# Chapter 4

# Microsoft® DirectX™ 3 Software Development Kit

DirectPlay

# DirectPlay



CHAPTER 4

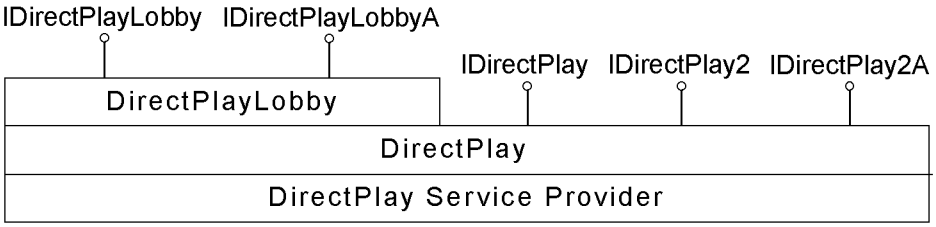
# About DirectPlay

The Microsoft® DirectPlay® application programming interface (API) for the Microsoft Windows® operating system is a software interface that simplifies your application's access to communication services. DirectPlay has become a technology family that not only provides a way for applications to communicate with each other that is independent of the underlying transport, protocol, or online service, but also provides this independence for matchmaking (lobby) servers.

Applications (especially games) can be more compelling 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 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 application.

# DirectPlay Architecture

DirectPlay uses a simple send/receive communication model to implement a connectivity API tailored to the needs of multiplayer applications. The DirectPlay architecture is composed of three components: DirectPlayLobby, DirectPlay, and the DirectPlay service provider. The following figure shows the relationship between these components, and their corresponding interfaces:




This section contains general information about these components:

- *DirectPlay Component*
- *DirectPlayLobby Component*
- *Service Providers*

# DirectPlay Component

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

DirectPlay will dynamically bind to any DirectPlay service provider installed on the user's system. The application 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.

The DirectPlay API is exposed to the application through a COM interface. For DirectPlay version 3, there are two interfaces available: **IDirectPlay2** and **IDirectPlay2A**. **IDirectPlay2** uses Unicode strings in all the DirectPlay structures, whereas **IDirectPlay2A** uses ANSI strings. The **IDirectPlay** interface still exists as the default interface for backward compatibility with applications written for DirectPlay versions 1 and 2, and it uses only ANSI strings.

The application instantiates a single DirectPlay object and performs all communications through that object, even if the application manages multiple players. For performance, only the DirectPlay objects communicate directly with each other; after a DirectPlay object receives a message, it will replicate that message (if necessary) for all the players the local application created, and add those messages to the message queue.

This version of DirectPlay supports the peer-to-peer gaming paradigm—that is, any player can send a message directly to any other player in the session. A session *host* arbitrates new computers that join the session and assigns ID numbers when new players and groups are created. You can design a game in a client/server model where all messages are sent to a server player on the host computer, who then forwards them to the appropriate client players. Future versions of DirectPlay will support application servers.

# DirectPlayLobby Component

DirectPlayLobby is a standard way for custom matchmaking lobby solutions to interact with DirectPlay applications. A custom lobby solution will usually include some kind of *lobby client* software, which runs on the user's computer and communicates with a *lobby server*. The lobby client implements the user interface through which a user can locate other players to engage in a gaming session. When a group of players has decided to start a session, the lobby client starts the application on each of their computers and supplies them with the information they need to select a service provider and connect to the session. A DirectPlay application that can be started and connected through the

DirectPlayLobby API function is referred to as *lobby-able*. DirectPlayLobby also has methods that allow an application to communicate with the lobby client while the session is in progress, and to inform the lobby client when the application has terminated.

