerminateSession(*lFlags* **As Long, _**
UserData **As Any, _**
UserDataSize **As Long)**

Parts

lFlags
Reserved. Must be 0.

UserData
Pointer to termination data.

UserDataSize
Size of data contained in the *UserData* parameter.

IDH_DirectPlay8Peer.SetSPCaps_dplay_vb
IDH_DirectPlay8Peer.TerminateSession_dplay_vb

Remarks

This method may be called only by the host player.

When this method is called, the **DirectPlay8Event.TerminateSession** method is called in the message handler of all players connected to the session.

DirectPlay8Peer.UnregisterMessageHandler

Unregisters the current message handler.

```
UnRegisterMessageHandler()
```

Remarks

You must call this method before registering a new message handler.

DirectPlay8Server

[#]Applications use the methods of the **DirectPlay8Server** class to create and manage the server for a Microsoft® DirectPlay® client/server transport session

The methods of the **DirectPlay8Server** class can be organized into the following groups.

Session Management	**Close**
	GetApplicationDesc
	GetCaps
	GetConnectionInfo
	GetCountServiceProviders
	GetServiceProvider
	GetSPCaps
	Host
	SetApplicationDesc
	SetCaps
	SetSPCaps
Message Management	**GetSendQueueInfo**
	RegisterMessageHandler
	SendTo
	UnregisterMessageHandler
Player Management	**DestroyClient**

IDH_DirectPlay8Server_dplay_vb

	GetClientAddress
	GetClientInfo
	GetClientOrGroup
	GetCountClientsAndGroups
	RemoveClientFromGroup
	SetServerInfo
Group Management	**AddClientToGroup**
	CreateGroup
	DestroyGroup
	GetCountGroupMembers
	GetGroupInfo
	GetGroupMember
	SetGroupInfo
Miscellaneous	**CancelAsyncOperation**
	RegisterLobby

DirectPlay8Server.AddClientToGroup

[#]Adds a client to a group.

AddClientToGroup(*idGroup* **As Long, _**
idClient **As Long,**
lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group to add the client to.

idClient
Long value that specifies the identifier of the client that is added to the group.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

IDH_DirectPlay8Server.AddClientToGroup_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

Remarks

Any client can add itself or another client to an existing group. After the client is successfully added to the group, all messages sent to the group are also sent to the client.

For a client to add itself to the group, pass DPNID_ME in the *idClient* parameter.

DirectPlay8Server.CancelAsyncOperation

#Cancels asynchronous requests. Many methods of the **DirectPlay8Server** class run asynchronously by default. Depending on the situation, you might want to cancel requests before they are processed. All the methods of this class that can be run asynchronously return a *lAsyncHandle* parameter.

Specific requests are canceled by passing the *lAsyncHandle* of the request in this method's *lAsyncHandle* parameter. You can cancel all pending asynchronous operations by calling this method, passing 0 in the *lAsyncHandle* parameter, and specifying DPNCANCEL_ALL_OPERATIONS in the *lFlags* parameter. If a specific handle is provided to this method, you must pass 0 in the *lFlags* parameter.

CancelAsyncOperation(*lAsyncHandle* **As Long, _**
lFlags **As CONST_DPNCANCELFLAGS)**

Parts

lAsyncHandle
: Handle of the asynchronous operation to stop. This value can be 0 to stop all requests or a particular type of asynchronous request. If a specific handle for the request to cancel is specified, the *lFlags* parameter must be 0.

lFlags
: Flag that specifies which asynchronous request to canceled. You can set this parameter to one of the flags of the **CONST_DPNCANCELFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

IDH_DirectPlay8Server.CancelAsyncOperation_dplay_vb

DPNERR_PENDING
DPNERR_INVALIDFLAGS
DPNERR_CANNOTCANCEL
DPNERR_INVALIDHANDLE

Remarks

You can cancel a send by providing the handle returned from **DirectPlay8Server.SendTo** method.. The **DirectPlay8Event.SendComplete** method will still be called unless the message was sent with the DPNSEND_NOCOMPLETE flag set. If you cancel a send operation by calling **DirectPlay8Peer.CancelAsyncOperation** the **hResultCode** member of the **DPNMSG_SEND_COMPLETE** type that is passed to the **DirectPlay8Event.SendComplete** method will be set to DPNERR_CANCELLED.

DirectPlay8Server.Close

#Closes the open connection with a session.

Close();

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_UNINITIALIZED

DirectPlay8Server.CreateGroup

#Creates a group in the current session. A group is a logical collection of players.

CreateGroup(*GroupInfo* **As DPN_GROUP_INFO, _**
***lFlags* As Long) As Long**

Parts

GroupInfo
DPN_GROUP_INFO structure that contains the group description.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

IDH_DirectPlay8Server.Close_dplay_vb
IDH_DirectPlay8Server.CreateGroup_dplay_vb

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

Remarks

Microsoft® DirectPlay® does not maintain hierarchical groups because they can easily be implemented with flat groups and expeditious use of the group data.

DirectPlay8Server.DestroyClient

[#]Deletes a client from the session.

DestroyClient(*idClient* **As Long, _**
lFlags **As Long, _**
UserData **As Any, _**
UserDataSize **As Long)**

Parts

idClient
Long value that specifies the identifier of the client to delete.

lFlags
Reserved. Must be 0.

UserData
Application-specific user data.

UserDataSize
Long value that specifies the size of the data contained in the *UserData* parameter.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOTHOST

DPNERR_INVALIDPLAYER

DPNERR_INVALIDPARAM

IDH_DirectPlay8Server.DestroyClient_dplay_vb

Remarks

This method calls the **DirectPlay8Event.DestroyPlayer** method in the destroyed player's message handler.

DirectPlay8Server.DestroyGroup

#Deletes a group created by the **DirectPlay8Server.CreateGroup** method.

DestroyGroup(*idGroup* **As Long,** *lFlags* **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group to delete.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in the *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS
DPNERR_INVALIDGROUP

DirectPlay8Server.GetApplicationDesc

#Retrieves the full application description for the connected application.

GetApplicationDesc(*lFlags* **As Long) _**
As DPN_APPLICATION_DESC

IDH_DirectPlay8Server.DestroyGroup_dplay_vb
IDH_DirectPlay8Server.GetApplicationDesc_dplay_vb

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns a **DPN_APPLICATION_DESC** type describing the application.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDFLAGS
DPNERR_BUFFERTOOSMALL
DPNERR_INVALIDPARAM

DirectPlay8Server.GetCaps

[#]Retrieves the **DPN_CAPS** type for the current object.

GetCaps() As DPN_CAPS

Return Values

Returns a **DPN_CAPS** type filled with caps information.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Server.RegisterMessageHandler** must be made before this method can be called.

IDH_DirectPlay8Server.GetCaps_dplay_vb

DirectPlay8Server.GetClientAddress

#Retrieves the address for the specified player in the session.

GetClientAddress(*idPlayer* **As Long) As DirectPlay8Address**

Parts

idPlayer
Long value specifying the identification of the player.

Return Values

Returns a **DirectPlay8Address** object that specifies the address of the player.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_UNINITIALIZED

DirectPlay8Server.GetClientInfo

#Retrieves player information set for the specified client.

GetClientInfo(*idClient* **As Long) As DPN_PLAYER_INFO**

Parts

idClient
Long value that specifies the identifier of the client whose information will be retrieved.

Return Values

Returns a **DPN_PLAYER_INFO** type describing player information.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDPLAYER
DPNERR_INVALIDPARAM

IDH_DirectPlay8Server.GetClientAddress_dplay_vb
IDH_DirectPlay8Server.GetClientInfo_dplay_vb

Remarks

Call this method after DirectPlay calls the **DirectPlay8Event.InfoNotify** method, indicating that the player data has been modified.

DirectPlay8Server.GetClientOrGroup

#Retrieves the client or group identifier for the specified client or group. The number of clients or groups for the application can be determined by calling **DirectPlay8Server.GetCountClientsAndGroups**.

GetClientOrGroup(*lIndex* **As Long) As Long**

Parts

lIndex
Index of the client or group.

Return Values

Returns the identification of the client or group.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPARAM

DirectPlay8Server.GetConnectionInfo

#Retrieves statistical information about the connection between the specified player and the server.

GetConnectionInfo(*idPlayer* **As Long) As DPN_CONNECTION_INFO**

Parameter

idPlayer
Long value specifying the identifier of the player.

IDH_DirectPlay8Server.GetClientOrGroup_dplay_vb
IDH_DirectPlay8Server.GetConnectionInfo_dplay_vb

Return Values

Returns a **DPN_CONNECTION_INFO** type to retrieve information about the specified connection.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

This method can be called only after a successful **Connect** call has completed.

DirectPlay8Server.GetCountClientsAndGroups

#Retrieves the number of clients or groups for the session.

```
GetCountClientsAndGroups( _
    lFlags As CONST_DPNENUMCLIENTGROUPFLAGS) As Long
```

Parts

lFlags
Flag that describes enumeration behavior. You can set one of the flags from the **CONST_DPNENUMCLIENTGROUPFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

DirectPlay8Server.GetCountGroupMembers

#Retrieves the number of players in a group.

IDH_DirectPlay8Server.GetCountClientsAndGroups_dplay_vb
IDH_DirectPlay8Server.GetCountGroupMembers_dplay_vb

GetCountGroupMembers(*dpid* As Long) As Long

Parts

dpid
Long value that specifies the identification of the group.

Return Values

Returns the number of players in the group.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

Remarks

Because player information changes frequently, the required buffer size returned may change between subsequent calls. Check and reallocate the buffer until the method succeeds.

DirectPlay8Server.GetCountServiceProviders

[#]Retrieves the number of registered service providers available to the application.

GetCountServiceProviders() As Long

Return Values

Returns the number of registered service providers available to the application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDOBJECT

IDH_DirectPlay8Server.GetCountServiceProviders_dplay_vb

DirectPlay8Server.GetGroupInfo

#Retrieves a block of data associated with a group, including the group name.

Call this method after DirectPlay calls the **DirectPlay8Event.InfoNotify** method, indicating that the group data has been modified.

GetGroupInfo(*idGroup* **As Long) As DPN_GROUP_INFO**

Parts

idGroup
Long value that specifies the identifier of the group whose data block will be retrieved.

Return Values

Returns a **DPN_GROUP_INFO** type that describes the group data.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDFLAGS

DirectPlay8Server.GetGroupMember

#Retrieves the identifier of the specified member of a group. The number of players in a group is determined by calling **DirectPlay8Server.GetCountGroupMembers**.

GetGroupMember(*lIndex* **As Long,** *dpid* **As Long) As Long**

Parts

lIndex
Index value of the player in a group.
dpid
Long value that specifies the identification of the group.

Return Values

Long value that specifies the identification of the player.

IDH_DirectPlay8Server.GetGroupInfo_dplay_vb
IDH_DirectPlay8Server.GetGroupMember_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP
DPNERR_INVALIDPARAM

DirectPlay8Server.GetSendQueueInfo

#Used by the application to monitor the size of the send queue. Microsoft® DirectPlay® does not send messages faster than the receiving computer can process them. As a result, if the sending computer is sending faster than the receiver can receive, messages accumulate in the sender's queue. If the application registers that the send queue is growing too large, it should decrease the rate at which messages are sent.

GetSendQueueInfo(*idPlayer* **As Long, _**
lNumMsgs **As Long, _**
lNumBytes **As Long)**

Parts

idPlayer
Long value specifying the identifier of the player.

lNumMsgs
Number of messages currently queued.

lNumBytes
Amount of data, in bytes, of the messages currently queued.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Server.GetServiceProvider

#Retrieves information for the specified service provider. Before you call this method, call the **DirectPlay8Server.GetCountServiceProviders** method to obtain the number of registered service providers available to the application.

GetServiceProvider(*lIndex* **As Long) As DPN_SERVICE_PROVIDER_INFO**

IDH_DirectPlay8Server.GetSendQueueInfo_dplay_vb
IDH_DirectPlay8Server.GetServiceProvider_dplay_vb

Parts

lIndex
Index value specifying the specific server provider.

Return Values

Returns a **DPN_SERVICE_PROVIDER_INFO** type that describes the service provider.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Server.GetSPCaps

[#]Retrieves the **DPN_SP_CAPS** structure for the specified service provider.

GetSPCaps(*guidSP* **As String) As DPN_SP_CAPS**

Parts

guidSP
String specifying the GUID of the service provider you want to get information about.

Return Values

Returns a **DPN_SP_CAPS** type to receive the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

IDH_DirectPlay8Server.GetSPCaps_dplay_vb

Remarks

A successful call to **DirectPlay8Server.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Server.Host

#Specifies the host for the client/server session.

Host(*AppDesc* **As DPN_APPLICATION_DESC, _**
Address **As DirectPlay8Address)**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application.

Address
DirectPlay8Address object that specifies the address of a service-provider device or a service provider on which to host the application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

DirectPlay8Server.RegisterLobby

#Enables launched applications to automatically propagate game status to the lobby.

RegisterLobby(*LobbyApp* **As DirectPlay8LobbiedApplication)**

Parts

LobbyApp
DirectPlay8LobbiedApplication object that specifies the application.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDPARAM

IDH_DirectPlay8Server.Host_dplay_vb
IDH_DirectPlay8Server.RegisterLobby_dplay_vb

DirectPlay8Server.RegisterMessageHandler

#Registers an entry point in the server's code that receives the messages from the **DirectPlay8Server** class and from connected clients.

RegisterMessageHandler(*event* **As DirectPlay8Event)**

Parts

event
DirectPlay8Event object that will receive the messages generated by the client and server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_ALREADYREGISTERED

Remarks

If you want to register a new message handler, you must first call **DirectPlay8Server.UnRegisterMessageHandler** to unregister the current message handler.

DirectPlay8Server.RemoveClientFromGroup

#Removes a client from a group.

RemoveClientFromGroup(*idGroup* **As Long, _**
idClient **As Long, _**
lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group from which the client will be removed.

idClient
Long value that specifies the identifier of the client that will be removed from the group.

lFlags

IDH_DirectPlay8Server.RegisterMessageHandler_dplay_vb
IDH_DirectPlay8Server.RemoveClientFromGroup_dplay_vb

Flag that controls how this method is processed. The following flag can be set for this method.

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP

DPNERR_INVALIDFLAGS

DPNERR_INVALIDPLAYER

DirectPlay8Server.SendTo

[#]Transmits data to the specified player. The message can be sent synchronously or asynchronously.

SendTo(*idSend* **As Long, _**
buffer() **As BYTE, _**
lPriority **As Long, _**
lTimeOut **As Long, _**
lFlags **As CONST_DPNSENDFLAGS) As Long**

Parts

idSend
Long value specifying the identifier of the player to receive data.

buffer()
Array of type **BYTE** that describes the data to send.

lPriority
Priority of the message.

lTimeOut
Number of milliseconds to wait for the message to be sent. If the message has not been sent by the *lTimeOut* value, the message is not sent. If you do not want a time-out for message sends, set this parameter to 0.

lFlags

IDH_DirectPlay8Server.SendTo_dplay_vb

Flags that describe send behavior. You can set one or more of the flags defined in the **CONST_DPNSENDFLAGS** enumeration.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDFLAGS

DPNERR_TIMEDOUT

Remarks

This method will call the **DirectPlay8Event.Receive** method in the recipient's message handler. When the **SendTo** request is completed, the **DirectPlay8Event.SendComplete** method is called in the sender's message handler. The **SendComplete** method contains a **DPNMSG_SEND_COMPLETE** message type. The success or failure of the request is contained in the **hResultCode** member of this message type.

DirectPlay8Server.SetApplicationDesc

[#]Changes the settings for the application that is being hosted. Only some settings can be changed.

SetApplicationDesc(*AppDesc* **As DPN_APPLICATION_DESC, _**
lFlags **As Long)**

Parts

AppDesc
DPN_APPLICATION_DESC type that describes the application settings to modify.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DPNERR_INVALIDFLAGS

IDH_DirectPlay8Server.SetApplicationDesc_dplay_vb

Remarks

You can use this method to modify only the following members of the **DPN_APPLICATION_DESC** type:

- **lMaxPlayers**
- **SessionName**
- **Password**

DirectPlay8Server.SetCaps

#Sets the capabilities for the session.

