Table of Contents

About DirectPlay

The Microsoft® DirectPlay™ application programming interface (API) for Windows® 95 is a software interface that simplifies game access to communication services. DirectPlay provides a way for games to communicate with each other that is independent of the underlying transport, protocol, or online service.

Games are more fun if they can be played against real players, and the personal computer has richer connectivity options than any game platform in history. Instead of forcing the game developer to deal with the differences that each of these connectivity solutions represents, DirectPlay provides well defined, generalized communication capabilities. DirectPlay shields the developer from the underlying complexities of diverse connectivity implementations, freeing them to concentrate on producing a great game.

DirectPlay Architecture

DirectPlay uses a simple send/receive communications model to implement a connectivity API tailored to the needs of game play. The DirectPlay architecture is composed of two types of components: DirectPlay itself and the *Service Provider*. DirectPlay is provided by Microsoft and presents a common interface to the game. The service providers furnish medium-specific communications services as requested by DirectPlay. Anyone, including online services, can provide service providers for specialized hardware and communications media. Microsoft includes two service providers, for networking and modem support, with DirectPlay.

The DirectPlay interface hides the complexities and unique tasks required to establish an arbitrary communications link inside the DirectPlay service provider implementation. A game using DirectPlay need only concern itself with the performance of the communications medium, not whether that medium is being provided by a modem, network, or online service.

DirectPlay will dynamically bind to any DirectPlay Service Provider installed on the user's system. The game interacts with the DirectPlay object. The DirectPlay object interacts with one of the available DirectPlay service providers, and the selected service provider interacts with the transport or protocol.

Globally Unique Identifiers

Each game requires a globally unique identifier (GUID) that it uses to identify itself on the communications medium. These GUIDs (sometimes called UUIDS) can be generated on any computer that has a network card and a copy of UUIDGEN.EXE, which is provided as part of the Microsoft Win32® SDK. You create a GUID once, while developing the game, and use that GUID throughout the life of the product. There is no need to register this number with Microsoft.

Using DirectPlay

You can implement DirectPlay in your application using the following steps:

1. Request the user to select a communication medium for the game.

   Your application can identify the service providers installed on a personal computer by using the DirectPlayEnumerate function.

2. Create a DirectPlay object based on the selected provider by calling the DirectPlayCreate function and specifying the appropriate service provider GUID.

   The call to **DirectPlayCreate** causes DirectPlay to load the library for the selected service provider.

3. Request game information, including preferences, from the user. Your application can store this information in the **dwUser** fields of the DPSESSIONDESC structure.

4. Enumerate existing sessions (existing games that a user can join) by using the EnumSessions member function.

   If the user wants to start a new game, skip this step and continue with step six.

5. If the user wants to join a game enumerated by the EnumSessions member function, connect to that game by using the Open member function and specifying the DPOPEN_OPENSESSION flag.

6. If the user wants to start a new game, create a game by using Open and specifying the DPOPEN_CREATESESSION flag.

7. Create a player or players.

   A player's communication capabilities can be determined using GetCaps and GetPlayerCaps. Other players can be discovered by using EnumPlayers.

8. Exchange messages among players and the system by using the Send and Receive member functions.

Each player is associated with a "friendly name" and a "formal name" that the game can use for tasks such as error reporting and scoring. The game can exchange messages among players by using the unique player ID that is created with the player. The service provider, rather than DirectPlay, limits the number of players that can participate in a gaming session. In the current implementation, the number of players ranges from 16 for a modem connection to 256 for a network connection.

Most messages are defined by the game developer to address the particular needs of the game. However, some system messages are defined by DirectPlay. For example, when a player quits or a new player joins the game, the game receives a system message that provides the name of the player and the status change that has just occurred. System messages are always sent by the name server, a virtual player whose player ID is zero. System messages start with a 32-bit value that identifies the type of message. Constants that represent system messages begin with 'DPSYS_', and have a corresponding message structure that must be used to interpret them.

Broadcasting a message to all players in the game is simply a matter of sending a message to the name server (that is, to player ID zero). The players receiving a message that was broadcast in this way see the message as having come from the player who sent it, not from the name server.

DirectPlay does not attempt to provide a general approach for game synchronization; to do so would necessarily imposes limitations on the game-playing paradigm. However, the system includes some services that are designed to help you with these tasks. For example, you can specify a notification event when your application creates a player and then use the Microsoft Win32® function **WaitForSingleObject** to find out whether there is a message pending for that player.

Session Management

A DirectPlay "session" is an instance of a game. An application uses DirectPlay's session-management functions to open or close a communications channel, save a session in the registry, or enumerate past sessions that have been saved in the registry. A game either creates a new session or enumerates existing or previous sessions and finds one to connect to. If a game has saved a session, it could enumerate previous sessions and perhaps reconnect to the saved session. (This is a particularly appropriate scenario in a modem environment, where a saved session would include phone numbers.) Not all DirectPlay service providers will support the saving of sessions, however, and this functionality is currently only implemented for modem connections.

The Open member function is used to create a new session or connect to an existing or saved session. A session is described by its corresponding DPSESSIONDESC structure. This structure contains game-specific values and session particulars such as the name of the session, an optional password for the session, and the number of players to be allowed in the session. After opening a session, your application can call the GetCaps member function to retrieve the speed of the communications link. To save a record of the session in the registry, call the SaveSession member function. For a modem connection, you can save the current session and later enumerate all of the saved sessions by calling EnumSessions specifying the DPENUMSESSIONS_PREVIOUS flag. Opening one of these saved sessions retrieves the phone number for that session and dials it. When a game session is over, it can be closed with the Close member.

Player Management

An application uses DirectPlay's player-management functions to manage the players in a game session. In addition to creating and destroying players, the application can enumerate the players or retrieve a player's communication capabilities.

The CreatePlayer and DestroyPlayer member functions create and delete players in a game session. Upon creation, each player is given a "friendly name", a "formal name", and a DirectPlay Player ID. The Player ID is used by the game and DirectPlay to route message traffic. The friendly and formal names are not used internally by DirectPlay; instead, your application can use them when communicating with the players. The GetPlayerName and SetPlayerName member functions allow your application to work with the friendly and formal names while the game is being played. The EnableNewPlayers member function enables or disables the addition of new players and can be used to prohibit the creation of new players once a game is in progress.