The lobby client can determine which DirectPlay applications the user has by using the **IDirectPlayLobby::EnumLocalApplications** method. It can also determine which service providers are available by using the **DirectPlayEnumerate** function. After a user decides to join a session and the lobby client verifies the availability of the necessary application and service provider, the lobby client can start the application and connect it to the session by using the **IDirectPlayLobby::RunApplication** method. In this call, the lobby client specifies the application to run, the DirectPlay service provider to use, the information the service provider will need to connect to the session (by using the **IDirectPlayLobby::CreateAddress** and **IDirectPlayLobby::EnumAddress** methods), and the name the user is known by within the lobby environment. DirectPlayLobby locates the application executable and starts it with the appropriate command line switches. DirectPlayLobby also stores all the service provider and connection information.

The lobby client and application can communicate by using the **IDirectPlayLobby::SendLobbyMessage** and **IDirectPlayLobby::ReceiveLobbyMessage** methods. The lobby client sets up an event to occur when a message is received in the **IDirectPlayLobby::RunApplication** method. The application sets up an event by using the **IDirectPlayLobby::SetLobbyMessageEvent** method. By using this method, the lobby can change its event, too.

For an application to be lobby-able, it must also create an *IDirectPlayLobby* interface. The application can examine the connection parameters that the lobby client provided by using the **IDirectPlayLobby::GetConnectionSettings** method, and then modify them by using the **IDirectPlayLobby::SetConnectionSettings** method. The application then calls **IDirectPlayLobby::Connect**, which uses the connection settings and connects to the session. During this time, the lobby client will receive system messages indicating the progress of starting the application.

The application should then obtain the player name data (by using **IDirectPlayLobby::GetConnectionSettings**) and assign it to the player being created. The **IDirectPlayLobby::Connect** call (if successful) takes the place of the following series of calls:

1. **DirectPlayEnumerate** (the lobby specifies the service provider)
2. **DirectPlayCreate** (**IDirectPlayLobby::Connect** will create a DirectPlay object)
3. **IDirectPlay2::EnumSessions** (the lobby also specifies the session information so the user doesn't have to pick one)
4. **IDirectPlay2::Open** (the session will open automatically)

If **IDirectPlayLobby::GetConnectionSettings** returns the DPERR_NOTLOBBIED error, the starting of the application was not initiated by a lobby client, and the application should go through its normal DirectPlay initialization.

## Service Providers

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 the following service providers with DirectPlay: direct modem-to-modem (TAPI), serial connection, Internet TCP/IP, and IPX.

# DirectPlay Overview

This section contains general information about the DirectPlay component. The following topics are discussed:

- *Session Management*
- *Player Management*
- *Group Management*
- *Message Management*
- *Data Management*
- *Using System Messages*
- *Synchronization*
- *DirectPlay Address*
- *What's New in DirectPlay Version 3?*

## Session Management

A DirectPlay session is an instance of several applications on remote computers communicating with each other. An application uses DirectPlay's session-management functions to open or close a communication channel. An application either creates a new session, or it enumerates existing sessions and finds one to connect to. The application that creates the session is referred to as the *host*. The host assigns player and group ID numbers and regulates new applications joining the session.

The application can use the **IDirectPlay2::EnumSessions** method to locate all the existing sessions in progress on the network. It can use the **IDirectPlay2::Open** method to create new sessions or to connect to an existing one. A session is described by its corresponding **DPSESSIONDESC2** structure.

This structure contains application-specific values and session particulars, such as the session's name, an optional password for the session, and the number of players permitted in the session. After it opens a session, your application can call the **IDirectPlay2::GetCaps** method to retrieve the speed of the communications link and other properties of the network and service provider.

The application can use the **IDirectPlay2::GetSessionDesc** method to obtain the session's current properties.

When an application must leave a session, it can use the **IDirectPlay2::Close** method. If the session host leaves the session, and the session was started by using the DPSESSION_MIGRATEHOST flag in the **DPSESSIONDESC2** structure, one of the other players in the session becomes the host, and a DPSYS_HOST system message is generated.

# Player Management

Your application can use DirectPlay's player-management methods to manage the players in a session. In addition to creating and destroying players, your application can enumerate the players or retrieve a player's communication capabilities.

