Microsoft® DirectX™ 2 Software Development Kit

C H A P T E R 4

DirectPlay

Overview

The Microsoft® DirectPlay™ application programming interface (API) for Windows® 95 is a software interface that simplifies your application's access to communication services. DirectPlay was primarily developed for game communication, but could be used for other applications that require communication 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 you, as a game developer, to deal with the differences that each of these connectivity solutions represents, DirectPlay provides well defined, generalized communication capabilities. DirectPlay shields you from the underlying complexities of diverse connectivity implementations, freeing you to concentrate on producing a great game.

DirectPlay Architecture

DirectPlay uses a simple send/receive communication 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 provider furnishes medium-specific communication services as requested by DirectPlay. Any organization, including online services, can supply service providers for specialized hardware and communications media. Microsoft includes two service providers with DirectPlay, one for networking and one for modem support.

The DirectPlay interface hides the complexities and unique tasks required to establish an arbitrary communications link inside the DirectPlay service provider implementation. When you design a game for use with DirectPlay, you need only concern yourself with the performance of the communications medium, not whether that medium is 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, 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

Implement DirectPlay in your application using the following steps:

1. Request that the user of your application select a communication medium for the game.

   You 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.

   Calling the **DirectPlayCreate** function causes DirectPlay to load the library for the service provider selected.

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

   If the user wants to start a new game, skip to step six.

4. Enumerate existing sessions (existing games that a user can join) by using the **IDirectPlay::EnumSessions** method.

5. If the user wants to join a game enumerated by the **IDirectPlay::EnumSessions** method, connect to that game by using the **IDirectPlay::Open** method and specify the DPOPEN_OPENSESSION flag.

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

7. Create a player or players.

   A player's communication capabilities can be determined using the **IDirectPlay::GetCaps** and **IDirectPlay::GetPlayerCaps** methods. Other players can be discovered by using the **IDirectPlay::EnumPlayers** method.

8. Exchange messages among players, including the name server (player ID 0), by using the **IDirectPlay::Send** and **IDirectPlay::Receive** methods.

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, not 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.

When using DirectPlay, you are able to define most messages in order to address the particular needs of the game. However, some system messages are defined by DirectPlay. For example, when a player quits or joins a game, that game receives a system message that provides the player's name and the status change that has

just occurred. System messages are always sent by the name server, a virtual player whose player ID is 0. 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 0). Players receiving a message 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 impose 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, 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. The applications you design can use DirectPlay's session-management functions to open or close a communication 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 saving sessions, however, and this functionality is currently only implemented for modem connections.

The **IDirectPlay::Open** method is used to create new sessions or to connect to existing or saved sessions. 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, you can call the **IDirectPlay::GetCaps** method to retrieve the speed of the communications link. To save a record of the session in the registry, call the **IDirectPlay::SaveSession** method. For a modem connection, you can save the current session and later enumerate all of the saved sessions by calling **IDirectPlay::EnumSessions** and 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 **IDirectPlay::Close** method.

Player Management

The applications you design can use DirectPlay's player-management methods to manage the players in a game session. In addition to creating and destroying players, you can enumerate them or retrieve their communication capabilities.

The **IDirectPlay::CreatePlayer** and **IDirectPlay::DestroyPlayer** methods 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, you can use them when communicating with the players. The **IDirectPlay::GetPlayerName** and **IDirectPlay::SetPlayerName** methods allow your application to work with the friendly and formal names while the game is being played. The **IDirectPlay::EnableNewPlayers** method 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.

You can use the **IDirectPlay::EnumPlayers** method to discover what players are in a current game session and their friendly and formal names. This function is typically called immediately after the **IDirectPlay::Open** method opens an existing session. The **IDirectPlay::GetPlayerCaps** method retrieves information about the speed of a player's connection to the session.

Group Management

The group-management methods allow your application to create groups of players in a session. You can then use a single instance of the **IDirectPlay::Send** method 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 can send them to the individual players, so, in addition to simplifying player management, groups can be used to conserve communication channel bandwidth.