An application uses the EnumPlayers member function to discover what players are in a current game session and their friendly and formal names. This function is typically called immediately after a call to the Open member function that opens an existing session. The GetPlayerCaps member function retrieves information about the speed of a player's connection to the session.

Group Management

The group-management functions allow your application to create groups of players in a session. Your application can then use a single call to the Send member function to send messages to an entire group, rather than to one player at a time. Some service providers can send messages to groups more efficiently than they could send them to the individual players in the group, so in addition to simplifying player management, groups can be used to conserve communication channel bandwidth.

The CreateGroup and DestroyGroup member functions create and delete a group of players. When you create a group you assign it a friendly and formal name, just as you would when creating a player. The group is initially empty; your application uses the AddPlayerToGroup and DeletePlayerFromGroup member functions to control the membership of the group. The state of the EnableNewPlayers member function does not affect the ability to create groups.

To discover what groups exist, your application can call the EnumGroups member function. To enumerate the players in a group, call the EnumGroupPlayers member function.

Message Management

The message-management functions help your application route messages between game players. With the exception of a small number of messages that have been defined by the system, the messages can be anything your application requires, although messages should not be excessively large. Your application can use the Send member function to send a message to an individual player, to a group, or to all the players in the session, by specifying a player ID, a group ID, or zero for the destination.

To receive a message from the message queue, use the Receive member function. This function allows your application to specify whether to retrieve the first message in the queue, only the messages to a particular player, or only those from a particular player. Your application can use the GetMessageCount member function to retrieve the number of messages waiting for a given player.

Structure Summary

Note that DirectPlay structures must have their "size" fields, where present, properly set before calling DirectPlay functions or an error will result.

DPCAPS	This structure defines the capabilities of the DirectPlay object as supported by a particular service provider.
DPMSG_ADDPLAYER	Message sent when a player or group has been added to a session.
DPMSG_DELETEPLAYER	Message sent when a player or group is deleted from a session.
DPMSG_GENERIC	A generic message structure provided for message handling.
DPMSG_GROUPADD	Message sent when a player is added to a group
DPMSG_GROUPDELETE	Message sent when a player is removed from a group.
DPSESSIONDESC	This structure defines the capabilities of a DirectPlay session.

Data Structures

DPCAPS

DPSESSIONDESC

DPCAPS

Contains the capabilities of a DirectPlay object after a call to the **GetCaps** function. This structure is read-only.

Structure

```
typedef struct {
   DWORD  dwSize;
   DWORD  dwFlags;
   DWORD  dwMaxBufferSize;
   DWORD  dwMaxQueueSize;
   DWORD  dwMaxPlayers;
   DWORD  dwHundredBaud;
   DWORD  dwLatency;
} DPCAPS;
```

dwSize
Size, in bytes, of this structure. Must be initialized before the structure is used.

dwFlags

DPCAPS_GUARANTEE	Supports verification of received messages. Retransmits message, if necessary.
DPCAPS_NAMESERVER	Computer represented by calling application is the name server.
DPCAPS_NAMESERVICE	A name server is supported.

dwMaxBufferSize
Maximum buffer size for this DirectPlay object.

dwMaxQueueSize
Maximum queue size for this DirectPlay object.

dwMaxPlayers
Maximum number of players supported in a session.

dwHundredBaud
Baud rate in multiples of one hundred. For example, the value 24 specifies 2400 baud.

dwLatency
Latency estimate, in milliseconds, by service provider. If this value is zero, DirectPlay cannot provide an estimate. Accuracy for some service providers rests on application-to-application testing, taking into consideration the average message size.

DPSESSIONDESC

Contains a description of the capabilities of a DirectPlay session.

```
typedef struct {
       DWORD  dwSize;
       GUID   guidSession;
       DWORD  dwSession;
       DWORD  dwMaxPlayers;
       DWORD  dwCurrentPlayers;
       DWORD  dwFlags;
       char   szSessionName[DPSESSIONNAMELEN];
       char   szUserField[DPUSERRESERVED];
       DWORD  dwReserved1;
       char   szPassword[DPPASSWORDLEN];
       DWORD  dwReserved2;
       DWORD  dwUser1;
       DWORD  dwUser1;
       DWORD  dwUser2;
       DWORD  dwUser3;
       DWORD  dwUser4;
} DPSESSIONDESC;


typedef DPSESSIONDESC FAR *LPDPSESSIONDESC;
```

dwSize
Size, in bytes, of this structure. Must be initialized before the structure is used.

guidSession
Globally unique identifier (GUID) for the game. It uniquely identifies the game so that DirectPlay connects only to other machines playing the same game.

dwSession
Session identifier of the session that has been created or opened.

dwMaxPlayers
Maximum number of players and groups allowed in this session. This member is ignored if the application is not creating a new session.

dwCurrentPlayers
Current players and groups in the session.

dwFlags
One of the following flags:

DPOPEN_CREATESESSION	Create a new session described by the DPSESSIONDESC structure.
DPOPEN_OPENSESSION	Open the session identified by dwSession.

szSessionName
String containing the name of the session.

szUserField
String containing user data.

dwReserved1, dwReserved2
Reserved for future use.

szPassword
String containing the optional password required to join this session.

dwUser1, dwUser2, dwUser3, dwUser4

User-specific data for the game or session.

System Messages

Messages returned by **Receive** from *PlayerID* 0 are system messages. All system messages begin with a DWORD dwType. Most programmers cast the buffer returned by **Receive** to a generic message (DPMSG_GENERIC) and switch on the dwType element, which will have a value equal to one of the DPSYS_* messages (DPSYS_ADDPLAYER, etc.).

Your application should be prepared to handle the following system messages:

Value of dwType	Message Structure	Cause
DPSYS_ADDPLAYER	DPMSG_ADDPLAYER	A player or group has been added to the session. **dwPlayerType** indicates whether it is a player or a group.
DPSYS_ADDPLAYERTOGROUP	DPMSG_GROUPADD	An existing player has been added to an existing group.
DPSYS_DELETEGROUP	DPMSG_DELETEPLAYER	A group has been deleted from the session.
DPSYS_DELETEPLAYER	DPMSG_DELETEPLAYER	A player has been deleted from the session.
DPSYS_DELETEPLAYERFROMGRP	DPMSG_GROUPDELETE	A player has been removed from a group.
DPSYS_SESSIONLOST	DPMSG_GENERIC	The connection has been lost.