SetCaps(*caps* **As DPN_CAPS,** *lFlags* **As Long)**

Parts

caps
DPN_CAPS type used to set the information about the current session.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM
DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Server.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Server.SetGroupInfo

#Sets a block of data associated with a group, including the name of the group.

SetGroupInfo(*idGroup* **As Long, _**
PlayerInfo **As DPN_GROUP_INFO, _**

IDH_DirectPlay8Server.SetCaps_dplay_vb
IDH_DirectPlay8Server.SetGroupInfo_dplay_vb

lFlags **As Long) As Long**

Parts

idGroup
Long value that specifies the identifier of the group whose data block will be modified.

PlayerInfo
DPN_GROUP_INFO type that describes the group data to set.

lFlags
Flag that controls how this method is processed. The following flag can be set for this method:

DPNOP_SYNC
Causes the method to process synchronously.

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDGROUP

DPNERR_INVALIDFLAGS

DirectPlay8Server.SetServerInfo

#Sets the static settings of a client with an application. Call this method before connecting to relay basic player information with the application. After the client successfully connects with the application, other players can retrieve information obtained through this method by calling the **DirectPlay8Client.GetServerInfo** method.

SetServerInfo(*PlayerInfo* **As DPN_PLAYER_INFO) As Long**

Parts

PlayerInfo
DPN_PLAYER_INFO type that contains the player information to set.

IDH_DirectPlay8Server.SetServerInfo_dplay_vb

Return Values

Returns the asynchronous handle for this operation. This is the handle that is used in *lAsyncHandle* parameter of the **DirectPlay8Server.CancelAsyncOperation** method to cancel the request, if the request is processed asynchronously.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_NOCONNECTION
DPNERR_INVALIDFLAGS
DPNERR_INVALIDPARAM

Remarks

Transmission of nonstatic information should be handled with the **DirectPlay8Server.SendTo** method because of the high cost of using the **DirectPlay8Server.SetClientInfo** method.

You can modify the client information with this method after connecting to the application. If this method is called after a connection, DirectPlay will call each player's **DirectPlay8Event.InfoNotify** method to notify him or her that the data has been updated.

DirectPlay8Server.SetSPCaps

[#]Sets the capabilities for the specified service provider.

SetSPCaps(*guidSP* **As String,** *spCaps* **As DPN_SP_CAPS)**

Parts

guidSP
: **String** specifying the GUID of the service provider you want to set information about.

spCaps
: **DPN_SP_CAPS** type to set the information about the specified service provider.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DPNERR_INVALIDOBJECT
DPNERR_INVALIDPOINTER
DPNERR_INVALIDPARAM

IDH_DirectPlay8Server.SetSPCaps_dplay_vb

DPNERR_UNINITIALIZED

Remarks

A successful call to **DirectPlay8Server.RegisterMessageHandler** must be made before this method can be called.

DirectPlay8Server.UnregisterMessageHandler

Unregisters the current message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlayVoiceClient8

[#]Applications use the methods of the **DirectPlayVoiceClient8** class to manage clients in a voice session.

The methods of the **DirectPlayVoiceClient8** class can be organized into the following groups.

Buffer management	**Create3DSoundBuffer**
	Delete3DSoundBuffer
Miscellaneous	**GetCaps**
	GetCompressionType
	GetCompressionTypeCount
	GetSoundDeviceConfig
	GetSoundDevices
	StartClientNotification
Session management	**Connect**
	Disconnect
	GetClientConfig
	GetSessionDesc
	GetTransmitTargets
	Initialize
	SetClientConfig

IDH_DirectPlayVoiceClient8_dplay_vb

SetCurrentSoundDevices
SetTransmitTargets
UnRegisterMessageHandler

DirectPlayVoiceClient8.Connect

#Connects the client to a Microsoft® DirectPlay® Voice session.

Connect(_
SoundDeviceConfig **As DVSOUNDDEVICECONFIG, _**
ClientConfig **As DVCLIENTCONFIG, _**
lFlags **As Long)**

Parts

SoundDeviceConfig
DVSOUNDDEVICECONFIG type that describes the sound device configuration.

ClientConfig
DVCLIENTCONFIG type that describes the general configuration of the client.

lFlags
You can specify the following flag.

DVFLAGS_SYNC
The method does not return until the operation is completed.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_ALREADYPENDING
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_INVALIDOBJECT
DVERR_SOUNDINITFAILURE
DVERR_INVALIDFLAGS
DVERR_OUTOFMEMORY
DVERR_NOTINITIALIZED
DVERR_COMPRESSIONNOTSUPPORTED
DVERR_TIMEOUT
DVERR_INVALIDDEVICE
DVERR_SENDERROR

IDH_DirectPlayVoiceClient8.Connect_dplay_vb

DVERR_INCOMPATIBLEVERSION
DVERR_TRANSPORTNOPLAYER
DVERR_TRANSPORTNOSESSION
DVERR_RUNSETUP

Remarks

You must test the sound devices selected for playback and capture by invoking the Setup Wizard before connecting the client to the DirectPlay Voice session. On application startup, check the audio configuration by using **DirectPlayVoiceTest8.CheckAudioSetup**. If this method returns DVERR_RUNSETUP, the sound configuration specified has not been tested. The Setup Wizard needs to be run only once for any configuration.

Any calls to **DirectPlayVoiceClient8.Connect** while a connection is pending return DVERR_ALREADYPENDING. Additionally, only one connection can be pending at a time.

A transport session must be started on the specified DirectPlay object before calling this method. A successful call to **DirectPlayVoiceClient8.Initialize** must be made before calling the **Connect** method.

DirectPlayVoiceClient8.Create3DSoundBuffer

#Retrieves a 3-D sound buffer for a player or group. You can use the methods of the 3-D sound buffer object to change the virtual 3-D position of incoming voice transmissions from the specified group or player.

Create3DSoundBuffer(*playerID* **As Long) _**
** As DirectSound3dBuffer8**

Parts

playerID
: **Long** value that specifies the identification of the player or group for which the user wants to reserve a buffer. You can also specify DVID_REMAINING to create a 3-D user buffer for all players or groups that do not have a user buffer.

Return Values

Returns a **DirectSound3dBuffer8** type used for the Microsoft® DirectPlay® Voice main buffer.

IDH_DirectPlayVoiceClient8.Create3DSoundBuffer_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_NOTCONNECTED
DVERR_SESSIONLOST
DVERR_ALREADYBUFFERED
DVERR_INVALIDPOINTER
DVERR_INVALIDOBJECT
DVERR_NOTINITIALIZED
DVERR_OUTOFMEMORY
DVERR_NOTALLOWED

Remarks

If the DirectPlay Voice session is a mixing server session, this method fails and returns DVERR_NOTALLOWED.

Because the DirectPlay Voice client uses the buffer to stream incoming audio, you can access all the member functions of the 3-D sound buffer object. However, you should not use the **Lock**, **UnLock**, or **Play** methods of the **DirectSound3DBuffer** object.

When the buffer for the individual user is no longer required or when a player leaves the voice session, it is important to call **DirectPlayVoiceClient8.Delete3DSoundBuffer** to free up resources.

DirectPlayVoiceClient8.Delete3DSoundBuffer

[#]Returns exclusive control of the 3-D sound buffer object back to the Microsoft® DirectPlay® Voice client object.

Delete3DSoundBuffer(*playerID* **As Long, _**
UserBuffer **As DirectSound3dBuffer8)**

Parts

playerID
Long value specifying the DVID of the player or group that the user wants to delete a buffer for.

UserBuffer

IDH_DirectPlayVoiceClient8.Delete3DSoundBuffer_dplay_vb

User buffer to delete. This must be a user buffer obtained through the **DirectPlayVoiceClient8.Create3DSoundBuffer** method.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_NOTCONNECTED
DVERR_SESSIONLOST
DVERR_ALREADYBUFFERED
DVERR_INVALIDPOINTER
DVERR_NOTBUFFERED
DVERR_INVALIDOBJECT
DVERR_NOTINITIALIZED
DVERR_NOTALLOWED

Remarks

If the DirectPlay Voice session is a mixing server session, this method fails and returns DVERR_NOTALLOWED.

DirectPlayVoiceClient8.Disconnect

[#]Disconnects the Microsoft® DirectPlay® Voice client from the existing DirectPlay Voice session.

Disconnect(*lFlags* **As Long)**

Parts

lFlags
You can specify the following flag.
DVFLAGS_SYNC
Do not return until the operation is completed.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_PENDING

IDH_DirectPlayVoiceClient8.Disconnect_dplay_vb

DVERR_ALREADYPENDING
DVERR_INVALIDPARAM
DVERR_NOTCONNECTED
DVERR_INVALIDFLAGS
DVERR_CONNECTABORTING
DVERR_NOTINITIALIZED
DVERR_TIMEOUT
DVERR_SESSIONLOST

Remarks

On calling this method, all recording and playback is stopped. If a connection is being processed, it is canceled by this call.

If this method is called synchronously by setting the DVFLAGS_SYNC flag, the method does not return until the **Disconnect** completes.

DirectPlayVoiceClient8.GetCaps

#Retrieves the Microsoft® DirectPlay® Voice capabilities.

GetCaps() As DVCAPS

Return Values

Returns a **DVCAPS** type that contains the capabilities of the **DirectPlayVoiceClient8** object.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER

DirectPlayVoiceClient8.GetClientConfig

#Retrieves the client configuration.

GetClientConfig() As DVCLIENTCONFIG

IDH_DirectPlayVoiceClient8.GetCaps_dplay_vb
IDH_DirectPlayVoiceClient8.GetClientConfig_dplay_vb

Return Values

Returns a **DVCLIENTCONFIG** type that contains the configuration of the local client.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_NOTCONNECTED
DVERR_SESSIONLOST
DVERR_INVALIDPOINTER
DVERR_NOTINITIALIZED

Remarks

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

DirectPlayVoiceClient8.GetCompressionType

[#]Retrieves the compression type.

GetCompressionType(*lIndex* **As Long, _**
Data **As DVCOMPRESSIONINFO, _**
lFlags **As Long)**

Parts

lIndex
Specific compression type. The number of supported compression types can be determined with a call to **DirectPlayVoiceClient8.GetCompressionTypeCount**.

Data
DVCOMPRESSIONINFO type describing compression information.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

IDH_DirectPlayVoiceClient8.GetCompressionType_dplay_vb

DVERR_INVALIDPARAM
DVERR_BUFFERTOOSMALL
DVERR_INVALIDPOINTER
DVERR_INVALIDFLAGS

DirectPlayVoiceClient8.GetCompression TypeCount

#Retrieves the number of supported compression types.

GetCompressionTypeCount() As Long

Return Values

Returns the number of supported compression types.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DVERR_INVALIDOBJECT

DirectPlayVoiceClient8.GetSessionDesc

#Retrieves the session properties.

GetSessionDesc() As DVSESSIONDESC

Return Values

Returns a **DVSESSIONDESC** type to receive the session description.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_SESSIONLOST

IDH_DirectPlayVoiceClient8.GetCompressionTypeCount_dplay_vb
IDH_DirectPlayVoiceClient8.GetSessionDesc_dplay_vb

DVERR_NOTINITIALIZED

Remarks

This method may be called only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

DirectPlayVoiceClient8.GetSoundDeviceConfig

#Retrieves the sound device configuration of the session.

GetSoundDeviceConfig() As DVSOUNDDEVICECONFIG

Return Values

Returns a **DVSOUNDDEVICECONFIG** type that is filled with the configuration of the sound device.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_NOTCONNECTED
DVERR_SESSIONLOST
DVERR_INVALIDPOINTER
DVERR_NOTINITIALIZED

Remarks

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

DirectPlayVoiceClient8.GetSoundDevices

#Retrieves the current Microsoft® DirectSound® capture and playback objects.

GetSoundDevices(_
DirectSoundObj **As DirectSound8, _**
DirectCaptureObj **As DirectSoundCapture8)**

IDH_DirectPlayVoiceClient8.GetSoundDeviceConfig_dplay_vb
IDH_DirectPlayVoiceClient8.GetSoundDevices_dplay_vb

Parts

DirectSoundObj
Current **DirectSound8** object being used for playback.

DirectCaptureObj
Current **DirectSoundCapture8** object being using for capture.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DVERR_INVALIDPARAM

DirectPlayVoiceClient8.GetTransmitTargets

[#]Retrieves the transmit targets, if any, of the voice stream from this client.

GetTransmitTargets(*lFlags* **As Long) As Long()**

Parts

lFlags
Reserved. Must be 0.

Return Values

Returns an array of **Long** values containing the DVIDs of the of the transmit targets, if any, of the client's voice stream.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_NOTALLOWED
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_BUFfERTOOSMALL
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_INVALIDFLAGS

IDH_DirectPlayVoiceClient8.GetTransmitTargets_dplay_vb

Remarks

The DVIDs returned can be player or group DVIDs.

DirectPlayVoiceClient8.Initialize

#Initializes the **DirectPlayVoiceClient8** object by associating the **DirectPlayVoiceClient8** object with a DirectPlay object.

This method must be called successfully before **DirectPlayVoiceClient8.Connect** method is called.

Initialize(*DplayObj* **As Unknown,** *lFlags* **As Long)**

Parts

DplayObj
Pointer to the **IUnknown** interface for the DirectPlay object that this **DirectPlayVoiceClient8** object should use.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_ALREADYINITIALIZED
DVERR_INVALIDPOINTER
DVERR_TRANSPORTNOTINIT
DVERR_NOCALLBACK
DVERR_GENERIC

DirectPlayVoiceClient8.SetClientConfig

#Sets the client configuration.

SetClientConfig(*ClientConfig* **As DVCLIENTCONFIG)**

Parts

ClientConfig
DVCLIENTCONFIG type that contains the configuration description to set.

IDH_DirectPlayVoiceClient8.Initialize_dplay_vb
IDH_DirectPlayVoiceClient8.SetClientConfig_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_INVALIDFLAGS
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_SESSIONLOST

Remarks

You can call this method only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

Calling this method sets all the parameters in the **DVCLIENTCONFIG** type. Therefore, to leave a setting unmodified, you must retrieve the current configuration with **DirectPlayVoiceClient8.GetClientConfig**. Then, modify the parameters to change and call **DirectPlayVoiceClient8.SetClientConfig**.

If the session is running in half duplex, the members of the **GetClientConfig** method related to recording are ignored.

DirectPlayVoiceClient8.SetCurrentSoundDevices

[#]Sets the current Microsoft® DirectSound® capture and playback objects.

SetCurrentSoundDevices(_
DirectSoundObj **As DirectSound8, _**
DirectCaptureObj **As DirectSoundCapture8)**

Parts

DirectSoundObj
DirectSound8 object to be used for playback.

DirectCaptureObj
DirectSoundCapture8 object to be used for capture.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

IDH_DirectPlayVoiceClient8.SetCurrentSoundDevices_dplay_vb

DVERR_INVALIDPARAM

DirectPlayVoiceClient8.SetTransmitTargets

#Specifies which players or groups receive audio transmissions from the local client.

SetTransmitTargets(*playerIDs()* **As Long,** *lFlags* **As Long)**

Parts

playerIDs()
Array of **Long** values specifying the DVIDs of the players or groups that are to receive the voice transmission.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDFLAGS
DVERR_INVALIDPOINTER
DVERR_NOTINITIALIZED
DVERR_INVALIDTARGET

Remarks

If the session was created with the DVSESSION_SERVERCONTROLTARGET flag, only the server can set the target for this local client. A call to this method returns DVERR_NOTALLOWED.

DirectPlayVoiceClient8.StartClientNotification

#Registers a **DirectPlayVoiceEvent8** object with the client to receive all messages generated by the client and server.

StartClientNotification(*event* **As DirectPlayVoiceEvent8)**

IDH_DirectPlayVoiceClient8.SetTransmitTargets_dplay_vb
IDH_DirectPlayVoiceClient8.StartClientNotification_dplay_vb

Parts

event
DirectPlayVoiceEvent8 object that will receive messages from this client and from the server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DVERR_INVALIDPARAM

DirectPlayVoiceClient8.UnregisterMessageHandler

#Unregisters the current **DirectPlayVoiceClient8** message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlayVoiceEvent8

#Applications use the methods of the **DirectPlayVoiceEvent8** class to capture Microsoft® DirectPlay® events generated by the **DirectPlayVoiceClient8** and **DirectPlayVoiceServer8** classes.