The **IDirectPlay::CreateGroup** and **IDirectPlay::DestroyGroup** methods 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; you can use the **IDirectPlay::AddPlayerToGroup** and **IDirectPlay::DeletePlayerFromGroup** methods to control the membership of the group. The state of the **IDirectPlay::EnableNewPlayers** method does not affect the ability to create groups.

To discover what groups exist, you can call the **IDirectPlay::EnumGroups** method. To enumerate the players in a group, call the **IDirectPlay::EnumGroupPlayers** method.

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 defined in any way that you require. Messages should not be excessively large, however. You can use the **IDirectPlay::Send** method to send a message to an individual player, to a group, or to all the players in the session; simply specify a player ID, a group ID, or 0 as the destination.

To retrieve a message from the message queue, use the **IDirectPlay::Receive** method. This method allows you 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. You can use the **IDirectPlay::GetMessageCount** method to retrieve the number of messages waiting for a given player.

Reference

Functions

The DirectPlay functions are used to initiate communication through the DirectPlay interface. The **DirectPlayCreate** function is used to instantiate a DirectPlay object for a particular service provider. The **DirectPlayEnumerate** function 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, your application need only select a specific service provider and instantiate it.

DirectPlayCreate

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

Creates an instance of a DirectPlay object.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_EXCEPTION **DPERR_INVALIDPARAMS**

  DPERR_UNAVAILABLE

lpGUID
Address for the GUID that represents the driver to be created.

lplpDP
Address for a pointer to initialize with a valid DirectPlay pointer.

pUnkOuter
Address for the **IUnknown** interface. This parameter is provided for future compatibility with COM aggregation features. Presently, however, the **DirectPlayCreate** function will return an error if this parameter is anything but NULL.

This function attempts to initialize a DirectPlay object and sets a pointer to it if successful. It is a good idea to call the **DirectPlayEnumerate** function immediately before initialization in order to determine what types of service providers are available.

See also **DirectPlayEnumerate**

DirectPlayEnumerate

```
LT DirectPlayEnumerate(
    LPDPENUMDPCALLBACK lpEnumDPCallback, LPVOID lpContext);
```

Enumerates the DirectPlay service providers installed on the system.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_EXCEPTION **DPERR_GENERIC**

lpEnumDPCallback
Address for the **EnumDPCallback** function that will be called with a description of each DirectPlay service provider interface installed in the system.

lpContext
Address for a caller-defined structure that will be passed to the callback function each time the function is invoked.

Callback Functions

Most of the functionality of DirectPlay is provided by the methods of its Component Object Model (COM) interfaces. This section lists the callback functions that are not implemented as part of a COM interface.

EnumDPCallback

```
BOOL EnumDPCallback(LPGUID lpSPGuid,
    LPSTR lpFriendlyName, DWORD dwMajorVersion,
    DWORD dwMinorVersion, LPVOID lpContext);
```

Application-defined callback procedure for the **DirectPlayEnumerate** function.

- Returns TRUE to continue the enumeration or FALSE to stop it.

lpSPGuid
Address for the unique identifier of the DirectPlay service provider driver.

lpFriendlyName
Address for a string containing the driver description.

dwMajorVersion
Major version number of the driver.

dwMinorVersion
Minor version number of the driver.

lpContext
Address for a caller-defined context.

EnumPlayersCallback

```
BOOL EnumPlayersCallback(DPID dpID,
    LPSTR lpFriendlyName, LPSTR lpFormalName,
    DWORD dwFlags, LPVOID lpContext);
```

Application-defined callback procedure for the **IDirectPlay::EnumGroups**, **IDirectPlay::EnumGroupPlayers**, and **IDirectPlay::EnumPlayers** methods.

- Returns TRUE to continue the enumeration or FALSE to stop it.

dpID
Player ID of the group being enumerated.

lpFriendlyName
Address for the zero-terminated string that contains the friendly name of the group.

lpFormalName
Address for the zero-terminated string that contains the formal name of the group.

dwFlags
Specifies the optional control flags.

DPENUMPLAYERS_GROUP
Specifies that group names and player names should be enumerated.

DPENUMPLAYERS_LOCAL
Specifies that only local player names need to be enumerated.

DPENUMPLAYERS_REMOTE
Specifies that only remote player names need to be enumerated.

lpContext
Address for a caller-defined context.