See:

DPMSG_ADDPLAYER
DPMSG_DELETEPLAYER
DPMSG_GENERIC
DPMSG_GROUPADD
DPMSG_GROUPDELETE

DPMSG_ADDPLAYER

Contains information for the DPSYS_ADDPLAYER system message. The system sends this message when players and groups are added to a session.

```
typedef struct{
    DWORD dwType;
    DWORD dwPlayerType;
    DPID  dpId;
    char  szLongName[DPLONGNAMELEN];
    char  szShortName[DPSHORTNAMELEN];
    DWORD dwCurrentPlayers;
} DPMSG_ADDPLAYER;
```

dwType
Identifies the message.

dwPlayerType
Flag indicating whether a player or a group was added. TRUE indicates a player was added; FALSE indicates a group was added.

dpID
Player or group identifier.

szLongName
Formal name for player or group.

chShortName
Friendly name for player or group.

dwCurrentPlayers
Number of players in the session.

DPMSG_DELETEPLAYER

Contains information for the DPSYS_DELETEPLAYER and DPSYS_DELETEGROUP system messages. The system sends these messages when players and groups are deleted from a session.

```
typedef struct{
   DWORD               dwType;
   DPID                dpId;
} DPMSG_DELETEPLAYER;
```

dwType
Identifies the message.

dpId
DirectPlay ID of player or group that was deleted.

DPMSG_GENERIC

This structure is provided for message processing; first cast the unknown message to the DPMSG_GENERIC type, then perform further processing based on the value of dwType. Note that one system message, DPSYS_SESSIONLOST, also uses this structure.

```
typedef struct{
   DWORD              dwType;

} DPMSG_GENERIC;
```

dwType
Identifies the message.

DPMSG_GROUPADD

Contains information for the DPSYS_ADDPLAYERTOGROUP and DPSYS_DELETEPLAYERFROMGRP system messages. The system sends these messages when players are added to and deleted from a group.

```
typedef struct{
       DWORD    dwType;
       DPID     dpIdGroup;
       DPID     dpIdPlayer;
} DPMSG_GROUPADD;
```

dwType
Identifies the message.

dpIdGroup
DirectPlay ID of the group to which the player was added or deleted.

dpIdPlayer
DirectPlay ID of the player that was added to or deleted from the specified group.

DPMSG_GROUPDELETE

Contains information for the DPSYS_DELETEPLAYERFROMGRP message. For a description of the structure members, see DPMSG_GROUPADD.

```
typedef DPMSG_GROUPADD DMSG_GROUPDELETE;
```

DP_OK

The OK message indicates success and is returned when any DirectDraw related member has performed the action requested of it.

DP_OK

Request completed successfully.

DirectPlay Error Return Codes

Errors are represented by negative values and cannot be combined. This table lists the failures that can be returned by all DirectPlay members. See the individual member descriptions for a list of the error codes each one is capable of returning.

DPERR_ACCESSDENIED
The session is full or an incorrect password was supplied.

DPERR_ACTIVEPLAYERS
Cannot perform the requested operation because there are existing active players.

DPERR_ALREADYINITIALIZED
This object is already initialized.

DPERR_BUFFERTOOSMALL
The supplied buffer was not large enough to contain the requested data.

DPERR_BUSY
The DirectPlay message queue is full.

DPERR_CANTADDPLAYER
The player could not be added to the session.

DPERR_CANTCREATEPLAYER
Can't create a new player.

DPERR_CAPSNOTAVAILABLEYET
The capabilities of the DirectPlay object have not been determined yet. This error will occur if the DirectPlay object is implemented on a connectivity solution that requires polling to determine available bandwidth and latency.

DPERR_EXCEPTION
An exception occurred when processing the request.

DPERR_GENERIC
Undefined error condition.

DPERR_INVALIDFLAGS
The flags passed to this function are invalid.

DPERR_INVALIDOBJECT
The DirectPlay object pointer is invalid.

DPERR_INVALIDPARAMS
One or more of the parameters passed to the function are invalid.

DPERR_INVALIDPLAYER
The player ID is not recognized as a valid player ID for this game session.

DPERR_NOCAPS
The communication link underneath DirectPlay is not capable of this function.

DPERR_NOCONNECTION
No communication link was established.

DPERR_NOMESSAGES
There are no messages to be received.

DPERR_NONAMESERVERFOUND
No name server could be found or created. A name server must exist in order to create a player.

DPERR_NOPLAYERS
There are no active players in the session.

DPERR_NOSESSIONS
There are no existing sessions for this game.

DPERR_OUTOFMEMORY
Insufficient memory to perform requested operation.

DPERR_TIMEOUT
The operation could not be completed in the specified time.

DPERR_UNAVAILABLE

The requested service provider or session is not available.

DPERR_UNSUPPORTED

The function is not available in this implementation.

DPERR_USERCANCEL

The user cancelled the connection process during a call to Open.

APIs

The DirectPlay APIs are used to initiate communication through the DirectPlay interface. The first API, **DirectPlayCreate**, is used to instantiate a DirectPlay object for a particular service provider. The second one, **DirectPlayEnumerate**, is used to obtain a list of all the DirectPlay service providers installed on the system. This is the mechanism DirectPlay uses to support multiple communication transports and protocols. To utilize protocol-independent communication services, the application need only select a specific service provider and instantiate it.

See:

DirectPlayCreate

DirectPlayEnumerate

DirectPlayCreate

This API is used to create an instance of a DirectPlay object. It attempts to initialize a DirectPlay object and sets a pointer to it if it was successful. Calling the DirectPlay member DirectPlayEnumerate immediately before initialization is advised to determine what types of service providers are available.

```
HRESULT DirectPlayCreate(
    LPGUID lpGUID,
    LPDIRECTPLAY FAR *lplpDP,
    IUnknown FAR *pUnkOuter )
```

Parameters

lpGUID
Points to the **GUID** representing the driver that should be created.

lplpDP
Points to a pointer to be initialized with a valid DirectPlay pointer if the call succeeds.

pUnkOuter
Pointer to the containing IUnkown. This parameter is provided for future compability with COM aggregation features. Presently, however, **DirectPlayCreate** will return an error if it is anything but NULL.