The **IDirectPlay2::CreatePlayer** and **IDirectPlay2::DestroyPlayer** methods create and delete players in a session. When the player is created, the application can supply friendly and formal names for it, as well as some initial remote data. (For more information, see *Data Management*.) DirectPlay assigns the player a player ID, which the application and DirectPlay use to route message traffic. The application and DirectPlay use the player ID to route message traffic. DirectPlay does not use the friendly and formal names, but the application can use them to identify players.

Your application can use the **IDirectPlay2::EnumPlayers** method to determine which players are in a current session and what their friendly and formal names are. Your application should typically call this method immediately after the **IDirectPlay2::Open** method opens a session. Your application can use the **IDirectPlay2::EnumPlayers** method to enumerate all the players in a session. If your application needs information about the speeds of the players' connections to the session, it can use the **IDirectPlay2::GetPlayerCaps** method.

Your application can change the name associated with a player by using the **IDirectPlay2::SetPlayerName** method. This method sends a system message to the other players informing them that a player's name has changed. These players can determine the new name from the **DPMSG_SETPLAYERORGROUPNAME** system message, or by using **IDirectPlay2::GetPlayerName**.

# Group Management

The group-management methods allow your application to create groups of players in a session. The application can then send messages to the group, rather than to one player at a time, by using a single call to the **IDirectPlay2::Send** method. Some service providers can send messages to groups (multicasting) more efficiently than sending them to individual players in a group, so, in addition to simplifying player management, you can use groups to conserve communication-channel bandwidth.

The **IDirectPlay2::CreateGroup** and **IDirectPlay2::DestroyGroup** methods create and delete a group of players. When a group is created, the application can assign it friendly and formal names, just as it does when it creates a player. DirectPlay assigns the group a group ID. The group is initially empty, but the application can use the **IDirectPlay2::AddPlayerToGroup** and **IDirectPlay2::DeletePlayerFromGroup** methods to add and delete players. The state of the DPSESSION_NEWPLAYERSDISABLED flag in the session description does not affect the ability to create groups, or to add or delete players from a group.

To determine what groups already exist, the application can use the **IDirectPlay2::EnumGroups** method. To enumerate the players in a group, it can use the **IDirectPlay2::EnumGroupPlayers** method.

An application can change the name of a group by using the **IDirectPlay2::SetGroupName** method. This causes a system message to be sent to the other players, who can determine the new name by using the **IDirectPlay2::GetGroupName** method.

# Message Management

The message-management functions help your application route messages among players. With the exception of a small number of messages that the system has defined, the messages can be defined in any way the application requires. Your application can use the **IDirectPlay2::Send** method to send a message to a player, a group, or all the players in the session by specifying a player ID, a group ID, or DPID_ALLPLAYERS, respectively, as the destination. There is no limit to the size of the message that DirectPlay can send. Your application can call **IDirectPlay2::GetCaps** to find out the maximum number of bytes that can be sent in a single packet. Larger messages are sent by using several packets.

If the global state of a player or group changes and that change must be propagated to all the other players in the session, it might be more convenient to use the data management functions rather than send a message with the new data to the players. For more information, see *Data Management*.

To receive a message from the message queue, your application can use the **IDirectPlay2::Receive** method. This method allows your application to specify whether to receive the first message in the queue, only the messages to a particular player ID, or only those from a particular player ID. Your application can use the **IDirectPlay2::GetMessageCount** method to retrieve the number of messages waiting for a given player.

DirectPlay generates system messages that notify players of changes in the session. All system messages are from a virtual player defined by DPID_SYSMSG. System messages start with a 32-bit value that identifies the type of message. Constants that represent system messages begin with DPSYS_, and they have a corresponding message structure that must be used to interpret them. The application can control what system messages are generated by using flags in the **DPSESSIONDESC2** structure.

If an application uses a separate thread for retrieving messages, the application can specify a synchronization event that will be set when a message is received.