EnumSessionsCallback

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

Application-defined callback procedure for the **IDirectPlay::EnumSessions** method.

- Returns TRUE to continue the enumeration or FALSE to stop it.

lpDPSGameDesc
Address for a **DPSESSIONDESC** structure describing the enumerated session. This parameter will be set to NULL if the enumeration has timed out.

lpContext
Address for a caller-defined context.

lpdwTimeOut
Address for a doubleword containing the current time-out value. This can be reset if you feel that some sessions have yet to respond.

dwFlags
Specifies the optional control flag.

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

IDirectPlay Interface

IDirectPlay Interface Method Groups

Applications use the methods of the **IDirectPlay** interface to create DirectPlay objects and work with system-level variables. This interface supports the following methods:

Group management	**AddPlayerToGroup**
	CreateGroup
	DeletePlayerFromGroup
	DestroyGroup
	EnumGroupPlayers
	EnumGroups
IUnknown	**AddRef**
	QueryInterface
	Release

Message management	**GetMessageCount**
	Receive
	Send
Player management	**CreatePlayer**
	DestroyPlayer
	EnableNewPlayers
	EnumPlayers
	GetPlayerCaps
	GetPlayerName
	SetPlayerName
Session management	**Close**
	EnumSessions
	GetCaps
	Open
	SaveSession

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the DirectPlay object without affecting the functionality of the original interface.

IDirectPlay::AddPlayerToGroup

```
HRESULT AddPlayerToGroup(DPID pidGroup, DPID pidPlayer);
```

Adds an existing player to an existing group.

- Returns DP_OK if successful, or one of the following error values otherwise:
  DPERR_GENERIC **DPERR_INVALIDOBJECT**
  DPERR_INVALIDPLAYER

pidGroup
ID of the group to be augmented.

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

A **DPMSG_GROUPADD** system message is automatically generated to inform other players that the specified player has been added. Groups cannot be added to

other groups, but players can be members of multiple groups. The **IDirectPlay::AddPlayerToGroup** method will generate an error if the player is already a member of the group.

See also **IDirectPlay::CreateGroup**, **IDirectPlay::DeletePlayerFromGroup**, **DPMSG_GROUPADD**

IDirectPlay::AddRef

```
ULONG AddRef();
```

Increases the reference count of the DirectPlay object by 1. This method is part of the **IUnknown** interface inherited by DirectPlay.

- Returns the new reference count of the object.

When the DirectPlay object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirectPlay::Release** method to decrease the reference count of the object by 1.

See also **IDirectPlay::Initialize**, **IDirectPlay::QueryInterface**, **IDirectPlay::Release**

IDirectPlay::Close

```
HRESULT Close();
```

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

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**

See also **IDirectPlay::DestroyPlayer**, **DPMSG_DELETEPLAYER**

IDirectPlay::CreateGroup

```
HRESULT CreateGroup(LPDPID lppidID,
    LPSTR lpGroupFriendlyName, LPSTR lpGroupFormalName);
```

Creates a group of players for a session. A player ID representing the new group will be returned to the caller.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

lppidID
Address for the DPID that will hold the DirectPlay player ID.

lpGroupFriendlyName
Address for the zero-terminated string that contains the friendly name of the group.

lpGroupFormalName
Address for the zero-terminated string that contains the formal name of the group.

Messages sent to a player ID designating this group will be sent to all members of the group. The *lpGroupFriendlyName* and *lpGroupFormalName* parameters are provided for players' convenience only; they are not used internally and do not need to be unique. However, player IDs assigned by DirectPlay will always be unique within a session. Groups are recognized as players for the session player count; if you have a four player game with four existing players, calls to **IDirectPlay::CreateGroup** will fail. The state of the **IDirectPlay::EnableNewPlayers** method does not affect your application's ability to create groups.

Upon successful completion, this method sends a **DPMSG_ADDPLAYER** system message to all of the other players in the game announcing that a new group has been created.

See also **IDirectPlay::DestroyGroup**, **DPMSG_ADDPLAYER**, **IDirectPlay::EnableNewPlayers**, **IDirectPlay::EnumGroups**, **IDirectPlay::EnumGroupPlayers**

IDirectPlay::CreatePlayer

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

Creates a player for the current game session.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

lppidID
Address for the DPID that will hold the DirectPlay player ID.

lpPlayerFriendlyName
Address for the zero-terminated string that contains the friendly name of the player.

lpPlayerFormalName
Address for the zero-terminated string that contains the formal name of the player.