Return Values

DP_OK	**DPERR_GENERIC**
DPERR_EXCEPTION	**DPERR_UNAVAILABLE**

See Also

DirectPlayEnumerate

DirectPlayEnumerate

Enumerate the DirectPlay service providers installed on the system.

```
HRESULT DirectPlayEnumerate(
    LPDPENUMCALLBACK lpCallback,
    LPVOID lpContext )
```

Parameters

lpCallback

Points to a callback function that will be called with a description of each DirectPlay service provider interface installed in the system.

```
BOOL Callback(
    LPGUID lpGUID,
    LPSTR lpDriverDescription,
    DWORD dwMajorVersion,
    DWORD dwMinorVersion,
    LPVOID lpContext)
```

lpSPGUID

Pointer to the unique identifier of the DirectPlay service provider driver.

lpDriverDescription

Pointer to a string containing the driver description.

dwMajorVersion

Major version number of the driver.

dwMinorVersion

Minor version number of the driver.

lpContext

Pointer to a caller-defined context.

Return Value

TRUE	**Continue the enumeration**
FALSE	**Stop the enumeration**

lpContext

Points to a caller-defined context that will be passed to the enumeration callback each time it is called.

Return Values

DP_OK	**DPERR_EXCEPTION**
DPERR_GENERIC	

Member Summary

Session management

Close	Closes a communication channel.
EnumSessions	Enumerates all of the sessions connected to a specified DirectPlay object.
GetCaps	Retrieves the capabilities of the specified DirectPlay object.
Open	Establishes a new gaming session or connects to an existing one.
SaveSession	Saves the current session in the registry.

Player management

CreatePlayer	Creates a player for a session.
DestroyPlayer	Deletes a player from a sessions.
EnableNewPlayers	Enables or disables the addition of new players or groups.
EnumPlayers	Enumerates all of the players in a specified session.
GetPlayerCaps	Retrieves the capabilities of a player's connection.
GetPlayerName	Retrieves a player's friendly and formal names.
SetPlayerName	Changes the player's friendly and formal names.

Group management

AddPlayerToGroup	Adds a player to an existing group.
CreateGroup	Creates an empty group of players for a session.
DeletePlayerFromGroup	Deletes a player from a group.
DestroyGroup	Destroys a group of players for a session.
EnumGroupPlayers	Enumerates the players in a group.
EnumGroups	Enumerates all of the groups associated with a specified session.

Message management

GetMessageCount	Retrieves the number of messages waiting for a player.
Receive	Retrieves messages that have been sent to a player.
Send	Sends messages to other players or all of the players in a session.

DirectPlay Members

AddPlayerToGroup

AddRef
Close
CreateGroup
CreatePlayer
DeletePlayerFromGroup
DestroyGroup
DestroyPlayer
EnableNewPlayers
EnumGroups
EnumGroupPlayers
EnumPlayers
EnumSessions
GetCaps
GetMessageCount
GetPlayerCaps
GetPlayerName
Initialize
Open
QueryInterface
Receive
Release
SaveSession
Send
SetPlayerName

AddPlayerToGroup

This member adds an existing player to an existing group. A DPMSG_GROUPADD system message is automatically generated to inform the other players that this has occured. Groups cannot be added to other groups. Players can be members of multiple groups. **AddPlayerToGroup** will generate an error if the player is already a member of the group.

```
HRESULT AddPlayerToGroup(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidGroup,
    DPID pidPlayer)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidGroup
Player ID of the group to be augmented.

pidPlayer
Player ID of the player to be added to group.

Return Values

DP_OK	**DPERR_GENERIC**
DPERR_INVALIDPLAYER	**DPERR_INVALIDOBJECT**

See Also

CreateGroup, **DeletePlayerFromGroup**, **DPMSG_GROUPADD**

AddRef

This member is part of the **IUnknown** interface inherited by DirectPlay. **AddRef** is used to increase the reference count of the DirectPlay object. When the DirectPlay object is initially created, its reference count is set to one. Each time a new application binds to the DirectPlay object, or a previously bound application binds to a different **COM** interface of the DirectPlay object, the reference count is increased by one. The DirectPlay object deallocates itself when its reference count goes to zero. The Release member is used to notify the DirectPlay object that an application is no longer bound to it.

```
ULONG AddRef(
    LPDIRECTPLAY lpDirectPlay)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

Initialize, **QueryInterface**, **Release**

Close

This function closes a previously opened communication channel. All locally created players will be destroyed, with corresponding DPMSG_DELETEPLAYER system messages sent to other session participants.

```
HRESULT Close(
    LPDIRECTPLAY lpDirectPlay)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS

See Also

DestroyPlayer, **DPMSG_DELETEPLAYER**

CreateGroup

This member is used to create a group of players for a session. A player ID representing the new group will be returned to the caller. Messages sent to a player ID designating a group will be sent to all members of the group. The **GroupFriendlyName** and **GroupFormalName** fields are provided for human use only; they are not used internally and need not be unique. Player IDs assigned by DirectPlay will always be unique within a session. Note that groups count as players in the session player count -- if you have a four player game with four existing players, calls to **CreateGroup** will fail. The state of EnableNewPlayers does not affect the ability to create groups.

This member, upon successful completion, sends a DPMSG_ADDPLAYER system message to all of the other players in the game announcing that a new group has been created.

```
HRESULT CreateGroup(
    LPDIRECTPLAY lpDirectPlay,
    LPDPID lppidID,
    LPSTR lpGroupFriendlyName,
    LPSTR lpGroupFormalName)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lppidID
Points to the **DPID** that will hold the DirectPlay player identification.

lpGroupFriendlyName
Points to the zero terminated string that contains the friendly name of the group.

lpGroupName
Points to the zero terminated string that contains the formal name of the group.

Return Values

DP_OK	**DPERR_CANTADDPLAYER**
DPERR_INVALIDOBJECT	**DPERR_INVALIDPARAMS**
DPERR_OUTOFMEMORY	

See Also

DestroyGroup, DPMSG_ADDPLAYER, EnableNewPlayers, EnumGroups, EnumGroupPlayers

CreatePlayer

This member is used to create a player for the current game session. A single process can have multiple players which can communicate through a DirectPlay object with any number of other players running on multiple computers. The Player ID returned to the caller will be used internally to direct the player's message traffic and manage the player. The **PlayerFriendlyName** and **PlayerFormalName** fields are provided for human use only, they are not used internally and need not be unique. Player IDs assigned by DirectPlay will always be unique within the session.