# Data Management

As of DirectX™ 3, DirectPlay has the ability to allow applications to associate data with players and groups. Because DirectPlay already keeps track of players and groups, applications no longer have to implement their own player and group list manager to keep track of information. In addition, DirectPlay allows applications to store two types of information: local and remote. Local data is available only to the object that sets it. Remote data, on the other hand, is propagated to each computer in the session. In effect, it becomes shared memory between all remote computers. You should use remote data to store data that does not change often and that all computers need to access. You should use local data to keep track of data that no other computer needs access to.

Your application can set the data for a player by using the **IDirectPlay2::SetPlayerData** method. The application can specify whether the data is local or remote by passing the appropriate flags with the call. If the data is remote, the application can also specify whether to use guaranteed or nonguaranteed message passing to propagate the data. An application can retrieve data for a player by using the **IDirectPlay2::GetPlayerData** method, again specifying whether to retrieve the local or remote data. In a similar manner, you can use the **IDirectPlay2::SetGroupData** and **IDirectPlay2::GetGroupData** methods for group data.

# Using System Messages

Messages returned by the **IDirectPlay2::Receive** method from player ID DPID_SYSMSG are system messages. All system messages begin with a doubleword value specified by **dwType**. You can cast the buffer returned by the **IDirectPlay2::Receive** method to a generic message (**DPMSG_GENERIC**) and

switch on the **dwType** element, which will have a value equal to one of the messages with the DPSYS_ prefix. After the application has determined which system message it is, the buffer should be cast to the appropriate structure (beginning with the DPMSG_ prefix) to read the data.

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

| Value of dwType | Message structure |
|---|---|
| DPSYS_ADDPLAYERTOGROUP | **DPMSG_ADDPLAYERTOGROUP** |
| DPSYS_CREATEPLAYERORGROUP | **DPMSG_CREATEPLAYERORGROUP** |
| DPSYS_DELETEPLAYERFROMGROUP | **DPMSG_DELETEPLAYERFROMGROUP** |
| DPSYS_DESTROYPLAYERORGROUP | **DPMSG_DESTROYPLAYERORGROUP** |
| DPSYS_HOST | **DPMSG_HOST** |
| DPSYS_SESSIONLOST | **DPMSG_SESSIONLOST** |
| DPSYS_SETPLAYERORGROUPDATA | **DPMSG_SETPLAYERORGROUPDATA** |
| DPSYS_SETPLAYERORGROUPNAME | **DPMSG_SETPLAYERORGROUPNAME** |

Messages returned by the **IDirectPlayLobby::ReceiveLobbyMessage** method that have a *dwFlags* parameter set to DPLAD_SYSTEM are system messages. All system messages begin with a doubleword value specified by **dwType**. You can cast the buffer returned by the **IDirectPlayLobby::ReceiveLobbyMessage** method to a generic message (**DPLMSG_GENERIC**) and switch on the **dwType** element, which will have a value equal to one of the messages with the DPLSYS_ prefix.

# Synchronization

DirectPlay does not attempt to provide a general approach for application synchronization; to do so would necessarily impose limitations on the application-communications 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 **WaitForSingleObject** Win32® function to find out whether a message is pending for that player.

# DirectPlay Address

The information in this section is included for DirectPlayLobby client developers, and contains information that, in general, is not relevant for application developers.

DirectPlay can encapsulate network address data. DirectPlay contains all the information needed to connect to a DirectPlay session. The purpose of this information is to be able to connect an application to a session without the service provider having to display any dialog boxes prompting the user for information.

By providing the service provider with a complete DirectPlay Address, an application can bypass the dialog boxes that the service provider would typically display to obtain this information from the user.