Note
When implementing the **DirectPlayVoiceEvent8** object, Microsoft® Visual Basic® requires that every method of this object is implemented in your form.

The methods of the **DirectPlayVoiceEvent8** class are:

DirectPlayVoiceEvent8 Methods	**ConnectResult**
	CreateVoicePlayer
	DeleteVoicePlayer
	DisconnectResult
	HostMigrated
	InputLevel
	OutputLevel

IDH_DirectPlayVoiceClient8.UnregisterMessageHandler_dplay_vb
IDH_DirectPlayVoiceEvent8_dplay_vb

PlayerOutputLevel
PlayerVoiceStart
PlayerVoiceStop
RecordStart
RecordStop
SessionLost

DirectPlayVoiceEvent8.ConnectResult

#Called when the connect request generated through a call to the DirectPlayVoiceClient8.Connect method has completed.

ConnectResult(*ResultCode* **As Long)**

Parts

ResultCode
Long value set to the result code.

DirectPlayVoiceEvent8.CreateVoicePlayer

#Called when a new player joins the voice session.

CreateVoicePlayer(*playerID* **As Long, _**
flags **As Long)**

Parts

playerID
Long value set to the player's ID.

flags
Flags with information about the player.

DirectPlayVoiceEvent8.DeleteVoicePlayer

#Called when a player quits the voice session**.**

IDH_DirectPlayVoiceEvent8.ConnectResult_dplay_vb
IDH_DirectPlayVoiceEvent8.CreateVoicePlayer_dplay_vb
IDH_DirectPlayVoiceEvent8.DeleteVoicePlayer_dplay_vb

DeleteVoicePlayer(*playerID* **As Long)**

Parts

playerID
Long value set to the player's ID.

DirectPlayVoiceEvent8.DisconnectResult

#Called when the disconnect request generated through a call to the **DirectPlayVoiceClient8.Disconnect** method has completed.

DisconnectResult(*ResultCode* **As Long)**

Parts

ResultCode
Long value set to the result code.

DirectPlayVoiceEvent8.HostMigrated

#Called when the voice host has changed.

HostMigrated(*NewHostID* **As Long, _**
NewServer **As DirectPlayVoiceServer8)**

Parts

NewHostID
Long value set to the ID of the new voice host.

NewServer
If the local client has become the new voice session host, this member will point to a newly created **DirectPlayVoiceServer8** object that can be used by the local client for providing host services. If the local client is not the new host, then this member will be NULL.

DirectPlayVoiceEvent8.InputLevel

#Called periodically to notify the user of the input level from the microphone.

InputLevel(*PeakLevel* **As Long, _**

IDH_DirectPlayVoiceEvent8.DisconnectResult_dplay_vb
IDH_DirectPlayVoiceEvent8.HostMigrated_dplay_vb
IDH_DirectPlayVoiceEvent8.InputLevel_dplay_vb

RecordVolume **As Long)**

Parts

PeakLevel

Long value representing peak level across the current frame, which corresponds to approximately 1/10 second of audio stream. The current frame typically lags 50-200 milliseconds (ms) behind real-time. This value can range from 0 through 99, with 0 being completely silent and 99 being the highest possible input level.

RecordVolume

Current recording volume for the client. The value can range from -10,000 through 0. This member is available even when automatic gain control is active.

DirectPlayVoiceEvent8.OutputLevel

#Called periodically to notify the user of the output level from the microphone.

OutputLevel(*PeakLevel* **As Long,** _
OutputVolume **As Long)**

Parts

PeakLevel

Long value representing peak level across the current frame, that corresponds to approximately 1/10 second of audio stream. The current frame typically lags 50-200 ms behind real-time. This value can range from 0 through 99, with 0 being completely silent and 99 being the highest possible output level.

OutputVolume

Current playback volume for the client.

DirectPlayVoiceEvent8.PlayerOutputLevel

#Called periodically to notify the user of the output level of an individual player's voice stream. It is generated while voice is being played back for an individual player. If multiple player voices are being played, one message for each player speaking will be sent each notification period.

PlayerOutputLevel(*SourcePlayerID* **As Long,** _
PeakLevel **As Long)**

IDH_DirectPlayVoiceEvent8.OutputLevel_dplay_vb
IDH_DirectPlayVoiceEvent8.PlayerOutputLevel_dplay_vb

Parts

SourcePlayerID
Long value set to the ID of the player whose voice is being played back.

PeakLevel
Integer representing the peak output level of the player's voice stream. This value is in the range from 0 through 99, with 0 being completely silent and 99 being the highest possible output level

DirectPlayVoiceEvent8.PlayerVoiceStart

#Called when an incoming audio stream begins playing back.

PlayerVoiceStart(*SourcePlayerID* **As Long)**

Parts

SourcePlayerID
Long value set to the ID of the player where the voice transmission originated.

DirectPlayVoiceEvent8.PlayerVoiceStop

#Called when an incoming audio stream stops.

PlayerVoiceStop(*SourcePlayerID* **As Long)**

Parts

SourcePlayerID
Long value set to the ID of the player where the voice transmission stopped.

DirectPlayVoiceEvent8.RecordStart

#Called when audio input on the local client begins. This can be caused by the voice activation sensitivity level being exceeded or when a valid target is specified in push-to-talk mode.

RecordStart(*PeakVolume* **As Long)**

Parts

PeakVolume

IDH_DirectPlayVoiceEvent8.PlayerVoiceStart_dplay_vb
IDH_DirectPlayVoiceEvent8.PlayerVoiceStop_dplay_vb
IDH_DirectPlayVoiceEvent8.RecordStart_dplay_vb

Long value set to the voice activation level that caused the transmission to begin. In push-to-talk mode, this value is 0.

DirectPlayVoiceEvent8.RecordStop

#Called when audio input on the local client stops. This can be caused by the voice activation sensitivity level not being reached or when a target is deselected in push-to-talk mode.

RecordStop(*PeakVolume* **As Long)**

Parts

PeakVolume
Long value set to the voice activation level that caused the transmission to stop. In push-to-talk mode, this value is 0.

DirectPlayVoiceEvent8.SessionLost

#Called when the voice session terminates.

SessionLost(*ResultCode* **As Long)**

Parts

ResultCode
Long value set to a result code that indicates why the session terminated.

DirectPlayVoiceServer8

#Applications use the methods of the **DirectPlayVoiceServer8** class to manage the host of the voice session.

The methods of the **DirectPlayVoiceServer8** class can be organized into the following groups.

Miscellaneous	**GetCaps**
	GetCompressionType
	GetCompressionTypeCount
Session Management	**GetSessionDesc**

IDH_DirectPlayVoiceEvent8.RecordStop_dplay_vb
IDH_DirectPlayVoiceEvent8.SessionLost_dplay_vb
IDH_DirectPlayVoiceServer8_dplay_vb

GetTransmitTargets
Initialize
SetSessionDesc
SetTransmitTargets
StartServerNotification
StartSession
StopSession
UnregisterMessageHandler

DirectPlayVoiceServer8.GetCaps

#Retrieves the Microsoft® DirectPlay® Voice capabilities.

GetCaps() As DVCAPS

Return Values

Returns a **DVCAPS** type that contains the capabilities of the **DirectPlayVoiceClient8** object.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER

DirectPlayVoiceServer8.GetCompressionType

#Retrieves the compression type.

GetCompressionType(*lIndex* **As Long, _**
Data **As DVCOMPRESSIONINFO, _**
lFlags **As Long)**

Parts

lIndex

IDH_DirectPlayVoiceServer8.GetCaps_dplay_vb
IDH_DirectPlayVoiceServer8.GetCompressionType_dplay_vb

Specific compression type. The number of supported compression types can be determined with a call to **DirectPlayVoiceServer8.GetCompressionTypeCount**.

Data

DVCOMPRESSIONINFO type describing compression information.

lFlags

Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM

DVERR_BUFFERTOOSMALL

DVERR_INVALIDPOINTER

DVERR_INVALIDFLAGS

DirectPlayVoiceServer8.GetCompressionTypeCount

#Retrieves the number of supported compression types.

GetCompressionTypeCount() As Long

Return Values

Returns the number of supported compression types.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

DVERR_INVALIDOBJECT

DirectPlayVoiceServer8.GetSessionDesc

#Retrieves the session properties.

GetSessionDesc() As DVSESSIONDESC

IDH_DirectPlayVoiceServer8.GetCompressionTypeCount_dplay_vb
IDH_DirectPlayVoiceServer8.GetSessionDesc_dplay_vb

Return Values

Returns a **DVSESSIONDESC** type to receive the session description.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_NOTCONNECTED
DVERR_SESSIONLOST
DVERR_NOTINITIALIZED

Remarks

This method may be called only after a connection is successfully established with a Microsoft® DirectPlay® Voice session.

DirectPlayVoiceServer8.GetTransmitTargets

#Retrieves the transmit targets, if any, of the voice stream for a player in a session.

GetTransmitTargets(*playerSourceID* **As Long, _**
lFlags **As Long) As Long()**

Parts

playerSourceID
Long value specifying the DVID of the user or group whose target is returned.

lFlags
Reserved. Must be 0.

Return Values

Returns an array of **Long** values containing the DVIDs of the of the transmit targets, if any, of the server's voice stream. When you call this method, this should be the same value as the number of targets set in the **DirectPlayVoiceServer8.SetTransmitTargets** method.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

IDH_DirectPlayVoiceServer8.GetTransmitTargets_dplay_vb

DVERR_NOTALLOWED
DVERR_INVALIDPARAM
DVERR_INVALIDPOINTER
DVERR_BUFFERTOOSMALL
DVERR_NOTCONNECTED
DVERR_NOTINITIALIZED
DVERR_INVALIDFLAGS

Remarks

This method can be used only if the DVSESSION_SERVERCONTROLTARGET flag is specified on creation of the DirectPlay Voice session. If the flag is not specified, this method returns DVERR_NOTALLOWED.

DirectPlayVoiceServer8.Initialize

[#]Initializes the **DirectPlayVoiceClient8** object by associating the **DirectPlayVoiceServer8** object with a Microsoft® DirectPlay® object.

Initialize(*DplayObj* **As Unknown,** *lFlags* **As Long)**

Parts

DplayObj
Pointer to the **IUnknown** interface for the DirectPlay object that this **DirectPlayVoiceServer8** object should use.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_ALREADYINITIALIZED
DVERR_INVALIDPOINTER
DVERR_TRANSPORTNOTINIT
DVERR_NOCALLBACK
DVERR_GENERIC

IDH_DirectPlayVoiceServer8.Initialize_dplay_vb

DirectPlayVoiceServer8.SetSessionDesc

#Sets the session settings.

SetSessionDesc(*ClientConfig* **As DVSESSIONDESC)**

Parts

ClientConfig
DVSESSIONDESC type that contains the session description.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_NOTHOSTING
DVERR_NOTINITIALIZED
DVERR_INVALIDPOINTER
DVERR_SESSIONLOST
DVERR_INVALIDOBJECT

Remarks

After the Microsoft® DirectPlay® Voice session has started, not all the session properties of the **DVSESSIONDESC** structure can be changed. For more information, see **DVSESSIONDESC**.

DirectPlayVoiceServer8.SetTransmitTargets

#Controls the transmission of audio from the client to the specified members of the session.

SetTransmitTargets(*playerSourceID* **As Long, _**
***playerTargetIDs* As Long(), _**
***lFlags* As Long)**

Parts

playerSourceID
DVID of the user whose targets are set.

playerTargetIDs

IDH_DirectPlayVoiceServer8.SetSessionDesc_dplay_vb
IDH_DirectPlayVoiceServer8.SetTransmitTargets_dplay_vb

Array of long values specifying the DVIDs of the players or groups that are to receive the voice transmission. To specify no targets, pass an empty array for this parameter. To specify all players, create a single-item array containing DVID_ALLPLAYERS.

lFlags

Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM

DVERR_NOTALLOWED

DVERR_INVALIDFLAGS

DVERR_INVALIDPOINTER

DVERR_NOTINITIALIZED

DVERR_INVALIDTARGET

Remarks

This method can be used only if the DVSESSION_SERVERCONTROLTARGET flag is specified on creation of the DirectPlay Voice session. If the flag is not specified, this method returns DVERR_NOTALLOWED.

DirectPlayVoiceServer8.StartServerNotification

[#]Registers a **DirectPlayVoiceEvent8** object that is used to capture messages generated by this Microsoft® DirectPlay® Voice session.

StartServerNotification(*event* **As DirectPlayVoiceEvent8)**

Parts

event

DirectPlayVoiceEvent8 object that will receive messages from this client and from the server.

Error Codes

If the method fails, **Err.Number** can be set to the following value.

IDH_DirectPlayVoiceServer8.StartServerNotification_dplay_vb

DVERR_INVALIDPARAM

DirectPlayVoiceServer8.StartSession

#Starts an initialized Microsoft® DirectPlay® Voice session within a running DirectPlay transport session. This method must be successfully called before the clients can complete a connection to the voice session.

StartSession(*SessionDesc* **As DVSESSIONDESC,** *lFlags* **As Long)**

Parts

SessionDesc
DVSESSIONDESC type that contains the session description.

lFlags
Reserved. Must be 0.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_ALREADYPENDING
DVERR_NOTINITIALIZED
DVERR_INVALIDOBJECT
DVERR_INVALIDPOINTER
DVERR_INVALIDFLAGS
DVERR_HOSTING

Remarks

The **DirectPlayVoiceServer8.Initialize** method must be called before this method is called. The voice session can be hosted on any client in the session if the voice session is peer-to-peer. If the voice session is not peer-to-peer, it must be hosted on the transport client, which is the host of an active transport session.

The **DVSESSIONDESC** structure contains the type of voice session to start. The type of voice session can have a dramatic effect on the CPU and bandwidth usage for both the client and the server.

DirectPlayVoiceServer8.StopSession

#Stops the Microsoft® DirectPlay® Voice session.

IDH_DirectPlayVoiceServer8.StartSession_dplay_vb
IDH_DirectPlayVoiceServer8.StopSession_dplay_vb

StopSession(*lFlags* **As Long)**

Parts

lFlags
The following flag can be set.
DVFLAGS_NOHOSTMIGRATE
The host will not migrate regardless of session and transport settings. Use this flag to shut down the voice session completely.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_ALREADYPENDING
DVERR_SESSIONLOST
DVERR_NOTINITIALIZED
DVERR_NOTHOSTING
DVERR_INVALIDFLAGS
DVERR_INVALIDOBJECT

Remarks

This method returns the DVERR_ALREADYPENDING error value if it is called while another thread is processing a **StopSession** request.

DirectPlayVoiceServer8.UnregisterMessageHandler

#Unregisters the current **DirectPlayVoiceServer8** message handler.

UnRegisterMessageHandler()

Remarks

You must call this method before registering a new message handler.

DirectPlayVoiceTest8

#Applications use the method of the **DirectPlayVoiceTest8** class to test the Microsoft® DirectPlay® Voice audio configuration.

IDH_DirectPlayVoiceServer8.UnregisterMessageHandler_dplay_vb
IDH_DirectPlayVoiceTest8_dplay_vb

The **CheckAudioSetup** method of the **DirectPlayVoiceTest8** class is used to check the audio configuration.

DirectPlayVoiceTest8.CheckAudioSetup

#Runs the Audio Setup Wizard on the specified devices. This wizard runs a series of tests on the devices to determine if they are capable of full duplex audio and to ensure that the microphone is plugged in and working correctly on the capture device.

CheckAudioSetup(_
guidPlaybackDevice **As String, _**
guidCaptureDevice **As String, _**
hwndOwner **As Long, _**
lFlags **As Long) As Long**

Parts

guidPlaybackDevice
GUID that identifies the playback device to test. If an empty string is passed for this parameter, Microsoft® DirectPlay® Voice tests the default voice playback device defined by Microsoft® DirectSound®.

guidCaptureDevice
GUID that identifies the capture device to test. If an empty string is passed for this parameter, DirectPlay Voice tests the default voice capture device defined by DirectSound.

hwndOwner
The test wizard invoked by this method is modal. If the calling application has a window that should be the parent window of the wizard, it should pass a handle to that window in this parameter. If the calling application does not have a window, it can pass 0. If the DVFLAGS_QUERYONLY flag is specified, this parameter is not used and the application can pass 0.

lFlags
Flags. The following flags can be set.