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

A single process can have multiple players that 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 *lpPlayerFriendlyName* and *lpPlayerFormalName* parameters are provided for players' convenience only; they are not used internally and need not be unique. Player IDs assigned by DirectPlay will always be unique within the session.

Upon successful completion, this method 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 **IDirectPlay::EnumPlayers** method 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 creation of a player, use *WaitForSingleObject(*lpEvent, dwTimeout = 0)* from Win32 to determine if a player has messages (the return value will be WAIT_TIMEOUT if there are not any waiting messages) or use a different time-out to wait for a message to come in. It is inefficient to loop on the **IDirectPlay::Receive** method.

See also **IDirectPlay::DestroyPlayer**, **DPMSG_ADDPLAYER**, **IDirectPlay::EnumPlayers**, **IDirectPlay::Receive**

IDirectPlay::DeletePlayerFromGroup

```
HRESULT DeletePlayerFromGroup(DPID pidGroup,
    DPID pidPlayer);
```

Removes a player from a group.

- Returns DP_OK if successful, or one of the following error messages otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPLAYER**

pidGroup
Player ID of the group to be adjusted.

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

A **DPMSG_GROUPDELETE** system message is automatically generated to inform the other players of this change.

See also **IDirectPlay::AddPlayerToGroup**, **DPMSG_GROUPDELETE**

IDirectPlay::DestroyGroup

```
HRESULT DestroyGroup(DPID pidID);
```

Deletes a group from the session. The player ID belonging to this group will not be reused during the current session.

- Returns DP_OK if successful, or one of the following error messages otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPLAYER**

pidID
Player ID of the group being removed from the game.

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.

See also **IDirectPlay::CreateGroup**, **DPMSG_DELETEPLAYER**

IDirectPlay::DestroyPlayer

```
HRESULT DestroyPlayer(DPID pidID);
```

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.

- Returns DP_OK if successful, or one of the following error messages otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPLAYER**

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

Calling this method automatically sends a **DPMSG_DELETEPLAYER** system message to all other players, informing them that this player has been removed from the session.

See also **IDirectPlay::CreatePlayer**, **DPMSG_DELETEPLAYER**

IDirectPlay::EnableNewPlayers

```
HRESULT EnableNewPlayers(BOOL bEnable);
```

Enables or disables the creation of new players.

- Returns DP_OK if successful, or **DPERR_INVALIDOBJECT** otherwise.

bEnable
If TRUE (the default condition for a session), new players can be created unless the session has reached its maximum capacity. If FALSE, any attempt to create a new player will return an error.

This method does not affect your ability to create groups. Typically, new players and groups can be added to a session until the session's player limit has been reached. This method can override this behavior if, for example, a session is in progress and new players are not desired. The **IDirectPlay::EnumSessions** method will not enumerate sessions where **IDirectPlay::EnableNewPlayers** has been set to FALSE unless the DPENUMSESSIONS_ALL flag is used.

See also **IDirectPlay::CreatePlayer**, **IDirectPlay::CreateGroup**, **IDirectPlay::EnumSessions**

IDirectPlay::EnumGroupPlayers

```
HRESULT EnumGroupPlayers(DPID pidGroupPID,
    LPDPENUMPLAYERSCALLBACK lpEnumPlayersCallback,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates all of the players of a particular group existing in the current session.

- Returns DP_OK if successful, or one of the following error messages otherwise:

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

pidGroupPID
Player ID of the group to be enumerated.

lpEnumPlayersCallback
Address for the **EnumPlayersCallback** function to be called for every player in the group.

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

dwFlags
This parameter is not currently used and must be set to 0.

See also **IDirectPlay::CreatePlayer**, **IDirectPlay::DestroyPlayer**

IDirectPlay::EnumGroups

```
HRESULT EnumGroups(DWORD dwSessionID,
    LPDPENUMPLAYERSCALLBACK lpEnumPlayersCallback,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the groups in a session.