The DirectPlay Address is in a format similar to the Resource Interchange File Format (RIFF). It is composed of a series of chunks. Each chunk consists of the following:

- A globally unique identifier (GUID) indicating what type of data this chunk contains
- The size of the data
- The data field

DirectPlay has predefined the following chunk identifiers:

| GUID | Type of data |
|---|---|
| DPAID_ComPort | A **DPCOMPORTADDRESS** structure that specifies:<br>• COM port to use (1-4)<br>• Baud (100-256k)<br>• Number of stop bits (1-2)<br>• Parity (0-none, 1-odd, 2-even, 3-mark)<br>• Flow control (0-none, 1-xon/xoff, 2-RTS, 3-DTR, 4-RTS/DTR) |
| DPAID_Inet | ASCII string representing an IP address of the form "xxx.xxx.xxx.xxx" or a server name, such as "dplay.microsoft.com". |
| DPAID_Phone | ASCII string representing the numeric digits of a phone number. |
| DPAID_ServiceProvider | A 16-byte GUID of the service provider for which this address was created. This chunk can be ignored because several different service providers can use the same type of network address. |

A chunk identifier is a 16-byte GUID.

# What's New in DirectPlay Version 3?

The DirectPlay version 3 API is fully compatible with applications written for any previous version of DirectPlay. That is, you can recompile your application by using DirectPlay on the DirectX 3 SDK without making any changes to the code. DirectPlay supplied with the DirectX 3 SDK supports all the APIs and behavior of earlier DirectPlay versions.

The names of the DirectPlay 3 DLLs are different from those in previous DirectPlay versions, so applications compiled with DirectX 2 or earlier do not use the new DirectPlay DLLs. To use the new DLLs, the application must be recompiled and linked to the Dplayx.lib import library.

It is strongly recommended that you update your application to use the new *IDirectPlay2* or **IDirectPlay2A** interfaces, and add the code necessary to make the application lobby-able. This means that an external lobby or matchmaking program can start the application and supply it with all the information necessary to connect to a session. The application need not ask the user to decide on a service provider, select a session, or supply any other information (such as a telephone number or network address).

In addition, several other new features have been added to the DirectPlay API, including the following:

- Internet support.
- Direct serial connection.
- More stability and robustness.
- Support for Unicode to better support localization.
- Host migration. If the host of a session drops out of the session, host responsibilities are passed on to another player. In DirectPlay version 2, if the host (name server) left a session, no new players could be created.
- Ability of the application to communicate with the lobby program to update games status for spectators, as well as receive information about initial conditions.
- Ability to host more than one application session on a computer.
- Ability to determine when a remote computer loses its connection and to generate appropriate messages.

There are also other features in DirectPlay 3 that you can use to reduce the amount of communication-management code in your application, including the following:

- Ability to associate application-specific data with a DirectPlay group ID or player ID. This allows the application to leverage the player and group list-management code that is already part of DirectPlay. Local data is data that the local application uses directly, such as a bitmap that represents a player. Local data is not propagated over the network. Remote data is associated with the player or group. DirectPlay propagates any changes to remote data to all other applications in the session. Remote data must be shared among all the applications in a session, such as a player's position, orientation, and velocity. By using DirectPlay functions to propagate this data, the application need not manage it by using a series of methods that send and receive information.

- Ability for application to associate a name with a group. This is useful for team play.

Some of the new features of DirectPlay 3 are not directly related to your application, including the following:

- APIs that the lobby client software uses to start and connect a lobby-able DirectPlay application. Also included are APIs that allow an application and the lobby to exchange information during a session.
- Service Provider development kit that includes documentation and sample code for creating your own service provider.

This section discusses the new DirectPlay 3 methods, the steps you must take to migrate to the **IDirectPlay2** interface, and how you can access DirectPlay's updated functionality. The following topics are discussed:

- *New DirectPlay 3 Methods*
- *Migrating to the IDirectPlay2 Interface*

## New DirectPlay 3 Methods

DirectPlay version 3 supports the following new methods:

- **IDirectPlay2::SetGroupData** and **IDirectPlay2::GetGroupData**

  Associate application-specific data with a DirectPlay group ID either locally to the application or in the remote data space.
- **IDirectPlay2::SetGroupName** and **IDirectPlay2::GetGroupName**

  Associate a name with a group.
- **IDirectPlay2::SetPlayerData** and **IDirectPlay2::GetPlayerData**

  Associate application-specific data with a DirectPlay player ID either locally to the application or in the remote data space.
- **IDirectPlay2::GetSessionDesc**

  Retrieves the session's properties while the session is in progress.