DVFLAGS_QUERYONLY
Audio setup is not run. Instead, the method checks the registry to see if the devices have been tested. If the devices have not been tested, the method returns DVERR_RUNSETUP. If the devices have been tested, the method returns DV_FULLDUPLEX if the devices support full duplex audio, or DV_HALFDUPLEX if the devices do not support full duplex audio.

DVFLAGS_ALLOWBACK
Enable the **Back** button on the Welcome page of the wizard. If the user clicks the **Back** button on the Welcome page, the wizard exits, and **CheckAudioSetup** returns DVERR_USERBACK.

IDH_DirectPlayVoiceTest8.CheckAudioSetup_dplay_vb

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DVERR_INVALIDPARAM
DVERR_RUNSETUP

Remarks

This method contains user interface (UI) elements and raises dialog boxes. If the DVFLAGS_QUERYONLY flag is specified, the tests are not actually run and no UI is raised. Instead the registry is checked to determine the results of a previous test of these devices.

Functions

This section contains references for the following Microsoft® DirectPlay® functions. They are designed to simplify the process of handling byte arrays.

- **AddDataToBuffer**
- **AddStringToBuffer**
- **GetDataFromBuffer**
- **GetStringFromBuffer**
- **NewBuffer**

AddDataToBuffer

#Adds nonstring data to a byte array.

AddDataToBuffer(_
Buffer() **As Byte,** _
lData **As Any,** _
lSize **As Long,** _
lOffset **As Long)**

Parameters

Buffer
Byte array to which the data is to be added.

lData
Data that is to be added to *Buffer*.

lSize

IDH_AddDataToBuffer_dplay_vb

Size, in bytes, of the data in *lData*. The simplest way to determine this value is to use the Microsoft® Visual Basic® **LenB** function. You can also use one of the values from the CONST_DPLAYBUFSIZE enumeration.

lOffset

Long value containing the offset, in bytes, to the location in the byte array where the data is to be added. When the function returns, this parameter will be set to the offset of the first byte following the data that has just been added.

Remarks

If the byte array passed to *Buffer* is not large enough to hold the added data, the array will be enlarged without data loss to a sufficient size.

AddStringToBuffer

#Adds a string to a byte array.

AddStringToBuffer(_
Buffer() **As Byte, _**
StringData **As String, _**
lOffset **As Long)**

Parameters

Buffer

Byte array that the data is to be added to.

StringData

String value that is to be added to *Buffer*.

lOffset

Long value containing the offset, in bytes, to the location in the byte array where the data is to be added. When the function returns, this parameter will be set to the offset of the first byte following the data that has just been added.

Remarks

If the byte array passed to *Buffer* is not large enough to hold the added string, the array will be enlarged without data loss to a sufficient size.

GetDataFromBuffer

#Retrieves nonstring data from a byte array.

GetDataFromBuffer(_
Buffer() **As Byte, _**
lData **As Any, _**

IDH_AddStringToBuffer_dplay_vb
IDH_GetDataFromBuffer_dplay_vb

lSize **As Long, _**
lOffset **As Long)**

Parameters

Buffer
Byte array from which the data is to be retrieved.

lData
Variable to receive the data that is retrieved from *Buffer*.

lSize
Size of the data in *lData*, in bytes. The simplest way to determine this value is to use the Visual Basic **LenB** function. You can also use one of the values from the CONST_DPLAYBUFSIZE enumeration.

lOffset
Offset, in bytes, to the location in the byte array from which the data is to be retrieved. When the function returns, this parameter will be set to the offset of the first byte following the data that has just been retrieved.

GetStringFromBuffer

#Retrieves string data from a byte array.

GetStringFromBuffer(_
Buffer() **As Byte, _**
lOffset **As Long) As String**

Parameters

Buffer
Byte array that the data is to be retrieved from.

lOffset
Long value containing the offset, in bytes, to the location in the byte array from which the data is to retrieved. When the function returns, this parameter will be set to the offset of the first byte following the data that has just been retrieved.

Return Values

Returns a **String** value containing the retrieved string.

NewBuffer

#Creates a byte array.

NewBuffer(*Buffer()* **As Byte) As Long**

IDH_GetStringFromBuffer_dplay_vb
IDH_NewBuffer_dplay_vb

Parameters

Buffer
Byte array that will be initialized to a default number of members.

Return Values

Returns a **Long** value containing the offset to the start of the array. This value will always be zero.

Remarks

The default number of members in the returned array is 20.

Types

This section contains information on the following types used with Microsoft® DirectPlay®.

- **DPL_APPLICATION_INFO**
- **DPL_CONNECT_INFO**
- **DPL_CONNECTION_SETTINGS**
- **DPL_MESSAGE_CONNECT**
- **DPL_MESSAGE_CONNECTION_SETTINGS**
- **DPL_MESSAGE_RECEIVE**
- **DPL_PROGRAM_DESC**
- **DPN_APPLICATION_DESC**
- **DPN_CAPS**
- **DPN_CONNECTION_INFO**
- **DPN_GROUP_INFO**
- **DPN_PLAYER_INFO**
- **DPN_SERVICE_PROVIDER_INFO**
- **DPN_SP_CAPS**

- **DPNMSG_ASYNC_OP_COMPLETE**
- **DPNMSG_CONNECT_COMPLETE**
- **DPNMSG_ENUM_HOSTS_QUERY**
- **DPNMSG_ENUM_HOSTS_RESPONSE**
- **DPNMSG_INDICATE_CONNECT**
- **DPNMSG_RECEIVE**
- **DPNMSG_SEND_COMPLETE**
- **DVCAPS**
- **DVCLIENTCONFIG**
- **DVCOMPRESSIONINFO**
- **DVSESSIONDESC**
- **DVSOUNDDEVICECONFIG**

DPL_APPLICATION_INFO

[#]Returned in the **DirectPlay8LobbyClient.GetLocalProgram** method to describe the lobbied application.

```
Type DPL_APPLICATION_INFO
    ApplicationName As String
    guidApplication As String
    lFlags As CONST_DPLSESSION
    lNumRunning As Long
    lNumWaiting As Long
End Type
```

Members

ApplicationName
Name of the lobbied application.

guidApplication
String that specifies the GUID of the lobbied application.

lFlags
One of the constants of **CONST_DPLSESSION** enumeration that describes the current status of the lobbied application.

lNumRunning
Number of instances of the application.

lNumWaiting
Number of clients waiting to connect to the lobbied application.

IDH_DPL_APPLICATION_INFO_dplay_vb

DPL_CONNECT_INFO

#Used to specify connection information for a lobby client when connecting to the lobby application in the **DirectPlay8LobbyClient.ConnectApplication** method.

```
Type DPL_CONNECT_INFO
   ConnectionSettings As DPL_CONNECTION_SETTINGS
   guidApplication As String
   lFlags As CONST_DPLCONNECT
End Type
```

Members

ConnectionSettings
DPL_CONNECTION_SETTINGS type with the settings for this connection.

guidApplication
String that specifies the GUID of the application.

lFlags
One of the constants of the **CONST_DPLCONNECT** enumeration that determine connection behavior.

DPL_CONNECTION_SETTINGS

#**H**olds connection information for lobbied connections.

```
Type DPL_CONNECTION_SETTINGS
   AddressDeviceUrl As String
   AddressSenderUrl As String
   ApplicationDescription As DPN_APPLICATION_DESC
   lFlags As CONST_DPLCONNECTSETTINGS
   PlayerName As String
End Type
```

Members

AddressDeviceUrl
String set to the device URL.

AddressSenderUrl
String set to the sender URL.

ApplicationDescription
DPN_APPLICATION_DESC type containing the application description.

lFlags

IDH_DPL_CONNECT_INFO_dplay_vb
IDH_DPL_CONNECTION_SETTINGS_dplay_vb

Value from the **CONST_CONNECTSETTINGS** enumeration. This member is set to DPLCONNECTSETTINGS_HOST if your player is the session host, and 0 otherwise.

PlayerName
String value set to the player's name.

DPL_MESSAGE_CONNECT

#This type is generated by Microsoft® DirectPlay® when a lobby client connects to the lobbied application through the **DirectPlay8LobbyClient.ConnectApplication** method.

```
Type DPL_MESSAGE_CONNECT
   ConnectId As Long
   dplMsgCon As DPL_CONNECTION_SETTINGS
   LobbyConnectData() As Byte
End Type
```

Members

ConnectId
Handle used to identify the connection. This handle is used to in subsequent calls to **DirectPlay8LobbyClient.Send** and **DirectPlay8LobbyClient.ReleaseApplication**.

dplMsgCon
DPL_CONNECTION_SETTINGS type with the connection settings.

LobbyConnectData
Lobby connection data.

DPL_MESSAGE_CONNECTION_SETTINGS

#Used with the **DirectPlay8LobbyEvent.ConnectionSettings** method.

```
Type DPL_MESSAGE_CONNECTION_SETTINGS
   dplConnectionSettings
   hSender As Long
End Type
```

Members

dplConnectionSettings
DPL_CONNECTION_SETTINGS type that describes the connection settings.

hSender

IDH_DPL_MESSAGE_CONNECT_dplay_vb
IDH_DPL_MESSAGE_CONNECTION_SETTINGS_dplay_vb

Long value that contains the handle for the connection that sent this message.

DPL_MESSAGE_RECEIVE

[#]Microsoft® DirectPlay® generates this type when the target receives a message sent by the **DirectPlay8LobbyClient.Send** or **DirectPlay8LobbiedApplication.Send** method.

```
Type DPL_MESSAGE_RECEIVE
   buffer() As Byte
   lBufferSize As Long
   Sender As Long
End Type
```

Members

buffer
BYTE array with the message data.

lBufferSize
Size of the message data contained in the **buffer** member.

Sender
Handle of the client that sent the message.

DPL_PROGRAM_DESC

[#]Describes a Microsoft® DirectPlay® lobby-aware application.

```
Type DPL_PROGRAM_DESC
   ApplicationName As String
   CommandLine As String
   CurrentDirectory As String
   Description As String
   ExecutableFilename As String
   ExecutablePath As String
   guidApplication As String
   LauncherFilename As String
   LauncherPath As String
   lFlags As Long
End Type
```

Members

ApplicationName
Application name.

CommandLine

IDH_DPL_MESSAGE_RECEIVE_dplay_vb
IDH_DPL_PROGRAM_DESC_dplay_vb

Command line arguments.

CurrentDirectory
Current directory.

Description
Application description.

ExecutableFilename
File name of the application executable.

ExecutablePath
Path of the application executable.

guidApplication
String that specifies the GUID of the application.

LauncherFilename
File name of the launcher executable.

LauncherPath
Path of the launcher executable.

lFlags
Reserved. Must be 0.

DPN_APPLICATION_DESC

[#]Describes the settings for a Microsoft® DirectPlay® application.

```
Type DPN_APPLICATION_DESC
   guidApplication As String
   guidInstance As String
   lCurrentPlayers As Long
   lFlags As Long
   lMaxPlayers As Long
   Password As String
   SessionName As String
End Type
```

Members

guidApplication
Application GUID.

guidInstance
Globally unique identifier (GUID) that is generated at startup representing the instance of this application. This member is an [out] parameter when calling the **GetApplicationDesc** method exposed by the **DirectPlay8Peer**, **DirectPlay8Client**, and **DirectPlay8Server** objects. It is an optional [in] parameter when calling the **Connect** method exposed by the **DirectPlay8Peer** and **DirectPlay8Client** objects. It must be set to NULL when you call the **SetApplicationDesc** method exposed by the **DirectPlay8Server** and

IDH_DPN_APPLICATION_DESC_dplay_vb

DirectPlay8Peer objects. You can not obtain this GUID by calling the **DirectPlay8Server.Host** or **DirectPlay8Peer.Host** methods. You must obtain the GUID by calling a **GetApplicationDesc** method.

lCurrentPlayers
Number of clients currently connected to the session.

lFlags
One of the following flags describing application behavior.

DPNSESSION_CLIENT_SERVER
Specifies that this type of session is client/server. This flag cannot be combined with DPNSESSION_MIGRATE_HOST.

DPNSESSION_NODPNSVR
Specifies that you do not want enumerations forwarded to your host from DPNSVR. See Using the DirectPlay DPNSVR Application for a further discussion of DPNSVR.

DPNSESSION_REQUIREPASSWORD
Specifies that the session is password protected. If this flag is set, **Password** must be a valid string.

DPNSESSION_MIGRATE_HOST
Used in peer-to-peer sessions, setting this flag will enable host migration. This flag cannot be combined with DPNSESSION_CLIENT_SERVER.

lMaxPlayers
Maximum number of clients allowed in the session. Set this member to 0 to indicate an unlimited number of players.

Password
String specifying the Unicode™ password that is required to connect to the session. This must be an empty string if the DPNSESSION_REQUIREPASSWORD is not set in the **lFlags** member.

SessionName
String specifying the Unicode name of the session.

Remarks

The **lMaxPlayers**, **Password**, and **SessionName** members can be set when calling the **Host** or **SetApplicationDesc** methods exposed by the **DirectPlay8Server** and **DirectPlay8Peer** objects.

DPN_CAPS

#Used to set and retrieve general parameters for Microsoft® DirectPlay® sessions.

```
Type DPN_CAPS
    lConnectRetries As Long
    lConnectTimeout As Long
```

IDH_DPN_CAPS_dplay_vb

```
    lFlags As Long
    lTimeoutUntilKeepAlive As Long
End Type
```

Members

lConnectRetries
Number of connection retries DirectPlay should make during the connection process.

lConnectTimeout
Number of milliseconds DirectPlay should wait before it retries a connection request.

lFlags
Reserved, must be 0.

lTimeoutUntilKeepAlive
Number of milliseconds DirectPlay should wait, since the last time it received a packet from an endpoint, before it sends a keep-alive message.

DPN_CONNECTION_INFO

#Used to retrieve statistics for the connection between you and a particular endpoint.

```
Type DPN_CONNECTION_INFO
    lSize As Long
    lRoundTripLatencyMS As Long
    lThroughPutBPS As Long
    lPeakThroughPutBPS As Long
    lBytesSentGuaranteed As Long
    lPacketsSentGuaranteed As Long
    lBytesSentNonGuaranteed As Long
    lPacketsSentNonGuaranteed As Long
    lBytesRetried As Long
    lPacketsRetried As Long
    lBytesDropped As Long
    lPacketsDropped As Long
    lBandwidthBPS As Long
    lMessagesTransmittedHighPriority As Long
    lMessagesTimedOutHighPriority As Long
    lMessagesTransmittedNormalPriority As Long
    lMessagesTimedOutNormalPriority As Long
    lMessagesTransmittedLowPriority As Long
    lMessagesTimedOutLowPriority As Long
    lBytesReceivedGuaranteed As Long
    lPacketsReceivedGuaranteed As Long
```

IDH_DPN_CONNECTION_INFO_dplay_vb

```
    lBytesReceivedNonGuaranteed As Long
    lPacketsReceivedNonGuaranteed As Long
    lMessagesReceived As Long
End Type
```

Members

lSize
Long value set to the size of the type.

lRoundTripLatencyMS
Long value set to the approximate time, in milliseconds, for a packet to reach the endpoint and return to the local computer. This number will fluctuate throughout the session, as network conditions change.

lThroughPutBPS
Long value set to the approximate throughput, in bytes per second (BPS). This number will fluctuate throughout the session, as link conditions change.

lPeakThroughPutBPS
Long value set to the peak throughput, in bytes per second. This number will fluctuate throughout the session, as link conditions change.

lBytesSentGuaranteed
Long value set to the amount, in bytes, of guaranteed messages that have been sent.

lPacketsSentGuaranteed
Long value set to the number of guaranteed packets that have been sent.

lBytesSentNonGuaranteed
Long value set to the amount, in bytes, of nonguaranteed messages that have been sent.

lPacketsSentNonGuaranteed
Long value set to the number of nonguaranteed packets that have been sent.

lBytesRetried
Long value set to the amount, in bytes, of messages that have been retried.

lPacketsRetried
Long value set to the number of packets that have been retried.

lBytesDropped
Long value set to the amount, in bytes, of messages that have been dropped.

lPacketsDropped
Long value set to the number of packets that have been dropped.

lBandwidthBPS
Long value set to the bandwidth, in bytes per second.

lMessagesTransmittedHighPriority
Long value set to the number of high-priority messages that have been transmitted.

lMessagesTimedOutHighPriority
Long value set to the number of high-priority messages that have timed out.

lMessagesTransmittedNormalPriority
Long value set to the number of normal-priority messages that have been transmitted.

lMessagesTimedOutNormalPriority
Long value set to the number of normal-priority messages that have timed out.

lMessagesTransmittedLowPriority
Long value set to the number of low-priority messages that have been transmitted.

lMessagesTimedOutLowPriority
Long value set to the number of low-priority messages that have timed out.

lBytesReceivedGuaranteed
Long value set to the amount, in bytes, of guaranteed messages that have been received.

lPacketsReceivedGuaranteed
Long value set to the number of guaranteed packets that have been received.

lBytesReceivedNonGuaranteed
Long value set to the amount, in bytes, of nonguaranteed messages that have been received.

lPacketsReceivedNonGuaranteed
Long value set to the number of nonguaranteed packets that have been received.

lMessagesReceived
Long value set to the number of messages that have been received.