This function, upon successful completion, sends a DPMSG_ADDPLAYER system message to all of the other players in the game session announcing that a new player has joined the session. The newly created player can use the EnumPlayers member to find out who else is in the game session.

It is highly recommended that an application provide a non-NULL **lpEvent** and use this event for synchronization. After the creation of a player, use **WaitForSingleObject**(*lpEvent, dwTimeout = 0) to determine if a player has messages (the return value will be WAIT_TIMEOUT if there aren't any waiting messages) or use a different timeout to wait for a message to come in. It is inefficient to spin on Receive.

```
HRESULT CreatePlayer(
    LPDIRECTPLAY lpDirectPlay,
    LPDPID lppidID,
    LPSTR lpPlayerFriendlyName,
    LPSTR lpPlayerFormalName,
    LPHANDLE lpEvent)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lppidID
Points to the **DPID** that will hold the DirectPlay player ID.

lpPlayerFriendlyName
Points to the zero terminated string that contains the friendly name of the player.

lpPlayerName
Points to the zero terminated string that contains the formal name of the player.

lpEvent
Pointer to an event which will be triggered when a message addressed to this player is received.

Return Values

DP_OK	**DPERR_CANTCREATEPLAYER**
DPERR_NOCONNECTION	**DPERR_CANTADDPLAYER**
DPERR_INVALIDOBJECT	**DPERR_INVALIDPARAMS**
DPERR_GENERIC	

See Also

DestroyPlayer, **DPMSG_ADDPLAYER**, **EnumPlayers**, **Receive**

DeletePlayerFromGroup

This member removes a player from a group. A DPMSG_GROUPDELETE system message is automatically generated to inform the other players of the change.

```
HRESULT DeletePlayerFromGroup(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidGroup,
    DPID pidPlayer)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidGroup
Player ID of the group to be adjusted.

pidPlayer
Player ID of the player to be removed from group.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPLAYER

See Also

AddPlayerToGroup, **DPMSG_GROUPDELETE**

DestroyGroup

This member deletes a group from the session. The Player ID belonging to the group will not be reused during the current session. It is not necessary to empty a group before deleting it. The individual players belonging to the group are not destroyed, however they will be notified by a DPMSG_DELETEPLAYER system message that the group has been removed from the session.

```
HRESULT DestroyGroup(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
The player id of the group that is being removed from the game.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPLAYER

See Also

CreateGroup, **DPMSG_DELETEPLAYER**

DestroyPlayer

This member deletes a player from the game session, removes any pending messages destined for that player from the message queue, and removes the player from any groups to which it belonged. The Player ID will not be reused during the current session. Calling this member automatically sends a DPMSG_DELETEPLAYER system message to all other players, informing them that this player has been removed from the session.

```
HRESULT DestroyPlayer(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
The Player ID of the player that is being removed from the game.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPLAYER

See Also

CreatePlayer, **DPMSG_DELETEPLAYER**

EnableNewPlayers

This member can be used to enable or disable the creation of new players. It doe not affect the ability to create groups. Normally, new players and groups can be added to a session until the session's player limit has been reached. This member can be used to override this behavior if, for example, a session is "in progress" and new players are not desired. EnumSessions will not enumerate sessions where **EnableNewPlayers** has been set to false unless the DPENUMSESSIONS_ALL flag is used.

```
HRESULT EnableNewPlayers(
    LPDIRECTPLAY lpDirectPlay,
    BOOL bEnable)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

bEnable
If TRUE (the default condition for a session), new players can be created (assuming the session has not reached its maximum capacity). If FALSE, any attempt to create a new player will return an error.

Return Values

DP_OK **DPERR_INVALIDOBJECT**

See Also

CreatePlayer, **CreateGroup**, **EnumSessions**

EnumGroups

This function is used to enumerate the groups in a session. By default, the member will enumerate using the local player list for the current session. The DPENUMPLAYERS_SESSION flag can be used, along with a session ID, to request a session's name server to provide its list for enumeration. ***EnumGroups*** *cannot be called from within an* EnumSessions *enumeration.* Furthermore, use of the DPENUMPLAYERS_SESSION flag with this function must occur *after a call to EnumSessions and before any calls to* Close *or* Open*.*

```
HRESULT EnumGroups(
    LPDIRECTPLAY lpDirectPlay,
    DWORD dwSessionID,
    LPDPENUMPLAYERSCALLBACK lpEnumCallback,
    LPVOID pContext,
    DWORD dwFlags)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

dwSessionID
The DirectPlay session of interest. Not used unless the DPENUMPLAYERS_SESSION flag is specified.

lpEnumCallback
Points to the function which will be called for every group in the session.

```
BOOL EnumCallback(
    DPID pidID,
    LPSTR lpFriendlyName,
    LPSTR lpFormalName,
    DWORD dwFlags,
    LPVOID lpContext)
```

pidID
The Player ID of the group being enumerated.

lpFriendlyName
Pointer to a string containing the group's friendly name.

lpFormalName
Pointer to a string containing the group's formal name.

dwFlags
DPENUMPLAYERS_LOCAL
DPENUMPLAYERS_REMOTE
DPENUMPLAYERS_GROUP

lpContext
Pointer to a caller-defined context.

Return Value

TRUE	**Continue the enumeration**
FALSE	**Stop the enumeration**

dwFlags

DPENUMPLAYERS_SESSION
Request the name server for the specified session to supply its group list.

LPVOID lpContext
Pointer to a caller-defined context that is passed to each enumeration callback.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_UNSUPPORTED**

See Also

CreatePlayer, **DestroyPlayer**

EnumGroupPlayers

This function is used to enumerate all of the members of a particular group existing in the current session.

```
HRESULT EnumGroupPlayers(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidGroupPID,
    LPDPENUMPLAYERSCALLBACK lpEnumCallback,
    LPVOID pContext,
    DWORD dwFlags)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidGroupID
The DirectPlay ID of the group to be enumerated.

lpEnumCallback
Points to the function which will be called for every player in the group.