- Returns DP_OK if successful, or one of the following error messages otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**

  DPERR_UNSUPPORTED

dwSessionID
Session of interest. Do not use unless the DPENUMPLAYERS_SESSION flag is specified.

lpEnumPlayersCallback
Address of the **EnumPlayersCallback** function to be called for every group in the session.

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

dwFlags
Specifies the optional control flag.

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

By default, this method 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 that a session's name server provide its list for enumeration. This method cannot be called from within an **IDirectPlay::EnumSessions** enumeration. Furthermore, use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay::EnumSessions** method has been called, and before any attempt to open or close has been made.

See also **IDirectPlay::CreatePlayer**, **IDirectPlay::DestroyPlayer**

IDirectPlay::EnumPlayers

```
HRESULT EnumPlayers(DWORD dwSessionId,
    LPDPENUMPLAYERSCALLBACK lpEnumPlayersCallback,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the players in a session. Groups can also be included in the enumeration by using the DPENUMPLAYERS_GROUP flag.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

dwSessionID
Session of interest. Not used unless the DPENUMPLAYERS_SESSION flag is specified.

lpEnumPlayersCallback
Address for the **EnumPlayersCallback** function that will be called for every player in the session.

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

dwFlags
Specifies the control flags.

DPENUMPLAYERS_GROUP
Includes groups in the enumeration of players.

DPENUMPLAYERS_PREVIOUS
Enumerates players previously stored in the registry. Not yet supported.

DPENUMPLAYERS_SESSION
Requests that the name server supply its group list for the specified session.

This method is often called immediately after the DirectPlay object is opened, and will enumerate using the local player list for the current session by default. The **IDirectPlay::EnumPlayers** method may be called after the **IDirectPlay::EnumSessions** method in order to obtain a list of players in a particular session. **IDirectPlay::EnumPlayers** uses the DPENUMPLAYERS_SESSION flag and the session ID returned from the **IDirectPlay::EnumSessions** method to obtain this list. However, it cannot be called from within an **IDirectPlay::EnumSessions** enumeration. The use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay::EnumSessions** method has been called and before either the **IDirectPlay::Close** or the **IDirectPlay::Open** method has been called.

See also **IDirectPlay::CreatePlayer**, **IDirectPlay::DestroyPlayer**, **IDirectPlay::EnumSessions**

IDirectPlay::EnumSessions

```
HRESULT EnumSessions(LPDPSESSIONDESC lpSDesc,
    DWORD dwTimeout,
    LPDPENUMSESSIONSCALLBACK lpEnumSessionsCallback,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the game sessions connected to this DirectPlay object.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**

  DPERR_NOSESSIONS

lpSDesc

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

dwTimeout

Total amount of time, in milliseconds, that DirectPlay will allow for the enumeration to complete (not the time between each enumeration).

lpEnumSessionsCallback

Address for the **EnumSessionsCallback** function to be called for each DirectPlay session responding.

lpContext

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

dwFlags

Specifies the optional control flags.

DPENUMSESSIONS_AVAILABLE

Enumerates all sessions with a matching password (if provided), opens player slots, and sets the **IDirectPlay::EnableNewPlayers** method to TRUE.

DPENUMSESSIONS_PREVIOUS

Enumerates sessions previously stored in the registry.

This method is usually called immediately after the DirectPlay object is instantiated; it cannot be called while a game is connected to a session or after a game has created a session. **IDirectPlay::EnumSessions** broadcasts an enumeration request and collects replies from the 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 value will be passed for the *lpDPSGameDesc* parameter. At this point, you may choose to continue the enumeration by setting *dwTimeOut* to a new value and returning TRUE, or returning FALSE to cancel the enumeration.

See also **DPSESSIONDESC**, **IDirectPlay::EnableNewPlayers**

IDirectPlay::GetCaps

```
HRESULT GetCaps(LPDPCAPS lpDPCaps);
```

Obtains the capabilities of this DirectPlay object.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**

lpDPCaps
Address for a **DPCAPS** structure that will be filled with the capabilities of the DirectPlay object.