DPN_GROUP_INFO

#Describes static group information.

```
Type DPN_GROUP_INFO
   lGroupFlags As CONST_DPNGROUPINFOFLAGS
   lInfoFlags As CONST_DPNINFO
   name As String
End Type
```

Members

lGroupFlags
One of the constants of the **CONST_DPNGROUPINFOFLAGS** enumeration that describe the flags set for the player.

lInfoFlags
One of the constants of the **CONST_DPNINFO** enumeration that describe the type of data contained in this type.

Name
Name of the group.

IDH_DPN_GROUP_INFO_dplay_vb

Remarks

When using this type in the **DirectPlay8Peer.GetGroupInfo** and **DirectPlay8Server.GetGroupInfo** methods, **lInfoFlags** must be set to 0.

DPN_PLAYER_INFO

#Describes static player information.

```
Type DPN_PLAYER_INFO
   lInfoFlags As CONST_DPNINFO
   lPlayerFlags As CONST_DPNPLAYINFOFLAGS
   name As String
End Type
```

Members

lInfoFlags
One of the constants in the **CONST_DPNINFO** enumeration that specify the type of information contained in this type.

lPlayerFlags
One of the constants in the **CONST_DPNPLAYINFOFLAGS** enumeration describing the flags set for the player.

name
Name of the player.

Remarks

When using this type in the **DirectPlay8Peer.GetPeerInfo** and **DirectPlay8Server.GetClientInfo** methods, **lInfoFlags** must be set to 0.

DPN_SERVICE_PROVIDER_INFO

#Used when enumerating information for a specific service provider.

```
Type DPN_SERVICE_PROVIDER_INFO
   GUID As String
   lFlags As Long
   name As String
End Type
```

Members

GUID
GUID for the service provider.

IDH_DPN_PLAYER_INFO_dplay_vb
IDH_DPN_SERVICE_PROVIDER_INFO_dplay_vb

lFlags
Reserved. Must be 0.

name
Name of the service provider.

DPN_SP_CAPS

#Used to set and retrieve parameters for service providers.

```
Type DPN_SP_CAPS
    lBuffersPerThread As Long
    lDefaultEnumCount As Long
    lDefaultEnumRetryInterval As Long
    lDefaultEnumTimeout As Long
    lFlags As Long
    lMaxEnumPayloadSize As Long
    lNumThreads As Long
    lSystemBufferSizeAs Long
End Type
```

Members

lBuffersPerThread
The number of outstanding receive buffers allocated for each DirectPlay thread. If you increase the number of receive buffers, DirectPlay can pull more data out of the operating system buffers. However, you may also increase latency if data is arriving faster than your application can process it.

lDefaultEnumCount
Long value that specifies the default enumeration count.

lDefaultEnumRetryInterval
Long value that specifies the default retry interval, in milliseconds.

lDefaultEnumTimeout
Long value that specifies the default enumeration timeout value, in milliseconds.

lFlags
Long value that can be a combination of the following flags.

DPNSPCAPS_SUPPORTSALLADAPTERS
The service provider is supported on all the adapters that are present on the system.

DPNSPCAPS_SUPPORTSBROADCAST
For IP and IPX applications, the service provider has the ability to find games by broadcasting, if sufficient addressing information is not provided.

DPNSPCAPS_SUPPORTSDPNSRV

IDH_DPN_SP_CAPS_dplay_vb

Dpnsvr.exe will provide port sharing for the given SP. Currently, this flag is available on IP and IPX only. See Using the DirectPlay DPNSVR Application for a further discussion of DPNSVR.

lMaxEnumPayloadSize

Long that specifies the maximum size of the payload information that can be sent in the **ResponseData** member of the types that accompany the **DirectPlay8Event.EnumHostQuery** and **DirectPlay8Event.EnumHostQuery** methods**.**

lNumThreads

Number of threads the service provider will use for servicing network requests. The default value is based on an algorithm that takes into account the number of processors on the system. Most applications will not need to modify this value.

After a service provider is active in your process you may only *increase* this value. Decreasing the value will have no effect. The setting is process wide, meaning it will effect your current Microsoft® DirectPlay® object and any other DirectPlay objects in your process.

lSystemBufferSizeAs

The size of the operating system buffer. This buffer holds data from the communications device when your application cannot process data as fast as it arrives. The purpose of this buffer is to prevent data loss if you receive a sudden burst of data, or if the receive threads are momentarily stalled. Increasing **lSystemBufferSize** may increase latency if your application cannot process the received data fast enough. You can eliminate the operating system buffer by setting **lSystemBufferSize** to 0. However, if you do so, you run the risk of losing data if you cannot process the received data as fast as it arrives.

DPNMSG_ASYNC_OP_COMPLETE

[#]Used in the *dpnotify* parameter of the **DirectPlay8Event.AsyncOpComplete** method. This type is generated by Microsoft® DirectPlay® when an asynchronous operation has completed.

```
Type DPNMSG_ASYNC_OP_COMPLETE
   AsyncOpHandle As Long
   hResultCode As Long
End Type
```

Members

AsyncOpHandle

Long value specifying the handle of the asynchronous operation.

hResultCode

Long value specifying the result of the asynchronous operation.

IDH_DPNMSG_ASYNC_OP_COMPLETE_dplay_vb

DPNMSG_CONNECT_COMPLETE

#Used in the *dpnotify* parameter of the **DirectPlay8Event.ConnectComplete** method. This type is generated by Microsoft® DirectPlay® when a connection request has been completed.

```
Type DPNMSG_CONNECT_COMPLETE
   AsyncOpHandle As Long
   hResultCode As Long
   ReplyData() as Byte
End Type
```

Members

AsyncOpHandle
Long value specifying the handle of the asynchronous operation.

hResultCode
Long value specifying result of the connection request.

ReplyData
Byte array containing the connection data returned by the host or server.

DPNMSG_ENUM_HOSTS_QUERY

#Used in the *dpnotify* parameter of the **DirectPlay8Event.EnumHostsQuery** method. This type is generated by Microsoft® DirectPlay® when a player has requested a host enumeration.

```
Type DPNMSG_ENUM_HOSTS_QUERY
   AddressDeviceUrl As String
   AddressSenderUrl As String
   lMaxResponseDataSize As Long
   ReceivedData() As Byte
   ResponseData() As Byte
End Type
```

Members

AddressDeviceUrl
String value containing the device's URL.

AddressSenderUrl
String value containing the sender's URL.

lMaxResponseDataSize
Long value specifying the size of the response data block.

ReceivedData

IDH_DPNMSG_CONNECT_COMPLETE_dplay_vb
IDH_DPNMSG_ENUM_HOSTS_QUERY_dplay_vb

Data received from the enumeration.

ResponseData
Data block containing the response to the enumeration request.

DPNMSG_ENUM_HOSTS_RESPONSE

#Used in the *dpnotify* parameter of the **DirectPlay8Event.EnumHostsResponse** method. This type is generated by Microsoft® DirectPlay® when the host responds with information to an enumeration request.

```
Type DPNMSG_ENUM_HOSTS_RESPONSE
   AddressDeviceUrl As String
   AddressSenderUrl As String
   ApplicationDescription As DPN_APPLICATION_DESC
   lRoundTripLatencyMS As Long
   ResponseData As Byte
End Type
```

Members

AddressDeviceUrl
String specifying the device's address URL

AddressSenderUrl
String specifying the sender's address URL.

ApplicationDescription
DPN_APPLICATION_DESC type that describes the application.

lRoundTripLatencyMS
Round-trip latency, in milliseconds.

ResponseData
Response data from the enumeration.

DPNMSG_INDICATE_CONNECT

#Used in the *dpnotify* parameter of the **DirectPlay8Event.IndicateConnect** method. This type is generated by Microsoft® DirectPlay® when a player has requested a connection to the session.

```
Type DPNMSG_INDICATE_CONNECT
   AddressDeviceUrl As String
   AddressPlayerUrl As String
   UserData() As Byte
End Type
```

IDH_DPNMSG_ENUM_HOSTS_RESPONSE_dplay_vb
IDH_DPNMSG_INDICATE_CONNECT_dplay_vb

Members

AddressDeviceUrl
String specifying the device's address URL

AddressSenderUrl
String specifying the sender's address URL.

UserData
Data of the connecting player.

DPNMSG_RECEIVE

#Used in the *dpnotify* parameter of the **DirectPlay8Event.Receive** method. This type is generated by Microsoft® DirectPlay® when a message has been received.

```
Type DPNMSG_RECEIVE
   idSender As Long
   iDataSize As Long
   ReceivedData() As Byte
End Type
```

Members

idSender
Long value specifying the identifier of the message sender.

iDataSize
Long value specifying the number of bytes in the **ReceivedData** member. Because the byte array is zero-based, the upper limit of the array will be **IDataSize**-1.

ReceivedData
Byte array containing the message data that was sent.

DPNMSG_SEND_COMPLETE

#Used in the *dpnotify* parameter of the **DirectPlay8Event.SendComplete** method. This type is generated by Microsoft® DirectPlay® when a message has been received by the recipient.

```
Type DPNMSG_SEND_COMPLETE
   AsyncOpHandle As Long
   hResultCode As Long
   lSendTime As Long
End Type
```

IDH_DPNMSG_RECEIVE_dplay_vb
IDH_DPNMSG_SEND_COMPLETE_dplay_vb

Members

AsyncOpHandle
Long value specifying the handle of the asynchronous operation.

hResultCode
Long value specifying the success or failure of the message send.

lSendTime
Time, in milliseconds, between send call and completion.

DPNMSG_TERMINATE_SESSION

#Used in the **DirectPlay8Event.TerminateSession** method.

```
Type DPNMSG_TERMINATE_SESSION
   hResultCode As Long
   TerminateData() As Byte
End Type
```

Members

hResultCode
Long value specifying why the session was terminated. This member is set to DPNERR_HOSTTERMINATEDSESSION if the session was peer-to-peer, and the host called **DirectPlay8Peer.TerminateSession**. If the session was ended by the host calling **Close**, or if the host stops responding, **hResultCode** is set to DPNERR_CONNECTIONLOST.

TerminateData
Termination data. If **hResultCode** is set to DPNERR_HOSTTERMINATEDSESSION, **TerminateData** points to the data block that the host passed through the *UserData* parameter of **DirectPlay8Peer.TerminateSession**. If **hResultCode** is set to DPNERR_CONNECTIONLOST, **TerminateData** will be empty

DVCAPS

#Describes the capabilities of the Microsoft® DirectPlay® Voice client object.

```
Type DVCAPS
   lFlags As Long
End Type
```

Members

lFlags
Reserved. Must be 0.

IDH_DPNMSG_TERMINATE_SESSION_dplay_vb
IDH_DVCAPS_dplay_vb

DVCLIENTCONFIG

[#]Controls the run-time parameters for the client. This type is first used in the call to **DirectPlayVoiceClient8.Connect**, where it sets the initial state of these parameters. The type can be retrieved after a connection has been made by calling **DirectPlayVoiceClient8.GetClientConfig**, and set using **DirectPlayVoiceClient8.SetClientConfig**.

```
Type DVCLIENTCONFIG
    lBufferAggressiveness As CONST_DVBUFFERAGGRESSIVENESS
    lBufferQuality As CONST_DVBUFFERQUALITY
    lFlags As CONST_DVCLIENTCONFIGENUM
    lNotifyPeriod As Long
    lPlaybackVolume As Long
    lRecordVolume As Long
    lThreshold As CONST_DVTHRESHOLD
End Type
```

Members

lBufferAggressiveness
One of the constants of the **CONST_DVBUFFERAGGRESSIVENESS** enumeration that specifies the buffer aggressiveness setting for the adaptive buffer algorithm.

lBufferQuality
One of the constants of the **CONST_DVBUFFERQUALITY** enumeration that specifies the buffer quality setting for the adaptive buffering algorithm. For most applications, this should be set to DVBUFFERQUALITY_DEFAULT. It can be set to anything in the range of DVBUFFERQUALITY_MIN to DVBUFFERQUALITY_MAX. In general, the higher the value, the higher the quality of the voice but the higher the latency. The lower the value, the lower the latency but the lower the quality.

lFlags
Combination of flags from the **CONST_DVCLIENTCONFIGENUM** enumeration.

lNotifyPeriod
Specifies how often you want to receive DVMSGID_OUTPUTLEVEL and DVMSGID_INPUTLEVEL (if session is full duplex) messages. If this value is set to 0, these messages are disabled. The value specifies the number of milliseconds between these messages. DVNOTIFYPERIOD_MINPERIOD specifies the minimum period between messages that is allowed.

lPlaybackVolume
Specifies what the volume of the playback should be set to. Adjusting this volume adjusts both the main buffer and all 3-D sound buffers. See the **DirectSoundPrimaryBuffer8.SetVolume** method for the valid values for this

IDH_DVCLIENTCONFIG_dplay_vb

member. You can specify DVPLAYBACKVOLUME_DEFAULT to use a default value that is appropriate for most situations (full volume).

lRecordVolume

Specifies what the volume of the recording should be set to. See the **DirectSoundPrimaryBuffer8.SetVolume** method for the valid values for this member.

If automatic gain control is enabled, this value can be set to DVRECORDVOLUME_LAST, which tells the system to use the current volume as determined by the automatic gain control algorithm. If a value other then DVRECORDVOLUME_LAST is specified in combination with automatic gain control, this value will be used to restart the algorithm at the specified value.

On return from a call to **DirectPlayVoiceClient8.GetClientConfig**, this value will contain the current recording volume. When adjusting the recording volume, Microsoft® DirectPlay® Voice will adjust the volume for the microphone (if a microphone volume is present for the card) and the master recording volume (if one is present on the card). If neither a microphone volume nor a master record volume is present, DirectPlay Voice will be unable to adjust the recording volume.

lThreshold

One of the constants of the **CONST_DVTHRESHOLD** enumeration that specifies the input level used to trigger voice transmission if the DVCLIENTCONFIG_MANUALVOICEACTIVATED flag is specified in the **lFlags** member. When the flag is specified, this value can be set to anywhere in the range of DVTHRESHOLD_MIN to DVTHRESHOLD_MAX. Additionally, DVTHRESHOLD_DEFAULT can be set to use a default value.

If DVCLIENTCONFIG_MANUALVOICEACTIVATED or DVCLIENTCONFIG_AUTOVOICEACTIVATED is not specified in the **lFlags** member of this type (indicating push-to-talk mode) this value must be set to DVTHRESHOLD_UNUSED.