```
BOOL EnumCallback(
    DPID pidID,
    LPSTR lpFriendlyName,
    LPSTR lpFormalName,
    DWORD dwFlags,
    LPVOID lpContext)
```

pidID
The Player ID of the player being enumerated.

lpFriendlyName
Pointer to a string containing the players's friendly name.

lpFormalName
Pointer to a string containing the players's formal name.

dwFlags
DPENUMPLAYERS_LOCAL
DPENUMPLAYERS_REMOTE
DPENUMPLAYERS_GROUP

lpContext
Pointer to a caller-defined context.

Return Value

TRUE	**Continue the enumeration**
FALSE	**Stop the enumeration**

lpContext
Pointer to a caller-defined context that is passed to each enumeration callback.

dwFlags
Not used at this time.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDFLAGS	**DPERR_EXCEPTION**
DPERR_INVALIDPLAYER	

See Also

CreatePlayer, **DestroyPlayer**

EnumPlayers

This function is used to enumerate the players in a session. Groups can also be included in the enumeration by using the DPENUMPLAYERS_GROUP flag. By default, the member will enumerate using the local player list for the current session. This member is often called immediately after the DirectPlay object is opened. **EnumPlayers** may be called after the EnumSessions member to obtain a list of the players in a particular session by using the DPENUMPLAYERS_SESSION flag and the session id returned from **EnumSessions**. Note, however, *it cannot be called from within an **EnumSessions** enumeration.* The use of the DPENUMPLAYERS_SESSION flag with this function must occur *after a call to EnumSessions and before any calls to* Close *or* Open.

```
HRESULT EnumPlayers(
    LPDIRECTPLAY lpDirectPlay,
    DWORD dwSessionId,
    LPDPENUMPLAYERSCALLBACK lpEnumCallback,
    LPVOID pContext,
    DWORD dwFlags)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

dwSessionID
The DirectPlay session of interest. Not used unless the DPENUMPLAYERS_SESSION flag is specified.

lpEnumCallback
Points to the function which will be called for every player in the session.

```
BOOL EnumCallback(
    DPID pidID,
    LPSTR lpFriendlyName,
    LPSTR lpFormalName,
    DWORD dwFlags,
    LPVOID lpContext)
```

pidID
The Player ID of the player being enumerated.

lpFriendlyName
Pointer to a string containing the players's friendly name.

lpFormalName
Pointer to a string containing the players's formal name.

dwFlags
DPENUMPLAYERS_LOCAL
DPENUMPLAYERS_REMOTE
DPENUMPLAYERS_GROUP

lpContext
Pointer to a caller-defined context.

Return Value

TRUE	**Continue the enumeration**
FALSE	**Stop the enumeration**

lpContext
Pointer to a caller-defined context that is passed to each enumeration callback.

dwFlags

DPENUMPLAYERS_GROUP
Include groups in the enumeration of players.

DPENUMPLAYERS_PREVIOUS

Enumerate players previously stored in the registry. Not yet supported.

DPENUMPLAYERS_SESSION

Request the name server for the specified session to supply its group list.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_GENERIC	**DPERR_UNSUPPORTED**
DPERR_EXCEPTION	

See Also

CreatePlayer, **DestroyPlayer**, **EnumSessions**

EnumSessions

This member is used to enumerate the game sessions connected to this DirectPlay object. **EnumSessions** is usually called immediately after the DirectPlay object is instantiated -- it cannot be called while connected to a session or after a game has created a session. **EnumSessions** works by broadcasting an enumeration request and collecting replies from DirectPlay objects that respond. The amount of time DirectPlay spends listening for these replies is controlled by the **dwTimeout** parameter. When this time interval has expired, your callback will be notified using the DPESC_TIMEDOUT flag, and a NULL will be passed for **lpDPSGameDesc**. At this point, you may choose to continue the enumeration by setting ***lpdwTimeOut** to a new value and returning TRUE, or returning FALSE to cancel the enumeration.

```
HRESULT EnumSessions(
    LPDIRECTPLAY lpDirectPlay,
    LPDPSESSIONDESC lpSDesc,
    DWORD dwTimeout,
    LPDPENUMSESSIONSCALLBACK lpEnumCallback,
    LPVOID lpvContext,
    DWORD dwFlags)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lpSDesc
Points to a DPSESSIONDESC structure describing the sessions to be enumerated. If a list of all connected sessions, regardless of GUID, is desired, then the **guidSession** field should be set to NULL. If the DPENUMSESSIONS_AVAILABLE flag is going to be used with a password, then **szPassword** should be set accordingly.

dwTimeout
A timeout value in milliseconds. This value is the *total* amount of time that DirectPlay will allow for the enumeration to complete (not the time between enumerations).

lpEnumCallback
Points to the function which will be called for each DirectPlay session responding.

```
BOOL EnumCallback(
    LPDPSESSIONDESC lpDPSGameDesc,
    LPVOID lpContext,
    LPDWORD lpdwTimeOut,
    DWORD dwFlags)
```

lpDPSGameDesc
Pointer to a DPSESSIONDESC structure describing the enumerated session. Will be set to NULL if the enumeration has timed out.

lpContext
Pointer to a caller-defined context.

lpdwTimeOut
Pointer to a DWORD containing the current timeout value. This can be reset if you feel that some sessions have yet to respond.

dwFlags

DPESC_TIMEDOUT
The enumeration has timed out. Reset ***lpdwTimeOut** and return TRUE to continue, or FALSE to stop the enumeration.

Return Value

TRUE	**Continue the enumeration**
FALSE	**Stop the enumeration**

lpContext

Points to a user-defined context that is passed to each enumeration callback.

dwFlags

DPENUMSESSIONS_AVAILABLE

Enumerate all sessions with a matching password (if provided), open player slots, and EnableNewPlayers set to TRUE.

DPENUMSESSIONS_PREVIOUS

Enumerate sessions previously stored in the registry.

DPENUMSESSIONS_ALL

Enumerate all active sessions connected to this DirectPlay object, regardless of their occupancy, passwords, or **EnableNewPlayers** status.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS

See Also

DPSESSIONDESC, **EnableNewPlayers**

GetCaps

This member returns the capabilities of this DirectPlay object.

```
HRESULT GetCaps(
    LPDIRECTPLAY lpDirectPlay,
    LPDPCAPS lpDPCaps)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lpDPCaps
Points to a DPCAPS structure which will be filled in with the capabilities of the DirectPlay object.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS

See Also

GetPlayerCaps, **DPCAPS**

GetMessageCount

This member returns the number of messages queued for a specific local player.

```
HRESULT GetMessageCount(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID,
    LPDWORD lpdwCount)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
Player ID that the message count is being requested for. The player must be local.

lpdwCount
Points to a **DWORD** that will be set to the message count.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_INVALIDPLAYER**

See Also

Receive

GetPlayerCaps

This member returns the capabilities of this player's connection through the DirectPlay object. This member is needed because communicating with some players maybe slower than communicating with others.