See also **IDirectPlay::GetPlayerCaps**, **DPCAPS**

IDirectPlay::GetMessageCount

```
HRESULT GetMessageCount(DPID pidID, LPDWORD lpdwCount);
```

Queries the name server for the number of messages queued for a specific local player.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**
  DPERR_INVALIDPLAYER

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

lpdwCount
Address for a doubleword that will be set to the message count.

See also **IDirectPlay::Receive**

IDirectPlay::GetPlayerCaps

```
HRESULT GetPlayerCaps(DPID pidID, LPDPCAPS lpDPPlayerCaps);
```

Obtains the capabilities of the player's connection by querying the DirectPlay object.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**
  DPERR_INVALIDPLAYER

pidID
Player ID for which the communication capabilities are requested.

lpDPPlayerCaps
Address for a **DPCAPS** structure that will be filled with the communication capabilities of the specified player on this DirectPlay object.

This method is needed because communicating with some players may be slower than communicating with others.

See also **IDirectPlay::GetCaps**

IDirectPlay::GetPlayerName

```
HRESULT GetPlayerName(DPID pidID,
    LPSTR lpPlayerFriendlyName,
    LPDWORD lpdwFriendlyNameLength,
    LPSTR lpPlayerFormalName,
    LPDWORD lpdwFormalNameLength);
```

Obtains the player's friendly and formal names from the name server.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

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

lpPlayerFriendlyName
Address for the buffer to be filled with the player's friendly name. Set to NULL if you are only looking for the size of the friendly name or are only looking for the formal name.

lpdwFriendlyNameLength
Address for a doubleword that either contains the length of the buffer pointed to by the *lpPlayerFriendlyName* field or will be filled with the length needed for the buffer. Set to NULL if you are only interested in the formal name.

lpPlayerFormalName
Address for the buffer to be filled with the player's formal name. Set to NULL if you are only looking for the size of the formal name or are only looking for the friendly name.

lpdwFormalNameLength
Address for a doubleword that either contains the length of the buffer pointed to by the *lpPlayerFormalName* parameter or will be filled with the length needed for the buffer. Set to NULL if you are only interested in the friendly name.

If just one of the names is required, you can set the pair of pointers to the other name to NULL. If the length of the names needs to be determined, the pointers to the lengths must be valid; however, they can point to zeros or you can set the pointers to the friendly and formal names to 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.

See also **IDirectPlay::SetPlayerName**

IDirectPlay::Initialize

```
HRESULT Initialize(LPGUID lpGUID);
```

Provided to comply with the Common Object Model (COM) protocol.

- Returns **DPERR_ALREADYINITIALIZED**.

lpGUID
Address for the GUID used as the interface identifier.

Since the DirectPlay object is initialized when it is created, calling this method will always result in the DPERR_ALREADYINITIALIZED return value.

See also **IDirectPlay::AddRef**, **IDirectPlay::QueryInterface**

IDirectPlay::Open

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

Establishes the communication link between DirectPlay and a service provider.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

lpSDesc
Address for the **DPSESSIONDESC** structure describing the session to be connected to or created.

In a modem environment, calling this method is equivalent to actually dialing the phone. This prompts the required user interface to configure the communications protocol that 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, **IDirectPlay::Open** will return a DPERR_USERCANCEL error. This method will also return an error if any local players exist when it is called.

See also **IDirectPlay::Close**, **DPSESSIONDESC**

IDirectPlay::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID * ppvObj);
```

Determines if the DirectPlay object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by DirectPlay.

- Returns DP_OK if successful, or one of the following error values otherwise:

  DPERR_INVALIDOBJECT **DPERR_INVALIDPARAMS**

riid
Reference identifier of the interface being requested.

ppvObj
Address for a pointer to be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirectPlay::QueryInterface** method allows DirectPlay objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

See also **IDirectPlay::AddRef**, **IDirectPlay::Release**

IDirectPlay::Receive

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

Retrieves a message from the message queue.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

lppidFrom
Address for a DPID structure to be filled with the sender's player ID.

lppidTo
Address for a DPID structure to be filled with the receiver's player ID.

dwFlags
Specifies the optional control flags.

DPRECEIVE_ALL

Returns the first available message. This is the default.

DPRECEIVE_FROMPLAYER

Returns the first message from the player ID that the *lppidFrom* parameter points to.

DPRECEIVE_PEEK

Returns a message as specified by the other flags, but does not remove it from the message queue.

DPRECEIVE_TOPLAYER

Returns the first message intended for the player ID that the *lppidTo* parameter points to. System messages are addressed to player ID 0.

lpvBuffer

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

lpdwSize

Address for the doubleword that specifies the length of the message buffer.

Any message received from player ID 0 is a system message from the name server. A message sent to the name server to broadcast to all players still appears to be from the sender. Both the DPRECEIVE_TOPLAYER and DPRECEIVE_FROMPLAYER flags can be specified, in which case **IDirectPlay::Receive** will return whichever message is encountered first.

See also **IDirectPlay::Send**

IDirectPlay::Release