DVCOMPRESSIONINFO

[#]Describes the attributes of a specific Microsoft® DirectPlay® Voice compression type.

```
Type DVCOMPRESSIONINFO
   guidType As String
   lFlags As Long
   lMaxBitsPerSecond As Long
   strDescription As String
   strName As String
End Type
```

IDH_DVCOMPRESSIONINFO_dplay_vb

Members

guidType
String value specifying the GUID used to identify this compression type by DirectPlay Voice.

lFlags
Reserved; must be 0.

lMaxBitsPerSecond
Maximum number of bits per second claimed by the codec.

strDescription
Description of the codec.

strName
Name describing the codec.

DVSESSIONDESC

[#]Describes the desired or current session settings for the Microsoft® DirectPlay® Voice server. This structure is used by the voice session host to configure the session, and by the session host and the clients to retrieve information about the current session. The **lFlags**, **lSessionType**, and **guidCT** members can only be set when the host starts the voice session. The host can change the buffer settings at any time.

```
Type DVSESSIONDESC
   guidCT As String
   lBufferAggressiveness As CONST_DVBUFFERAGGRESSIVENESS
   lBufferQuality As CONST_DVBUFFERQUALITY
   lFlags As CONST_DVSESSION
   lSessionType As CONST_DVSESSIONTYPE
End Type
```

Members

guidCT
String value specifying the GUID of the compression type of the session.

lBufferAggressiveness
One of the constants of the **CONST_DVBUFFERAGGRESSIVENESS** enumeration that specifies the buffer aggressiveness setting. This member is unused for all session types except mixing sessions. For all sessions except mixing sessions, set this member to DVBUFFERAGGRESIVENESS_DEFAULT.

Allowable values are between DVBUFFERAGGRESIVENESS _MIN and DVBUFFERAGGRESIVENESS _MAX. Additionally, this member can be set to the following value.

DVBUFFERAGGRESIVENESS_DEFAULT

IDH_DVSESSIONDESC_dplay_vb

Specifying this value tells DirectPlay Voice to use the system default for this value, which is adjustable through a registry entry that can also be set through Control Panel.

lBufferQuality

One of the constants of the **CONST_DVBUFFERQUALITY** enumeration that specifies the buffer quality setting. This member is unused for all session types except mixing sessions. For all sessions except mixing sessions, set this member to DVBUFFERQUALITY_DEFAULT.

Allowable values are between DVBUFFERQUALITY_MIN and DVBUFFERQUALITY_MAX. Additionally, this member can be set to the following value.

DVBUFFERQUALITY_DEFAULT
Specifying this value tells DirectPlay Voice to use the system default for this value, which is adjustable through a registry entry that can also be set through Sounds and Multimedia in Control Panel.

lFlags

Combination of the flags of the **CONST_DVSESSION** enumeration.

lSessionType

One of the flags of the **CONST_DVSESSIONTYPE** enumeration to specify the type of DirectPlay Voice session to run. The DVSESSIONTYPE_PEER flag is not available in client/server sessions; all other flags are valid for all session types.

DVSOUNDDEVICECONFIG

[#]Used to set and retrieve information about the sound device configuration and cannot be changed once a connection has been made. After a connection is made, you can retrieve the current sound device configuration by calling **DirectPlayVoiceClient8.GetSoundDeviceConfig**.

```
Type DVSOUNDDEVICECONFIG
   guidCaptureDevice As String
   guidPlaybackDevice As String
   hwndAppWindow As Long
   lFlags As CONST_DVSOUNDEFFECT
   lMainBufferFlags As CONST_DSBPLAYFLAGS
   lMainBufferPriority As Long
   MainSoundBuffer As DirectSoundSecondaryBuffer8
End Type
```

Members

guidCaptureDevice

When this type is used in **DirectPlayVoiceClient8.Connect** method, this member specifies the GUID of the device used for capture.

IDH_DVSOUNDDEVICECONFIG_dplay_vb

When this type is used in the **DirectPlayVoiceClient8.GetSoundDeviceConfig** method, this member will contain the actual device GUID used for capture.

guidPlaybackDevice

When this type is used in the **DirectPlayVoiceClient8.Connect** method, this member specifies the GUID of the device used for playback.

When this type is used in the **DirectPlayVoiceClient8.GetSoundDeviceConfig** method, this member contains the actual device GUID used for playback.

hwndAppWindow

Must be set to the handle of the window that will be used to determine focus for sound playback. See **DirectSound8.SetCooperativeLevel** for information on Microsoft® DirectSound® focus. If you do not have a window to use for focus, use **GetDesktopWindowHandle** to use the desktop window.

lFlags

Combination of the flags from the **CONST_DVSOUNDEFFECT** enumeration.

lMainBufferFlags

Flags values taken from the **CONST_DSBPLAYFLAGS** enumeration that specify how to play the buffer.

lMainBufferPriority

Passed directly to the *dwPriority* parameter of the **DirectSoundSecondaryBuffer8.Play** method when **Play** is called on the main buffer.

MainSoundBuffer

DirectSoundSecondaryBuffer8 object, which is used to create the Microsoft DirectPlay® Voice main buffer. This parameter can be either NULL or a user-created DirectSound buffer. If this member is set to NULL, DirectPlay Voice will create a buffer for the main voice buffer. If users specify a buffer here, DirectPlay Voice will use their buffer for the main voice buffer.

- The buffer must be 22 kilohertz, 16-bit, Mono format.
- The buffer must be at least 1 second in length.
- The buffer must have been created with the DSBCAPS_GETCURRENTPOSITION2 and DSBCAPS_CTRL3D flags.
- The buffer must not be a primary buffer.
- The buffer must not be playing when it is passed to DirectPlay.
- The buffer must not be locked when it is passed to DirectPlay.

Enumerations

Microsoft® DirectPlay® uses enumerations to group constants in order to take advantage of the statement completion feature of Microsoft® Visual Basic®. The enumerations used in DirectPlay are:

- **CONST_DPLAYBUFSIZE**
- **CONST_DPLCONNECT**
- **CONST_DPLSESSION**
- **CONST_DPNCANCELFLAGS**
- **CONST_DPNENUMCLIENTGROUPFLAGS**
- **CONST_DPNERR**
- **CONST_DPNGROUPINFOFLAGS**
- **CONST_DPNINFO**
- **CONST_DPNLOBBY**
- **CONST_DPNMESSAGEID**
- **CONST_DPNOPERATIONS**
- **CONST_DPNPLAYERGROUPFLAGS**
- **CONST_DPNPLAYINFOFLAGS**
- **CONST_DPNSENDFLAGS**
- **CONST_DPNSESSIONFLAGS**
- **CONST_DPNSPCAPS**
- **CONST_DPNWAITTIME**
- **CONST_DVBUFFERAGGRESSIVENESS**
- **CONST_DVBUFFERQUALITY**
- **CONST_DVCLIENTCONFIGENUM**
- **CONST_DVERR**
- **CONST_DVFLAGS**
- **CONST_DVMESSAGE**
- **CONST_DVNOTIFY**
- **CONST_DVPLAYBACKVOLUME**
- **CONST_DVSESSION**
- **CONST_DVSESSIONTYPE**
- **CONST_DVSOUNDEFFECT**
- **CONST_DVTHRESHOLD**

CONST_DPLAYBUFSIZE

#Used to specify the data type used in buffers.

```
Enum CONST_DPLAYBUFSIZE
   SIZE_BOOLEAN = 2
   SIZE_BYTE = 1
```

IDH_CONST_DPLAYBUFSIZE_dplay_vb

```
    SIZE_CURRENCY = 8
    SIZE_DATE = 8
    SIZE_DECIMAL = 14
    SIZE_DOUBLE = 8
    SIZE_INTEGER = 2
    SIZE_LONG = 4
    SIZE_SINGLE = 4
End Enum
```

Constants

SIZE_BOOLEAN
Size of a BOOLEAN variable, in bytes.

SIZE_BYTE
Size of a BYTE variable, in bytes.

SIZE_CURRENCY
Size of a CURRENCY variable, in bytes.

SIZE_DATE
Size of a DATE variable, in bytes.

SIZE_DECIMAL
Size of a DECIMAL variable, in bytes.

SIZE_DOUBLE
Size of a DOUBLE variable, in bytes.

SIZE_INTEGER
Size of an INTEGER variable, in bytes.

SIZE_LONG
Size of a LONG variable, in bytes.

SIZE_SINGLE
Size of a SINGLE variable, in bytes.

CONST_DPLCONNECT

#Used in the *lFlags* parameter of the **DirectPlay8LobbyClient.ConnectApplication** method to determine connection behavior.

```
Enum CONST_DPLCONNECT
    DPLCONNECT_LAUNCHNEW = 1
    DPLCONNECT_LAUNCHNOTFOUND = 2
End Enum
```

Constants

DPLCONNECT_LAUNCHNEW
Launch a new instance of the application.

IDH_CONST_DPLCONNECT_dplay_vb

DPLCONNECT_LAUNCHNOTFOUND
Launch a new instance of the application if there is no application running that can supply launch settings.

CONST_CONNECTSETTINGS

#Used in the **DPL_CONNECTION_SETTINGS** type.

```
Enum CONST_CONNECTSETTINGS
   DPLCONNECTSETTINGS_HOST = 1
End Enum
```

Constants

DPLCONNECTSETTINGS_HOST
Your player is the session host.

CONST_DPLSESSION

#Used in the *lStatus* parameter of the **DirectPlay8LobbiedApplication.UpdateStatus** method to set the current state of the connection between the lobby client and the lobbied application.

```
Enum CONST_DPLSESSION
   DPLSESSION_CONNECTED = 1
   DPLSESSION_COULDNOTCONNECT = 2
   DPLSESSION_DISCONNECTED = 3
   DPLSESSION_HOSTMIGRATED = 5
   DPLSESSION_HOSTMIGRATEDHERE = 6
   DPLSESSION_TERMINATED = 4
End Enum
```

Constants

DPLSESSION_CONNECTED
The lobbied application is currently connected to a session.

DPLSESSION_COULDNOTCONNECT
The lobbied application could not connect to the session.

DPLSESSION_DISCONNECTED
The lobbied application is currently disconnected from the session.

DPLSESSION_HOSTMIGRATED
The host of a peer-to-peer session has migrated. The local client is not the new host.

DPLSESSION_HOSTMIGRATEDHERE

IDH_CONST_CONNECTSETTINGS_dplay_vb
IDH_CONST_DPLSESSION_dplay_vb

The host of a peer-to-peer session has migrated. The local client is the new host.

DPLSESSION_TERMINATED
The connection between session host and the lobbied application has been terminated.

CONST_DPNCANCELFLAGS

#Used in the *lFlags* parameter of the **DirectPlay8Client.CancelAsyncOperation**, **DirectPlay8Server.CancelAsyncOperation** and **DirectPlay8Peer.CancelAsyncOperation** methods to specify which type of asynchronous operation to cancel.

```
Enum CONST_DPNCANCELFLAGS
   DPNCANCEL_ALL_OPERATIONS = 32768 (&H8000)
   DPNCANCEL_CONNECT = 1
   DPNCANCEL_ENUM = 2
   DPNCANCEL_SEND = 4
End Enum
```

Constants

DPNCANCEL_ALL_OPERATIONS
Cancel all asynchronous requests.

DPNCANCEL_CONNECT
Cancel an asynchronous **Connect** request.

DPNCANCEL_ENUM
Cancel all asynchronous **EnumHosts** requests. A single **EnumHosts** request can be canceled by specifying the handle returned from the **EnumHosts** method.

DPNCANCEL_SEND
Cancel an asynchronous **Send** request.

CONST_DPNENUMCLIENTGROUPFLAGS

#Used in the *lFlags* parameter of the **DirectPlay8Peer.GetCountClientsAndGroups** and the **DirectPlay8Server.GetCountClientsAndGroups** methods to determine which type of enumeration to return.

```
Enum CONST_DPNENUMCLIENTGROUPFLAGS
   DPNENUM_ALL = 17 (&H11)
   DPNENUM_GROUP_MULTICAST = 32 (&H20)
```

IDH_CONST_DPNCANCELFLAGS_dplay_vb
IDH_CONST_DPNENUMCLIENTGROUPFLAGS_dplay_vb

```
    DPNENUM_GROUPS = 16 (&H10)
    DPNENUM_PLAYERS = 1
End Enum
```

Constants

DPNENUM_ALL
Return the number of all groups and players.

DPNENUM_GROUP_MULTICAST
Return the number of multicast groups.

DPNENUM_GROUPS
Return the number of groups.

DPNENUM_PLAYERS
Return the number of players.

CONST_DPNERR

#Contains the error values for Microsoft® DirectPlay®. For more information, see **Error Codes**.

CONST_DPNGROUPINFOFLAGS

#Describes the settings for a group. This enumeration is used in the **lGroupFlags** member or the **DPN_GROUP_INFO** type.

```
Enum CONST_DPNGROUPINFOFLAGS
    DPNGROUP_AUTODESTRUCT = 1
End Enum
```

Constants

DPNGROUP_AUTODESTRUCT
Automatically destroy the group when the group creator leaves the group.

CONST_DPNINFO

#Used to specify the type of information you set or retrieve from group and player information methods. This enumeration is used in the **lInfoFlags** member of both the **DPN_GROUP_INFO** and **DPN_PLAYER_INFO** types.

```
Enum CONST_DPNINFO
    DPNINFO_DATA = 2
    DPNINFO_NAME = 1
```

IDH_CONST_DPNERR_dplay_vb
IDH_CONST_DPNGROUPINFOFLAGS_dplay_vb
IDH_CONST_DPNINFO_dplay_vb

```
End Enum
```

Constants

DPNINFO_DATA
The information is data set for the player or group.

DPNINFO_NAME
The information is the name of the player or group.

CONST_DPNLOBBY

[#]Used with the *lFlags* parameter of the **DirectPlay8Client.RegisterLobby** method. Set one of the two values to indicate whether the application is to be registered or unregistered.

```
Enum CONST_DPNLOBBY
   DPNLOBBY_REGISTER = 1
   DPNLOBBY_UNREGISTER = 2
End Enum
```

Constants

DPNLOBBY_REGISTER
Register the application with the lobby.

DPNLOBBY_UNREGISTER
Unregister the application with the lobby.

CONST_DPNMESSAGEID

[#]Contains message identification flags that are used by some of the methods of the **DirectPlay8Event** class. For more information, see the **DirectPlay8Event.AddRemovePlayerGroup**, and **DirectPlay8Event.InfoNotify** methods.

```
Enum CONST_DPNMESSAGEID
   DPN_MSGID_ADD_PLAYER_TO_GROUP = -65528 (&HFFFF0008)
   DPN_MSGID_APPLICATION_DESC = -65526 (&HFFFF000A)
   DPN_MSGID_ASYNC_OP_COMPLETE = -65533 (&HFFFF0003)
   DPN_MSGID_CLIENT_INFO = -65524 (&HFFFF000C)
   DPN_MSGID_CONNECT_COMPLETE = -65520 (&HFFFF0010)
   DPN_MSGID_CREATE_GROUP = -65530 (&HFFFF0006)
   DPN_MSGID_CREATE_PLAYER = -65529 (&HFFFF0007)
   DPN_MSGID_DESTROY_GROUP = -65528 (&HFFFF0007)
   DPN_MSGID_DESTROY_PLAYER = -65527 (&HFFFF0005)
   DPN_MSGID_ENUM_HOSTS_QUERY = -65516 (&HFFFF0014)
```

IDH_CONST_DPNLOBBY_dplay_vb
IDH_CONST_DPNMESSAGEID_dplay_vb

```
   DPN_MSGID_ENUM_HOSTS_RESPONSE = -65515 (&HFFFF0015)
   DPN_MSGID_GROUP_INFO = -65522 (&HFFFF000E)
   DPN_MSGID_HOST_MIGRATE =  -65523 (&HFFFF000D)
   DPN_MSGID_INDICATE_CONNECT = -65521 (&HFFFF000F)
   DPN_MSGID_INDICATED_CONNECT_ABORTED = -65521
(&HFFFF000F)
   DPN_MSGID_PEER_INFO = -65525 (&HFFFF000B)
   DPN_MSGID_RECEIVE = -65535 (&HFFFF0001)
   DPN_MSGID_REMOVE_PLAYER_FROM_GROUP = -65518
(&HFFFF0012)
   DPN_MSGID_RETURN_BUFFER = -65517 (&HFFFF0013)
   DPN_MSGID_SEND_COMPLETE = -65534 (&HFFFF0002)
   DPN_MSGID_SERVER_INFO = -65523 (&HFFFF000D)
   DPN_MSGID_TERMINATE_SESSION = -65514 (&HFFFF0016)
End Enum
```

CONST_DPNOPERATIONS

[#]Contains a set of flags that are used by the methods of the **DirectPlay8Client** class. See the individual method references for details.

```
Enum CONST_DPNOPERATIONS
   DPNADDPLAYERTOGROUP_SYNC =  -2147483648 (&H80000000)
   DPNCONNECT_SYNC = -2147483648 (&H80000000)
   DPNCREATEGROUP_SYNC = -2147483648 (&H80000000)
   DPNDESTROYGROUP_SYNC =  -2147483648 (&H80000000)
   DPNENUMHOSTS_SYNC =  -2147483648 (&H80000000)
   DPNHOST_OKTOQUERYFORADDRESSING = 1
   DPNOP_SYNC =  -2147483648 (&H80000000)
   DPNREMOVEPLAYERFROMGROUP_SYNC =  -2147483648
(&H80000000)
   DPNSETCLIENTINFO_SYNC =  -2147483648 (&H80000000)
   DPNSETGROUPINFO_SYNC =  -2147483648 (&H80000000)
   DPNSETPEERINFO_SYNC =  -2147483648 (&H80000000)
   DPNSETSERVERINFO_SYNC =  -2147483648 (&H80000000)
End Enum
```

CONST_DPNPLAYERGROUPFLAGS

[#]Used by the **DirectPlay8Server.SendTo** and **DirectPlay8Peer.SendTo** methods.