```
HRESULT GetPlayerCaps(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID
    LPDPCAPS lpDPPlayersCaps)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
Player ID that the communication capabilities are being requested for.

lpDPPlayerCaps
Points to a DPCAPS structure which will be filled in with the communication capabilities of the specified player on this DirectPlay object.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_INVALIDPLAYER**

See Also

GetCaps

GetPlayerName

This member returns the player's friendly and formal names. If just one of the names is required, the other pair of pointers can be set to NULL. If the length of the names needs to be determined, the pointers to the lengths must be valid, but they can either point to zeros or the pointers to the friendly and formal names should be NULL.

If the supplied buffer is not long enough to hold one of the names, an error code will be returned and the corresponding buffer length will be adjusted to be the size of the buffer needed.

```
HRESULT GetPlayerName(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID,
    LPSTR lpPlayerFriendlyName,
    LPDWORD lpdwFriendlyNameLength,
    LPSTR lpPlayerName,
    LPDWORD lpdwPlayerNameLength)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
The Player ID for which the player names are being requested.

lpPlayerFriendlyName
Points to the buffer that should be filled in with the players friendly name. Can be set to NULL if only looking for the size of the friendly name or if only looking for the formal name.

lpdwFriendlyNameLength
Points to a **DWORD** that either contains the length of the buffer pointed to by **lpPlayerFriendlyName** or will be filled in with the length needed for the buffer. Can be set to NULL if only interested in the formal name.

lpPlayerFormalName
Points to the buffer that should be filled in with the players formal name. Can be set to NULL if only looking for the size of the formal name or if only looking for the friendly name.

lpdwFormalNameLength
Points to a **DWORD** that either contains the length of the buffer pointed to by **lpPlayerFormalName** or will be filled in with the length needed for the buffer. Can be set to NULL if only interested in the friendly name.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_BUFFERTOOSMALL**
DPERR_INVALIDPLAYER	

See Also

SetPlayerName

Initialize

This member is provided for compliance with the Common Object Model (COM) protocol. Since the DirectPlay object is initialized when it is created, calling this member will always result in the DDERR_ALREADYINITIALIZED return value.

```
HRESULT Initialize(
    LPDIRECTPLAY lpDirectPlay,
    GUID FAR *lpGUID)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lpGUID
Points to the GUID used as the interface identifier.

Return Values

DPERR_ALREADYINITIALIZED

See Also

AddRef, **QueryInterface**

Open

This function actually establishes the communication link. In a modem environment this would be equivalent to actually dialing the phone. This is where the user interface that is required to configure the communications protocol will be invoked with the DirectPlay object. In the case of the serial modem service provider supplied with DirectPlay, the user is prompted for dialing information. If the user cancels the dialing process, **Open** will return a USERCANCEL error. **Open** will also return an error if local players exist when it is called.

```
HRESULT Open(
    LPDPSESSIONDESC lpSDesc)
```

Parameters

lpSDesc
Points to the DPSESSIONDESC structure describing the session to be connected to or created.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_GENERIC**
DPERR_UNAVAILABLE	**DPERR_UNSUPPORTED**
DPERR_USERCANCEL	**DPERR_ACTIVEPLAYERS**

See Also

Close, **DPSESSIONDESC**

QueryInterface

This member is part of the **IUnknown** interface inherited by DirectPlay. **QueryInterface** is used to increase the reference count of the DirectPlay object. This is the member that applications use to determine whether the DirectPlay object supports additional interfaces that they may be interested in. An application can ask the DirectPlay object if it supports a particular **COM** interface and if it does, the application may begin using that interface immediately. If the application does not want to use that interface it must call Release to free it. This member allows DirectPlay objects to be extended by Microsoft and third parties without breaking, or interfering with, each other's existing or future functionality.

```
HRESULT QueryInterface(
    LPDIRECTPLAY lpDirectPlay,
    LPVOID riid,
    LPVOID FAR* obp)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay.

riid
Points to a **UUID**. (Universally Unique Identifier)

obp
Points to a pointer that will be filled with the interface pointer if the query is successful.

Return Values

DP_OK **DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS

See Also

AddRef, **Release**

Receive

This member is used to retrieve a message from the message queue. Messages received from player ID 0 are system messages from the name server. Messages sent to the name server as a way to broadcast them to all players still appear to come from the sender. Both DPRECEIVE_TOPLAYER and DPRECEIVE_FROMPLAYER may be specified, in which case **Receive** will return whichever message is encountered first.

```
HRESULT Receive(
    LPDIRECTPLAY lpDirectPlay,
    LPDPID lppidFrom,
    LPDPID lppidTo,
    DWORD dwFlags,
    LPVOID lpvBuffer,
    LPDWORD lpdwSize)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

lppidFrom
Points to the **DPID** structure which is, or will be filled in with, the sender's **DPID**.

lppidTo
Points to the **DPID** structure which is, or will be filled in with, the receiver's **DPID**.

dwFlags

DPRECEIVE_ALL
Return the first available message. This is the default.

DPRECEIVE_TOPLAYER
Return the first message intended for the Player ID pointed to by **lppidTo**. System messages are addressed to Player 0.

DPRECEIVE_FROMPLAYER
Return the first message from the Player ID pointed to by **lppidFrom**.

DPRECEIVE_PEEK
Return a message as specified by the other flags, but do not remove it from the message queue.

lpvBuffer
Points to the message buffer. If the message buffer is not long enough to hold the message, an error will be returned and **lpdwSize** will be filled in with the size of message buffer needed.

lpdwSize
Points to the DWORD that specifies how long the message buffer is.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_BUFFERTOOSMALL**
DPERR_NOMESSAGES	**DPERR_GENERIC**

See Also

Send

Release

This member is part of the **IUnknown** interface inherited by DirectPlay. **Release** is used to decrease the reference count of the DirectPlay object. When the DirectPlay object is initially created, its reference count is set to one. Each time **Release** is called by an application, the DirectPlay object reduces the reference count by one. The DirectPlay object deallocates itself when its reference count goes to zero. The AddRef member is used to increase the reference count every time a new application binds to the DirectPlay object.

```
ULONG Release(
    LPDIRECTPLAY lpDirectPlay)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, **QueryInterface**

SaveSession

This member function saves the current session in the registry. The functionality of this member is dependent on the service provider, which will save enough transport-specific information in the registry to restore the connection. **SaveSession** is unsupported in the TCP and IPX service providers. In the serial modem service provider, **SaveSession** functions only for the client session (the one that dials), in which case it will save the phone number in the registry for future use.