## Migrating to the IDirectPlay2 Interface

To migrate your application to use the *IDirectPlay2* interface, carry out the following steps:

1. Find out if your application was launched from a lobby client. For more information, see *Step 2: Retrieving the Connection Settings* in the DirectPlay tutorials.
2. If your application is enumerating service providers, use the **DirectPlayEnumerate** callback function to determine if a service provider is available. If so, call the **DirectPlayCreate** function on the service provider. If the

**DirectPlayEnumerate** callback function returns an error, the service provider cannot run on the system, and you should not add that service provider to the list to show to the user. If the call succeeds, use the **Release** method to release the DirectPlay object and add the service provider to the list.

3 Call the **QueryInterface** method on the **IDirectPlay** interface to obtain an *IDirectPlay2* (Unicode) or **IDirectPlay2A** (ANSI) interface. The only difference between the two interfaces is how strings in the structures are read and written. For the Unicode interface, Unicode strings are read and written to the member of the structure that is of the **LPWSTR** type. For the ANSI interface, ANSI strings are read and written to the member of the structure that is of the **LPSTR** type.

4 Make all the changes necessary to use the new structures in existing APIs. For example, instead of the following:

```
lpDP->SetPlayerName(pidPlayer, lpszFriendlyName, lpszFormalName)
```

where *lpDP* is an **IDirectPlay** interface, use the following:

```
DPNAME PlayerName, *lpPlayerName;
PlayerName.dwSize = sizeof(DPNAME);
lpPlayerName = &PlayerName;

lpPayerName->lpszShortNameA = lpszFriendlyName;
lpPlayerName->lpszLongNameA = lpszFormalName;
lpDP2A->SetPlayerName(pidPlayer, lpPlayerName, 0)
```

where *lpDP2A* is an **IDirectPlay2A** interface. If the application is using Unicode strings (and therefore instantiates an **IDirectPlay2** interface), use the following:

```
lpPayerName->lpszShortName = lpwszFriendlyName;
lpPlayerName->lpszLongName = lpwszFormalName;
lpDP2->SetPlayerName(pidPlayer, lpPlayerName, 0)
```

where *lpDP2* is an **IDirectPlay2** interface.

5 Update the following system messages:

- DPSYS_ADDPLAYER has been replaced by DPSYS_CREATEPLAYERORGROUP.
- DPSYS_DELETEPLAYER and DPSYS_DELETEGROUP have been combined in a single DPSYS_DESTROYPLAYERORGROUP message.
- DPSYS_DELETEPLAYERFROMGRP has been changed to DPSYS_DELETEPLAYERFROMGROUP.

6 Update your application to generate a DPSYS_SETPLAYERORGROUPNAME message when a player or group name changes, and a DPSYS_SETPLAYERORGROUPDATA message when the player or group data changes.

7 Update the **DPSESSIONDESC** structure to **DPSESSIONDESC2**, and add the new members to the **DPCAPS** structure.

8. Update the callback functions for **IDirectPlay2::EnumSessions**, **IDirectPlay2::EnumGroups**, **IDirectPlay2::EnumGroupPlayers**, and **IDirectPlay2::EnumPlayers**.
9. Update the manner in which the *hEvent* parameter is supplied to the **IDirectPlay2::CreatePlayer** method. In previous versions of DirectPlay, this parameter was *lpEvent*. DirectPlay does not return an event; instead, the application must create it. This allows the application the flexibility of creating one event for all the players.
10. Set the DPSESSION_KEEPALIVE flag in the **DPSESSIONDESC2** structure if the application needs DirectPlay to detect when players drop out of the game abnormally.
11. Update your application to create sessions with the DPSESSION_MIGRATEHOST flag. This enables another computer to become the host if the current host drops out of the session. If your application has any special code that only the host runs, then your application should set this flag when it creates a session. It should also add support for the DPSYS_HOST system message. For a list of system messages, see *Using System Messages*.
12. Become familiar with the new methods of the **IDirectPlay2** interface and use them in your application. Pay particular attention to the **IDirectPlay2::SetPlayerData** and **IDirectPlay2::GetPlayerData** methods. You might be able to substitute the code where you broadcast player state information to all the other players by using the **IDirectPlay2::Send** and **IDirectPlay2::Receive** methods.