IDH_CONST_DPNOPERATIONS_dplay_vb
IDH_CONST_DPNPLAYERGROUPFLAGS_dplay_vb

```
Enum CONST_DPNPLAYERGROUPFLAGS
   DPNID_ALL_PLAYERS_GROUP = 0
End Enum
```

Constants

DPNID_ALL_PLAYERS_GROUP
Send a message to all players in the session.

CONST_DPNPLAYINFOFLAGS

#Used in the **lPlayerFlags** member of the **DPN_PLAYER_INFO** structure to determine whether the player is a host player or local player.

```
Enum CONST_DPNPLAYINFOFLAGS
   DPNPLAYER_HOST = 4
   DPNPLAYER_LOCAL = 2
End Enum
```

Constants

DPNPLAYER_HOST
This information is for the local player.

DPNPLAYER_LOCAL
This player is the host for the application.

CONST_DPNSENDFLAGS

#Contains flags that can be set in the **DirectPlay8Client.Send**, **DirectPlay8Server.SendTo**, and **DirectPlay8Peer.SendTo** methods to control how messages are sent.

```
Enum CONST_DPNSENDFLAGS
   DPNSEND_COMPLETEONPROCESS = 4
   DPNSEND_GUARANTEED = 8
   DPNSEND_NOCOMPLETE = 1
   DPNSEND_NOCOPY = 2
   DPNSEND_NOLOOPBACK = 32 (&H20)
   DPNSEND_NONSEQUENTIAL = 16 (&H10)
   DPNSEND_PRIORITY_HIGH = 128 (&H80)
   DPNSEND_PRIORITY_LOW = 64 (&H40)
   DPNSEND_SYNC = -2147483648 (&H80000000)
End Enum
```

IDH_CONST_DPNPLAYINFOFLAGS_dplay_vb
IDH_CONST_DPNSENDFLAGS_dplay_vb

Constants

DPNSEND_COMPLETEONPROCESS
: Call the **DirectPlay8Event.SendComplete** method in the message handler when this message has been delivered to the target and the target's message handler returns from indicating its reception. There is additional internal message overhead when this flag is set, and the message transmission process may become significantly slower. If you set this flag, DPNSEND_GUARANTEED must also be set.

DPNSEND_GUARANTEED
: Send the message by a guaranteed method of delivery.

DPNSEND_NOCOMPLETE
: Do not call the **DirectPlay8Event.SendComplete** method in the sender's message handler. There is additional internal message overhead when this flag is set, and the message sends process is significantly slower. This flag cannot be combined with DPNSEND_NOCOPY.

DPNSEND_NOCOPY
: Use the data in the *buffer()* parameter type and do not make an internal copy. This might be a more efficient method of sending data. However, it is less robust because the sender might be able to modify the message before the receiver has processed it. This flag cannot be combined with DPNSEND_NOCOMPLETE or DPNSEND_GUARANTEED.

DPNSEND_NOLOOPBACK
: Do not call the **DirectPlay8Event.Receive** method in your message handler when you are sending to yourself. This flag is useful if you are broadcasting to the entire session, or you are sending a message to a group that you are a member of.

DPNSEND_NONSEQUENTIAL
: Messages are delivered to the target application in the order that they are received. If the flag is not set, messages are delivered to the target application in the order that they are sent, which may necessitate buffering out-of-sequence messages until the missing messages arrive.

DPNSEND_PRIORITY_HIGH
: Sets the priority of the message to high.

DPNSEND_PRIORITY_LOW
: Sets the priority of the message to low.

DPNSEND_SYNC
: Process the **SendTo** request synchronously.

CONST_DPNGETSENDQUEUEINFO

#Used with the *lFlags* parameter of **DirectPlay8Client.GetSendQueueInfo**. Setting one of these flags instructs the method to only return send-queue information on the associated send queue instead of the total for all queues.

```
Enum CONST_DPNGETSENDQUEUEINFO
   DPNGETSENDQUEUEINFO_PRIORITY_HIGH = 2
   DPNGETSENDQUEUEINFO_PRIORITY_LOW = 4
   DPNGETSENDQUEUEINFO_PRIORITY_NORMAL = 1
End Enum
```

Constants

DPNGETSENDQUEUEINFO_PRIORITY_HIGH
: Return information for the high-priority queue.

DPNGETSENDQUEUEINFO_PRIORITY_LOW
: Return information for the low-priority queue.

DPNGETSENDQUEUEINFO_PRIORITY_NORMAL
: Return information for the normal-priority queue.

CONST_DPNSESSIONFLAGS

#Used in the **lFlags** member of the **DPN_APPLICATION_DESC** type to specify the type of session.

```
Enum CONST_DPNSESSIONFLAGS
   DPNSESSION_CLIENT_SERVER = 1
   DPNSESSION_MIGRATE_HOST = 4
   DPNSESSION_NODPNSVR = 64 (&H40)
   DPNSESSION_REQUIREPASSWORD = 128 (&H80)
End Enum
```

Constants

DPNSESSION_CLIENT_SERVER
: Specify that this type of session is client/server. This flag cannot be combined with DPNSESSION_MIGRATE_HOST.

DPNSESSION_MIGRATE_HOST
: Enable host migration. This flag is used in peer-to-peer sessions. This flag cannot be combined with DPNSESSION_CLIENT_SERVER.

DPNSESSION_NODPNSVR

IDH_CONST_DPNGETSENDQUEUEINFO_dplay_vb
IDH_CONST_DPNSESSIONFLAGS_dplay_vb

Specify that you do not want enumerations forwarded to your host from DPNSVR.

DPNSESSION_REQUIREPASSWORD
Specify that the session is password protected. If this flag is set, the **Password** member of the **DPN_APPLICATION_DESC** type must be a valid string.

CONST_DPNSPCAPS

#Used by the **DPN_SP_CAPS** type.

```
Enum CONST_DPNSPCAPS
   DPNSPCAPS_SUPPORTSALLADAPTERS = 4
   DPNSPCAPS_SUPPORTSBROADCAST = 2
   DPNSPCAPS_SUPPORTSDPNSRV = 1
End Enum
```

Constants

DPNSPCAPS_SUPPORTSALLADAPTERS
The service provider is supported on all the adapters present on the system.

DPNSPCAPS_SUPPORTSBROADCAST
On IP and IPX applications, the service provider has the ability to broadcast to find games if not enough addressing information is passed.

DPNSPCAPS_SUPPORTSDPNSRV
DPNSVR.EXE will provide port sharing for the given SP. Currently this flag is available on IP and IPX only.

CONST_DPNWAITTIME

#Used by the **DirectPlay8Client.EnumHosts**, and **DirectPlay8Peer.EnumHosts** methods.

```
Enum CONST_DPNWAITTIME
   INFINITE = -1 (&HFFFFFFFF)
End Enum
```

Constants

INFINITE
Wait until the enumeration is canceled.

IDH_CONST_DPNSPCAPS_dplay_vb
IDH_CONST_DPNWAITTIME_dplay_vb

CONST_DVBUFFERAGGRESSIVENESS

#Contains flags used in the **lBufferAggressiveness** member of the **DVCLIENTCONFIG** type. For most applications, this can be set to DVBUFFERAGGRESSIVENESS_DEFAULT. It can also be set to anything in the range of DVBUFFERAGGRESSIVENESS_MIN and DVBUFFERAGGRESSIVENESS_MAX. In general, the higher the value, the quicker the adaptive buffering adjusts to changing conditions. The lower the value, the slower the adaptive buffering adjusts to changing conditions.

```
Enum CONST_DVBUFFERAGGRESSIVENESS
   DVBUFFERAGGRESSIVENESS_DEFAULT = 0
   DVBUFFERAGGRESSIVENESS_MAX = 100 (&H64)
   DVBUFFERAGGRESSIVENESS_MIN = 1
End Enum
```

Constants

DVBUFFERAGGRESSIVENESS_DEFAULT
: The default buffer aggressiveness value.

DVBUFFERAGGRESSIVENESS_MAX
: The maximum buffer aggressiveness value.

DVBUFFERAGGRESSIVENESS_MIN
: The minimum buffer aggressiveness value.

CONST_DVBUFFERQUALITY

#Contains flags used in the **lBufferQuality** member of the **DVCLIENTCONFIG** type. For most applications, this should be set to DVBUFFERQUALITY_DEFAULT. It can be set to anything in the range of DVBUFFERQUALITY_MIN to DVBUFFERQUALITY_MAX. In general, the higher the value, the higher the quality of the voice but the also higher the latency. The lower the value, the lower the latency but also the lower the quality.

```
Enum CONST_DVBUFFERQUALITY
   DVBUFFERQUALITY_DEFAULT = 0
   DVBUFFERQUALITY_MAX = 100 (&H64)
   DVBUFFERQUALITY_MIN = 1
End Enum
```

Constants

DVBUFFERQUALITY_DEFAULT

IDH_CONST_DVBUFFERAGGRESSIVENESS_dplay_vb
IDH_CONST_DVBUFFERQUALITY_dplay_vb

The default buffer quality value.

DVBUFFERQUALITY_MAX
The maximum buffer quality value.

DVBUFFERQUALITY_MIN
The minimum buffer quality value.

CONST_DVCLIENTCONFIGENUM

[#]Used in the **lFlags** member of the **DVCLIENTCONFIG** type to control voice transmission behavior.

```
Enum CONST_DVCLIENTCONFIGENUM
   DVCLIENTCONFIG_AUTORECORDVOLUME = 8
   DVCLIENTCONFIG_AUTOVOICEACTIVATED = 32 (&H20)
   DVCLIENTCONFIG_ECHOSUPPRESSION = 134217728 (&H8000000)
   DVCLIENTCONFIG_MANUALVOICEACTIVATED = 4
   DVCLIENTCONFIG_MUTEGLOBAL = 16 (&H10)
   DVCLIENTCONFIG_PLAYBACKMUTE = 2
   DVCLIENTCONFIG_RECORDMUTE = 1
End Enum
```

Constants

DVCLIENTCONFIG_AUTORECORDVOLUME
Activate automatic gain control. With automatic gain control, Microsoft® DirectPlay® Voice adjusts the hardware input volume on your sound card automatically to get the best input level possible. You can determine the current input volume by looking at the **lRecordVolume** member after a call to **DirectPlayVoiceClient8.GetClientConfig**.

DVCLIENTCONFIG_AUTOVOICEACTIVATED
Place the transmission control system into automatic voice activation mode. In this mode, the sensitivity of voice activation is determined automatically by the system. The input level is adaptive, adjusting itself automatically to the input signal. For most applications this should be the setting used. This flag and the DVCLIENTCONFIG_MANUALVOICEACTIVATED flag are mutually exclusive.

DVCLIENTCONFIG_ECHOSUPPRESSION
Activate the echo suppression mode. This mode reduces echo introduced by configurations with external speakers and extremely sensitive microphones. While remote player's voices are being played back on the local speaker, the microphone is automatically muted. If the local player is transmitting, then the playback of remote player voices is buffered until local input stops. After local input stops, playback resumes.

DVCLIENTCONFIG_MANUALVOICEACTIVATED

IDH_CONST_DVCLIENTCONFIGENUM_dplay_vb

Place the transmission control system into manual voice activation mode. In this mode, transmission of voice begins when the input level passes the level specified by the **lThreshold** member of the **DVCLIENTCONFIG** type. When input levels drop below the specified level, transmission stops. This flag is mutually exclusive with the DVCLIENTCONFIG_AUTOVOICEACTIVATED flag.

DVCLIENTCONFIG_MUTEGLOBAL
Mute playback of the main sound buffer. Only sound buffers created through calls to **DirectPlayVoiceClient8.Create3DSoundBuffer** will be heard.

DVCLIENTCONFIG_PLAYBACKMUTE
Mute the playback of all DirectPlay Voice output and stop playback. This also stops decompression of incoming packets so CPU usage is reduced. Packets are effectively discarded while this flag is specified.

DVCLIENTCONFIG_RECORDMUTE
Mute the input from the microphone and stop recording. This also stops compression so CPU usage is reduced.

In addition to the preceding flags, the method of transmission is controlled by setting only one of the following flags or by not specifying either flag.

If you do not specify either DVCLIENTCONFIG_MANUALVOICEACTIVATED or DVCLIENTCONFIG_AUTOVOICEACTIVATED, the system will operate in push-to-talk mode. In push-to-talk mode, as long as there is a valid target specified the input from the microphone will be transmitted. Voice transmission stops when a NULL target is set or the current target leaves the session or is destroyed.

CONST_DVERR

#Contains the error values for Microsoft® DirectPlay® Voice. For more information, see **Error Codes**.

CONST_DVFLAGS

#Used by various Microsoft® DirectPlay® voice methods.

```
Enum CONST_DVFLAGS
   DVFLAGS_ALLOWBACK = 16 (&H10)
   DVFLAGS_NOHOSTMIGRATE = 8
   DVFLAGS_QUERYONLY = 2
   DVFLAGS_SYNC = 1
End Enum
```

Constants

DVFLAGS_ALLOWBACK

IDH_CONST_DVERR_dplay_vb
IDH_CONST_DVFLAGS_dplay_vb

Enable the **Back** button on the wizard's Welcome page. If the user clicks the **Back** button on the Welcome page, the wizard exits, and **CheckAudioSetup** returns DVERR_USERBACK.

DVFLAGS_NOHOSTMIGRATE
The voice host will not migrate regardless of session and transport settings. Use this flag when you want to shut down the voice session completely.

DVFLAGS_QUERYONLY
Audio setup is not run. Instead, the method checks the registry to see if the devices have been tested. If the devices have not been tested, the method returns DVERR_RUNSETUP. If the devices have been tested, the method returns DV_FULLDUPLEX if the devices support full duplex audio, or DV_HALFDUPLEX if the devices do not support full duplex audio.

DVFLAGS_SYNC
Do not return until the operation is completed.