```
HRESULT SaveSession(
    LPDIRECTPLAY lpDirectPlay)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

Return Values

DP_OK	**DPERR_INVALIDOBJECT**
DPERR_INVALIDPARAMS	**DPERR_GENERIC**
DPERR_OUTOFMEMORY	**DPERR_UNSUPPORTED**

See Also

EnumSessions

Send

This member function sends messages to other players, other groups of players, or all the players in the session. To send a message to another player, specify the receiving player's ID. To send a message to a group of players, send the message to the player ID assigned to the previously created group. To send messages to the entire session, the message is sent to the **DPID** of 0, which always represents the "name server". Send will either return a code as described below or the number of messages currently queued for transmission. If the internal queue fills to the limit specified by **DPCAPS.dwMaxQueueSize**, an error will be generated and the message will not be added to the queue. Note that **Send** *cannot be used* inside a DirectDraw Lock/Unlock or DirectDraw GetDC/Release DC pair.

```
HRESULT Send(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidFrom,
    DPID pidTo,
    DWORD dwFlags,
    LPVOID lpvBuffer,
    DWORD dwBuffSize)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidFrom
The **DPID** of the sending player.

pidTo
The **DPID** of the receiving player.

dwFlags
The flags indicating how the message should be sent. Note not all options may be supported by a particular service provider.

DPSEND_GUARANTEE
Send the message using a reliable method. Retry until it is received or the DirectPlay timeout occurs.

DPSEND_HIGHPRIORITY
Send the message as a HIGHPRIORITY message.

DPSEND_TRYONCE
Send the message without error detection and without retry options enabled.

lpvBuffer
Points to the message being sent.

dwBuffSize
The length of the message being sent.

Return Values

Send will either return a code as summarized below or the number of messages queued for transmission in DirectPlay's internal queue.

DP_OK	**DPERR_INVALIDPLAYER**
DPERR_INVALIDOBJECT	**DPERR_INVALIDPARAMS**
DPERR_BUSY	

See Also

Receive

SetPlayerName

This member changes the player's friendly and formal names. **SetPlayerName** cannot be used to change the names of a group.

```
HRESULT SetPlayerName(
    LPDIRECTPLAY lpDirectPlay,
    DPID pidID,
    LPSTR lpPlayerFriendlyName,
    LPSTR lpPlayerFormalName)
```

Parameters

lpDirectPlay
Points to the DirectPlay structure representing the DirectPlay object.

pidID
The player id for which the player name is being requested.

lpPlayerFriendlyName
A pointer to a string containing the player's new friendly name.

lpPlayerFormalName
A pointer to a string containing the player's new formal name.

Return Values

DP_OK **DPERR_INVALIDPLAYER**
DPERR_INVALIDOBJECT

See Also

GetPlayerName

Error Codes

DP_OK
DPERR_ACCESSDENIED
DPERR_ACTIVEPLAYERS
DPERR_ALREADYINITIALIZED
DPERR_BUFFERTOOSMALL
DPERR_BUSY
DPERR_CANTADDPLAYER
DPERR_CANTCREATEPLAYER
DPERR_CAPSNOTAVAILABLEYET
DPERR_EXCEPTION
DPERR_GENERIC
DPERR_INVALIDFLAGS
DPERR_INVALIDOBJECT
DPERR_INVALIDPARAMS
DPERR_INVALIDPLAYER
DPERR_NOCAPS
DPERR_NOCONNECTION
DPERR_NOMESSAGES
DPERR_NONAMESERVERFOUND
DPERR_NOPLAYERS
DPERR_NOSESSIONS
DPERR_OUTOFMEMORY
DPERR_TIMEOUT
DPERR_UNAVAILABLE
DPERR_UNSUPPORTED
DPERR_USERCANCEL

DP_OK

Request completed successfully.

DPERR_ACCESSDENIED

The session is full or an incorrect password was supplied.

DPERR_ACTIVEPLAYERS

Cannot perform the requested operation because there are existing active players.

DPERR_ALREADYINITIALIZED

This object is already initialized.

DPERR_BUFFERTOOSMALL

The supplied buffer was not large enough to contain the requested data.

DPERR_BUSY

The DirectPlay message queue is full.

DPERR_CANTADDPLAYER

The player could not be added to the session.

DPERR_CANTCREATEPLAYER

Can't create a new player.

DPERR_CAPSNOTAVAILABLEYET

The capabilities of the DirectPlay object have not been determined yet. This error will occur if the DirectPlay object is implemented on a connectivity solution that requires polling to determine available bandwidth and latency.

DPERR_EXCEPTION

An exception occurred when processing the request.

DPERR_GENERIC

Undefined error condition.

DPERR_INVALIDFLAGS

The flags passed to this function are invalid.

DPERR_INVALIDOBJECT

The DirectPlay object pointer is invalid.

DPERR_INVALIDPARAMS

One or more of the parameters passed to the function are invalid.

DPERR_INVALIDPLAYER

The player ID is not recognized as a valid player ID for this game session.

DPERR_NOCAPS

The communication link underneath DirectPlay is not capable of this function.

DPERR_NOCONNECTION

No communication link was established.

DPERR_NOMESSAGES

There are no messages to be received.

DPERR_NONAMESERVERFOUND

No name server could be found or created. A name server must exist in order to create a player.

DPERR_NOPLAYERS

There are no active players in the session.

DPERR_NOSESSIONS

There are no existing sessions for this game.

DPERR_OUTOFMEMORY

Insufficient memory to perform requested operation.

DPERR_TIMEOUT

The operation could not be completed in the specified time.

DPERR_UNAVAILABLE

The requested service provider or session is not available.

DPERR_UNSUPPORTED

The function is not available in this implementation.

DPERR_USERCANCEL

The user cancelled the connection process during a call to Open.