# DirectPlay Interface Overviews

DirectPlay is composed of objects and interfaces based on the component object model (COM). COM is a foundation for an object-based system that focuses on the reuse of interfaces, and it is the model at the heart of OLE programming. It is also an interface specification from which any number of interfaces can be built.

In previous versions of DirectX, the DirectPlay object was made up of only one interface, **IDirectPlay**. DirectPlay has now been expanded to include new functionality that provides better access to more types of communications solutions. New interfaces have been added to DirectPlay to include support for Unicode and ANSI strings, and the building blocks for creating matchmaking (lobby) services.

This section contains general information about the following DirectPlay COM interfaces:

- *IDirectPlay Interface*
- *IDirectPlay2 Interface*
- *IDirectPlayLobby Interface*

# IDirectPlay Interface

The **IDirectPlay** COM interface remains part of DirectPlay version 3. It contains the methods required to run applications that were written for the DirectX SDK versions 1 and 2. Although you could use this interface to create new applications, it is recommended that you use the newer DirectPlay interfaces, *IDirectPlay2* and **IDirectPlay2A**, to take advantage of their increased functionality.

# IDirectPlay2 Interface

DirectPlay supports both Unicode and ANSI strings by defining string pointers in a structure as the union of a Unicode string pointer (**LPWSTR**) and an ANSI string pointer (**LPSTR**). The two string pointers have different names. Typically, the ANSI string pointer ends with the letter "A". Depending on which **IDirectPlay** interface is chosen (*IDirectPlay2* for Unicode or **IDirectPlay2A** for ANSI), the application should read and write the appropriate strings from the structure and ignore the other one.

# IDirectPlayLobby Interface

The following topics contain additional information related to the *IDirectPlayLobby* interface:

- *Unicode Versus ANSI DirectPlayLobby Interfaces*
- *Registering Lobby-able Applications*

## Unicode Versus ANSI DirectPlayLobby Interfaces

DirectPlayLobby supports both Unicode and ANSI strings by defining string pointers in a structure as the union of a Unicode string pointer (**LPWSTR**) and an ANSI string pointer (**LPSTR**). The two string pointers have different names. Typically, the ANSI string pointer ends with the letter "A". Depending on which **IDirectPlayLobby** interface is chosen (*IDirectPlayLobby* for Unicode or **IDirectPlayLobbyA** for ANSI), the application should read and write the appropriate strings from the structure and ignore the other one.

## Registering Lobby-able Applications

For an application to be enumerated and started by DirectPlayLobby, it must add some information to the Windows registry when it is installed. The following registry keys have been defined for this purpose. "Application Name" is the human-readable name of the application that will be returned when DirectPlayLobby enumerates it. You can use the **DirectXRegisterApplication** function in DirectSetup to add these entries.