CONST_DVMESSAGE

[#]Not used.

```
Enum CONST_DVMESSAGE
   DVID_ALLPLAYERS = 0
   DVID_NOTARGET = -1 (&HFFFFFFFF)
   DVID_REMAINING = -1 (&HFFFFFFFF)
   DVID_SERVERPLAYER = 1
   DVID_SYS = 0
   DVMSGID_BASE = 0
   DVMSGID_CONNECTRESULT = 8
   DVMSGID_CREATEVOICEPLAYER = 1
   DVMSGID_DELETEVOICEPLAYER = 2
   DVMSGID_DISCONNECTRESULT = 9
   DVMSGID_HOSTMIGRATED = 14
   DVMSGID_INPUTLEVEL = 12
   DVMSGID_MAXBASE = 16 (&H10)
   DVMSGID_MINBASE = 1
   DVMSGID_OUTPUTLEVEL = 13
   DVMSGID_PLAYEROUTPUTLEVEL = 16 (&H10)
   DVMSGID_PLAYERVOICESTART = 4
   DVMSGID_PLAYERVOICESTOP = 5
   DVMSGID_RECORDSTART = 6
   DVMSGID_RECORDSTOP = 7
   DVMSGID_SESSIONLOST = 3
   DVMSGID_SETTARGET = 15
   DVMSGID_STARTSESSIONRESULT = 10
   DVMSGID_STOPSESSIONRESULT = 11
```

IDH_CONST_DVMESSAGE_dplay_vb

```
End Enum
```

CONST_DVNOTIFY

#Used by the **DVCLIENTCONFIG** type.

```
Enum CONST_DVNOTIFY
   DVNOTIFYPERIOD_MINPERIOD = 20 (&H14)
End Enum
```

Constants

DVNOTIFYPERIOD_MINPERIOD
Specifies the minimum period between messages that is allowed.

CONST_DVPLAYBACKVOLUME

#Used by the **DVCLIENTCONFIG** type.

```
Enum CONST_DVPLAYBACKVOLUME
   DVPLAYBACKVOLUME_DEFAULT = 0
End Enum
```

Constants

CONST_DVPLAYBACKVOLUME
Use a default value that is appropriate for most situations.

CONST_DVSESSION

#Used in the **lFlags** member of the **DVSESSIONDESC** type to control session behavior.

```
Enum CONST_DVSESSION
   DVSESSION_NOHOSTMIGRATION = 1
   DVSESSION_SERVERCONTROLTARGET = 2
End Enum
```

Constants

DVSESSION_NOHOSTMIGRATION
The voice host will not migrate regardless of the transport settings. If this flag is not specified, the voice host will migrate if the transport supports it.

DVSESSION_SERVERCONTROLTARGET

IDH_CONST_DVNOTIFY_dplay_vb
IDH_CONST_DVPLAYBACKVOLUME_dplay_vb
IDH_CONST_DVSESSION_dplay_vb

Only the server player can control the clients' speech target. If the server player does not specify this flag, only the clients can control their speech target. This flag can be specified only in multicast and mixing sessions.

CONST_DVSESSIONTYPE

#Used in the **lSessionType** member of the **DVSESSIONDESC** type to specify the type of Microsoft® DirectPlay® Voice session to run. The DVSESSIONTYPE_PEER flag is not available in client/server sessions; all other flags are valid for all session types.

```
Type CONST_DVSESSIONTYPE
   DVSESSIONTYPE_ECHO = 4
   DVSESSIONTYPE_MIXING = 2
   DVSESSIONTYPE_MULTICAST = 3
   DVSESSIONTYPE_PEER = 1
End Type
```

Constants

DVSESSIONTYPE_ECHO
Voice messages will be echoed back to the player.

DVSESSIONTYPE_MIXING
The voice session will use a mixing server. In this mode of operation, all voice messages are sent to the server, which mixes them and then forwards a single, premixed stream to each client. This reduces the bandwidth and CPU usage on clients significantly at the cost of increased bandwidth and CPU usage on the server.

DVSESSIONTYPE_MULTICAST
Voice messages will be routed through the session host. This will save bandwidth on the clients at the expense of bandwidth usage on the server. This option is useful only if the session host has a high-speed connection.

DVSESSIONTYPE_PEER
Voice messages will be sent directly between players.

CONST_DVSOUNDEFFECT

#Used in the **lFlags** member of the **DVSOUNDDEVICECONFIG** type to control sound behavior.

```
Enum CONST_DVSOUNDEFFECT
   DVSOUNDCONFIG_AUTOSELECT = 2
   DVSOUNDCONFIG_HALFDUPLEX = 4
   DVSOUNDCONFIG_NOFOCUS = 536870912 (&H20000000)
```

IDH_CONST_DVSESSIONTYPE_dplay_vb
IDH_CONST_DVSOUNDEFFECT_dplay_vb

```
    DVSOUNDCONFIG_NORECVOLAVAILABLE = 16 (&H10)
    DVSOUNDCONFIG_NORMALMODE = 1
    DVSOUNDCONFIG_SETCONVERSIONQUALITY = 8
    DVSOUNDCONFIG_STRICTFOCUS = 1073741824 (&H40000000)
End Enum
```

Constants

DVSOUNDCONFIG_AUTOSELECT
: Tell Microsoft® DirectPlay® Voice to attempt to automatically select (or unmute) the microphone line in the mixer for the specified recording device.

DVSOUNDCONFIG_HALFDUPLEX
: Tell DirectPlay Voice to initialize itself in half-duplex mode. In half-duplex mode no recording takes place. If the initialization of the sound system fails in full-duplex mode, this flag will be set by the system.

DVSOUNDCONFIG_NOFOCUS
: Not enabled for this release.

DVSOUNDCONFIG_NORECVOLAVAILABLE
: Indicate that the recording volume cannot be modified.

DVSOUNDCONFIG_NORMALMODE
: Tell DirectPlay Voice to use Microsoft DirectSound® Normal Mode when initializing the DirectSound object. If this flag is not specified the DirectSound object is initialized with DirectSound Priority Mode. See documentation for **DirectSound8.SetCooperativeLevel** for more information.

DVSOUNDCONFIG_SETCONVERSIONQUALITY
: Enable better quality audio at the expense of higher CPU usage.

DVSOUNDCONFIG_STRICTFOCUS
: Not enabled for this release.

CONST_DVTHRESHOLD

[#]Used in the **lThreshold** member of the **DVCLIENTCONFIG** type to specify the input level used to trigger voice transmission if the DVCLIENTCONFIG_MANUALVOICEACTIVATED flag is specified in the **lFlags** member. When the flag is specified, this value can be set to anywhere in the range of DVTHRESHOLD_MIN to DVTHRESHOLD_MAX. Additionally, DVTHRESHOLD_DEFAULT can be set to use a default value.

If DVCLIENTCONFIG_MANUALVOICEACTIVATED or DVCLIENTCONFIG_AUTOVOICEACTIVATED is not specified in the **lFlags** member of this structure (indicating push-to-talk mode), this value must be set to DVTHRESHOLD_UNUSED.

```
Enum CONST_DVTHRESHOLD
    DVTHRESHOLD_DEFAULT = -1 (&HFFFFFFFF)
```

IDH_CONST_DVTHRESHOLD_dplay_vb

```
    DVTHRESHOLD_MAX = 99 (&H63)
    DVTHRESHOLD_MIN = 0
    DVTHRESHOLD_UNUSED = -2 (&HFFFFFFFE)
End Enum
```

Constants

DVTHRESHOLD_DEFAULT
Default threshold value.

DVTHRESHOLD_MAX
Maximum threshold value.

DVTHRESHOLD_MIN
Minimum threshold value.

DVTHRESHOLD_UNUSED
Must be set for push-to-talk mode.

Error Codes

This table lists the error codes that can be returned by Microsoft® DirectPlay® methods and functions. Errors are represented by negative values and cannot be combined.

For a list of the errors each method or function can raise, see the individual descriptions. Lists of error codes in the documentation are necessarily incomplete. For example, any DirectPlay method can return DPERR_OUTOFMEMORY even though the error code is not explicitly listed as a possible return value in the documentation for that method.

DPNERR_ABORTED
The operation was canceled before it could be completed.

DPNERR_ADDRESSING
The address specified is invalid.

DPNERR_ALREADYCONNECTED
The object is already connected to the session.

DPNERR_ALREADYCLOSING
An attempt to call **Close** on a session has been made more than once.

DPNERR_ALREADYDISCONNECTING
The client is already disconnecting from the session.

DPNERR_ALREADYINITIALIZED
The object has already been initialized.

DPNERR_ALREADYREGISTERED
A message handler has already been registered. You must unregister the current handler before registering a new one.

DPNERR_BUFFERTOOSMALL
The supplied buffer is not large enough to contain the requested data.

DPNERR_CANNOTCANCEL
The operation could not be canceled.

DPNERR_CANTCREATEGROUP
A new group cannot be created.

DPNERR_CANTCREATEPLAYER
A new player cannot be created.

DPNERR_CANTLAUNCHAPPLICATION
The lobby cannot launch the specified application.

DPNERR_CONNECTING
The method is in the process of connecting to the network.

DPNERR_CONNECTIONLOST
The service provider connection was reset while data was being sent.

DPNERR_DOESNOTEXIST
Requested element is not part of the address.

DPNERR_EXCEPTION
An exception occurred when processing the request.

DPNERR_GENERIC
An undefined error condition occurred.

DPNERR_GROUPNOTEMPTY
The specified group is not empty.

DPNERR_HOSTING

DPNERR_HOSTREJECTEDCONNECTION
The connection request was rejected. Check the **ReplyData** member of the **DPNMSG_CONNECT_COMPLETE** type for details.

DPNERR_HOSTTERMINATEDSESSION
The host terminated the session.

DPNERR_INCOMPLETEADDRESS
The address specified is not complete.

DPNERR_INVALIDADDRESSFORMAT
The address format is invalid.

DPNERR_INVALIDAPPLICATION
The GUID supplied for the application is invalid.

DPNERR_INVALIDCOMMAND
The command specified is invalid.

DPNERR_INVALIDFLAGS
The flags passed to this method are invalid.

DPNERR_INVALIDGROUP
The group ID is not recognized as a valid group ID for this game session.

DPNERR_INVALIDHANDLE

The handle specified is invalid.

DPNERR_INVALIDINSTANCE
The GUID for the application instance is invalid.

DPNERR_INVALIDINTERFACE
The interface parameter is invalid. This value will be returned in a connect request if the connecting player was not a client in a client/server game or a peer in a peer-to-peer game.

DPNERR_INVALIDLOCALADDRESS
The address for the local computer or adapter is invalid.

DPNERR_INVALIDOBJECT
The DirectPlay object pointer is invalid.

DPNERR_INVALIDPARAM
One or more of the parameters passed to the method are invalid.

DPNERR_INVALIDPASSWORD
An invalid password was supplied when attempting to join a session that requires a password.

DPNERR_INVALIDPLAYER
The player ID is not recognized as a valid player ID for this game session.

DPNERR_INVALIDPOINTER
Pointer specified as a parameter is invalid.

DPNERR_INVALIDPRIORITY
The specified priority is not within the range of allowed priorities, which is inclusively from 0 through 65535.

DPNERR_INVALIDSTRING
String specified as a parameter is invalid.

DPNERR_INVALIDREMOTEADDRESS
The specified remote address is invalid.

DPNERR_INVALIDURL
Specified string is not a valid DirectPlay URL.

DPNERR_INVALIDVERSION
There was an attempt to connect to an invalid version of DirectPlay.

DPNERR_NOCAPS
The communication link that DirectPlay is attempting to use is not capable of this function.

DPNERR_NOCONNECTION
No communication link was established.

DPNERR_NOHOSTPLAYER
There is currently no player acting as the host of the session.

DPNERR_NOINTERFACE
The interface is not supported.

DPNERR_NORESPONSE
There was no response from the specified target.

DPNERR_NOTALLOWED

The object is read-only; this function is not allowed on this object.

DPNERR_NOTHOST
The client attempted to connect to a nonhost computer. Additionally, this error value may be returned by a nonhost that tried to set the application description.

DPNERR_OUTOFMEMORY
There is insufficient memory to perform the requested operation.

DPNERR_PENDING
Not an error, this return indicates that an asynchronous operation has reached the point where it is successfully queued.

DPNERR_PLAYERLOST
A player has lost the connection to the session.

DPNERR_SENDTOOLARGE
The buffer was too large.

DPNERR_SESSIONFULL
The request to connect to the host or server failed because the maximum number of players allotted for the session has been reached.

DPNERR_TIMEDOUT
The operation could not complete because it has timed out.

DPNERR_UNINITIALIZED
The requested object has not been initialized.

DPNERR_UNSUPPORTED
The function or feature is not available in this implementation or on this service provider.

DPNERR_USERCANCEL
The user canceled the operation.

DV_OK
The request completed successfully.

DV_FULLDUPLEX
The sound card is capable of full-duplex operation.

DV_HALFDUPLEX
The sound card can be run only in half-duplex mode.

DVERR_BUFFERTOOSMALL
The supplied buffer is not large enough to contain the requested data.

DVERR_EXCEPTION
An exception occurred when processing the request.

DVERR_GENERIC
An undefined error condition occurred.

DVERR_INVALIDFLAGS
The flags passed to this method are invalid.

DVERR_INVALIDOBJECT
The DirectPlay object pointer is invalid.

DVERR_INVALIDPARAM

One or more of the parameters passed to the method are invalid.

DVERR_INVALIDPLAYER
The player ID is not recognized as a valid player ID for this game session.

DVERR_INVALIDGROUP
The group ID is not recognized as a valid group ID for this game session.

DVERR_INVALIDHANDLE
The handle specified is invalid.

DVERR_OUTOFMEMORY
There is insufficient memory to perform the requested operation.

DVERR_PENDING
Not an error, this return indicates that an asynchronous operation has reached the point where it is successfully queued.

DVERR_NOTSUPPORTED
The operation is not supported.

DVERR_NOINTERFACE
The specified interface is not supported. This error could indicate that you are using an earlier version of DirectX that does not expose the interface.

DVERR_SESSIONLOST
The transport has lost the connection to the session.

DVERR_NOVOICESESSION
The session specified is not a voice session.

DVERR_CONNECTIONLOST
The connection to the voice session has been lost.

DVERR_NOTINITIALIZED
The **DirectPlayVoiceClient8.Initialize** or **DirectPlayVoiceServer8.Initialize** method must be called before calling this method.

DVERR_CONNECTED
The DirectPlay Voice object is connected.

DVERR_NOTCONNECTED
The DirectPlay Voice object is not connected.

DVERR_CONNECTABORTING
The connection is being disconnected.

DVERR_NOTALLOWED
The object does not have the permission to perform this operation.

DVERR_INVALIDTARGET
The specified target is not a valid player ID or group ID for this voice session.

DVERR_TRANSPORTNOTHOST
The object is not the host of the voice session.

DVERR_COMPRESSIONNOTSUPPORTED
The specified compression type is not supported on the local computer.

DVERR_ALREADYPENDING
An asynchronous call of this type is already pending.

DVERR_ALREADYINITIALIZED
The object has already been initialized.

DVERR_SOUNDINITFAILURE
A failure was encountered initializing the sound card.

DVERR_TIMEOUT
The operation could not be performed in the specified time.

DVERR_CONNECTABORTED
The connect operation was canceled before it could be completed.

DVERR_NO3DSOUND
The local computer does not support 3-D sound.

DVERR_ALREADYBUFFERED
There is already a user buffer for the specified ID.

DVERR_NOTBUFFERED
There is no user buffer for the specified ID.

DVERR_HOSTING
The object is the host of the session.

DVERR_NOTHOSTING
The object is not the host of the session.

DVERR_INVALIDDEVICE
The specified device is invalid.

DVERR_RECORDSYSTEMERROR
An error in the recording system occurred.

DVERR_PLAYBACKSYSTEMERROR
An error in the playback system occurred.

DVERR_SENDERROR
An error occurred while sending data.

DVERR_USERCANCEL
The user canceled the operation.

DVERR_UNKNOWN
An unknown error occurred.

DVERR_RUNSETUP
The specified audio configuration has not been tested. Call the **DirectPlayVoiceTest8.CheckAudioSetup** method.

DVERR_INCOMPATIBLEVERSION
The client connected to a voice session that is incompatible with the host.

DVERR_INITIALIZED
The **Initialize** method failed because the object has already been initialized.

DVERR_INVALIDPOINTER
The pointer specified is invalid.

DVERR_NOTRANSPORT
The specified object is not a valid transport.

DVERR_NOCALLBACK

This operation cannot be performed because no callback function was specified.

DVERR_TRANSPORTNOTINIT
The specified transport is not yet initialized.

DVERR_TRANSPORTNOSESSION
The specified transport is valid but is not connected/hosting.

DVERR_TRANSPORTNOPLAYER
The specified transport is connected/hosting but no local player exists.