```
ULONG Release();
```

Decreases the reference count of the DirectPlay object by 1. This method is part of the **IUnknown** interface inherited by DirectPlay.

- Returns the new reference count of the object.

The DirectPlay object deallocates itself when its reference count reaches 0. Use the **IDirectPlay::AddRef** method to increase the reference count of the object by 1.

See also **IDirectPlay::AddRef**, **IDirectPlay::QueryInterface**

IDirectPlay::SaveSession

```
HRESULT SaveSession();
```

Saves the current session in the registry.

- Returns DP_OK if successful, or one of the following error values otherwise:

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

The functionality of this method is dependent on the service provider, which will save enough transport-specific information in the registry to restore the connection. **IDirectPlay::SaveSession** is unsupported in the TCP and IPX service providers. In the serial modem service provider, **IDirectPlay::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.

See also **IDirectPlay::EnumSessions**

IDirectPlay::Send

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

Sends messages to other players, to other groups of players, or to all players in the session.

- Returns DP_OK if successful, the number of messages waiting for transmission in DirectPlay's internal queue, or one of the following error values:

  DPERR_BUSY **DPERR_INVALIDOBJECT**
  DPERR_INVALIDPLAYER **DPERR_SENDTOOBIG**

pidFrom
ID of the sending player.

pidTo
ID of the receiving player.

dwFlags
Indicates how the message should be sent. Not all options may be supported by a particular service provider.

DPSEND_GUARANTEE

Sends the message using a reliable method. Retries until the message is received or until the DirectPlay time-out occurs.

DPSEND_HIGHPRIORITY

Sends the message as a HIGHPRIORITY message.

DPSEND_TRYONCE

Sends the message without error detection and without retry options enabled.

lpvBuffer
Address for the message being sent.

dwBuffSize
Length of the message being sent.

To send a message to another player, specify the appropriate player 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, specify the player ID 0, which always represents the name server. **IDirectPlay::Send** will either return one of the values listed above or the number of messages currently queued for transmission. If the internal queue fills to the limit specified by the **dwMaxQueueSize** member of the **DPCAPS** structure, an error will be generated and the message will not be added to the queue. The **IDirectPlay::Send** method cannot be used inside an **IDirectDrawSurface::Lock** / **IDirectDrawSurface::Unlock** or **IDirectDrawSurface::GetDC** / **IDirectDrawSurface::ReleaseDC** method pair.

See also **IDirectPlay::Receive**

IDirectPlay::SetPlayerName

```
HRESULT SetPlayerName(DPID pidID,
    LPSTR lpPlayerFriendlyName, LPSTR lpPlayerFormalName);
```

Changes the player's friendly and formal names.

- Returns DP_OK if successful, or one of the following error values otherwise:
  DPERR_INVALIDPLAYER **DPERR_INVALIDOBJECT**

pidID
ID for which the player name is being requested.

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

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

This method cannot be used to change the names of a group.

See also **IDirectPlay::GetPlayerName**

Structures

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

Data structures	**DPCAPS**
	DPSESSIONDESC
Message structures	**DPMSG_ADDPLAYER**
	DPMSG_DELETEPLAYER
	DPMSG_GENERIC
	DPMSG_GROUPADD
	DPMSG_GROUPDELETE

DPCAPS

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

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

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

dwFlags
Specifies the optional control flags.

DPCAPS_GUARANTEED

Supports verification of received messages. Retransmits the message, if necessary.

DPCAPS_NAMESERVER

The computer represented by the 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 specified in hundredths. For example, 2400 baud is represented by the value 24.

dwLatency
Latency estimate per service provider, in milliseconds. If this value is 0, DirectPlay cannot provide an estimate. Accuracy for some service providers rests on application-to-application testing, taking into consideration the average message size.

DPSESSIONDESC

```
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    dwUser2;
    DWORD    dwUser3;
    DWORD    dwUser4;
} DPSESSIONDESC;
typedef DPSESSIONDESC FAR *LPDPSESSIONDESC;
```

Contains a description of the capabilities of a DirectPlay session.

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

guidSession
Globally unique identifier (GUID) for the game. It 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
Specifies one of the control flags:

DPOPEN_CREATESESSION

Creates a new session described by the **DPSESSIONDESC** structure.

DPOPEN_OPENSESSION

Opens the session identified by the **dwSession** member.

DPENUMSESSIONS_ALL

Enumerates all active sessions connected to this DirectPlay object, regardless of their occupancy, passwords, or the **IDirectPlay::EnableNewPlayers** method's status.

szSessionName
String containing the name of the session.

szUserField
String containing user data.

dwReserved1
Reserved for future use.

szPassword
String containing the optional password that, once set up, is required to join this session.

dwReserved2
Reserved for future use.

dwUser1, **dwUser2**, **dwUser3**, **dwUser4**
User-specific data for the game or session.

System Messages

Messages returned by the **IDirectPlay::Receive** method from player ID 0 are system messages. All system messages begin with a doubleword **dwType**. You can cast the buffer returned by the **IDirectPlay::Receive** method 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, and so on).

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

Value of dwType	Message structure
DPSYS_ADDPLAYER	**DPMSG_ADDPLAYER**
DPSYS_ADDPLAYERTOGROUP	**DPMSG_GROUPADD**
DPSYS_DELETEGROUP	**DPMSG_DELETEPLAYER**
DPSYS_DELETEPLAYER	**DPMSG_DELETEPLAYER**
DPSYS_DELETEPLAYERFROMGRP	**DPMSG_GROUPDELETE**
DPSYS_SESSIONLOST	**DPMSG_GENERIC**

DPMSG_ADDPLAYER

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

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

dwType
Identifier for the message.

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

dpId
ID for a player or group.

szLongName
Formal name for player or group.

szShortName
Friendly name for player or group.

dwCurrentPlayers
Number of players in the session.

DPMSG_DELETEPLAYER

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

dwType
Identifier for the message.

dpId
ID of a player or group that has been deleted.

DPMSG_GENERIC

```
typedef struct{
    DWORD    dwType;
} DPMSG_GENERIC;
```

This structure is provided for message processing.

dwType
Identifier for the message.

Your application can first cast the unknown message to the DPMSG_GENERIC type, then perform further processing based on the value of **dwType**. One other system message, DPSYS_SESSIONLOST, also uses this structure.

DPMSG_GROUPADD

```
typedef struct{
    DWORD    dwType;
    DPID     dpIdGroup;
    DPID     dpIdPlayer;
} 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, respectively.

dwType
Identifier for the message.

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

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

DPMSG_GROUPDELETE

```
typedef DPMSG_GROUPADD DMSG_GROUPDELETE;
```

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

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all IDirectPlay methods. For a list of the error codes each method is capable of returning, see the individual method descriptions.

DP_OK

The request completed successfully.

DPERR_ACCESSDENIED

The session is full or an incorrect password was supplied.

DPERR_ACTIVEPLAYERS

The requested operation cannot be performed because there are existing active players.

DPERR_ALREADYINITIALIZED

This object is already initialized.

DPERR_BUFFERTOOSMALL

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

DPERR_BUSY

The DirectPlay message queue is full.

DPERR_CANTADDPLAYER

The player cannot be added to the session.

DPERR_CANTCREATEPLAYER

A new player cannot be created.

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

An undefined error condition occurred.

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

There is insufficient memory to perform the requested operation.

DPERR_SENDTOOBIG

The message buffer passed to the **IDirectPlay::Send** method is larger than allowed.

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 canceled the connection process during a call to the **IDirectPlay::Open** method.