**Note**

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\DirectPlay\Applications\
Application Name]
"Guid"              GUID of the application
"Filename"          Filename of the executable
"CommandLine"       Command-line switches for the application (if any)
"Path"              Path of the application executable
"CurrentDirectory"  Path of the directory to start application into
```

# DirectPlay Tutorials

This section contains two tutorials that provide step-by-step instructions about how to connect an application with or without a lobby. The LOBBY example demonstrates how to connect an application by using a DirectPlay lobby. The DIALOG example demonstrates how to connect an application by using a dialog box that queries the user for connection information. You should write your application so that it can start by using either method.

- *Tutorial 1: Connecting by Using the Lobby* (LOBBY)
- *Tutorial 2: Connecting by Using a Dialog Box* (DIALOG)

---

The sample files in these tutorials are written in C++. If you are using a C compiler, you must make the appropriate changes to the files for them to successfully compile. At the very least, you must add the vtable and **this** pointers to the interface methods. For more information, see *Accessing COM Objects by Using C*.

---

## Tutorial 1: Connecting by Using the Lobby

An application written to use the *IDirectPlayLobby* interface can be connected without requiring the user to manually enter connection information in a dialog box. To demonstrate how to create a lobbied application, the LOBBY sample performs the following steps:

- *Step 1: Creating a DirectPlayLobby Object*
- *Step 2: Retrieving the Connection Settings*
- *Step 3: Configuring the Session Description*
- *Step 4: Connecting to a Session*
- *Step 5: Creating a Player*

### Step 1: Creating a DirectPlayLobby Object

To use a DirectPlay lobby, you first create an instance of a DirectPlayLobby object by calling the **DirectPlayLobbyCreate** function. This function contains five parameters. The first, third, and fourth parameters are always set to NULL and are included for future expansion. The second parameter contains the address

of a pointer that identifies the location of the DirectPlayLobby object if it is created. The fifth parameter is always set to 0, and is also included for future expansion.

The following example shows one way to create a DirectPlayLobby object:

```
// Get an ANSI DirectPlay lobby interface.
hr = DirectPlayLobbyCreate(NULL, &lpDirectPlayLobbyA, NULL, NULL, 0);
if FAILED(hr)
    goto FAILURE;
```

## Step 2: Retrieving the Connection Settings

After the DirectPlayLobby object has been created, use the **IDirectPlayLobby::GetConnectionSettings** method to retrieve the connection settings returned from the lobby. If this method returns DPERR_NOTLOBBIED, the lobby did not start this application and the user will have to configure the connection manually. If any other error occurs, your application should report an error that indicates that lobbying the application failed.

The following example shows how to retrieve the connection settings:

```
// Retrieve the connection settings from the lobby.
// If this routine returns DPERR_NOTLOBBIED, then a lobby did not
// start this application and the user needs to configure the
// connection.

// Pass a NULL pointer to retrieve only the size of the
// connection setttings
hr = lpDirectPlayLobbyA->GetConnectionSettings(0, NULL, &dwSize);
if (DPERR_BUFFERTOOSMALL != hr)
    goto FAILURE;

// Allocate memory for the connection setttings.
lpConnectionSettings = (LPDPLCONNECTION) GlobalAllocPtr(GHND, dwSize);
    if (NULL == lpConnectionSettings)
    {
        hr = DPERR_OUTOFMEMORY;
        goto FAILURE;
    }

    // Retrieve the connection settings.
    hr = lpDirectPlayLobbyA->GetConnectionSettings(0,
        lpConnectionSettings, &dwSize);
    if FAILED(hr)
        goto FAILURE;
```

## Step 3: Configuring the Session Description

You should examine the **DPSESSIONDESC2** structure to ensure that all the flags and properties that your application needs are set properly. If modifications are necessary, store the modified connection settings by using the **IDirectPlayLobby::SetConnectionSettings** method.

The following example shows how to configure the session description and set the connection settings:

```
// Before the game connects, it should configure the session
// description with any settings it needs.

// Set the flags and maximum players used by the game.
lpConnectionSettings->lpSessionDesc->dwFlags = DPSESSION_MIGRATEHOST |
    DPSESSION_KEEPALIVE;
lpConnectionSettings->lpSessionDesc->dwMaxPlayers = MAXPLAYERS;

// Store the updated connection settings.
hr = lpDirectPlayLobbyA->SetConnectionSettings(0, 0,
        lpConnectionSettings);
if FAILED(hr)
    goto FAILURE;
```

## Step 4: Connecting to a Session

After the session description is properly configured, your application can use the **IDirectPlayLobby::Connect** method to start and connect itself to a session. If this method returns DP_OK, you can create one or more players. If it returns DPERR_NOTLOBBIED, the user will have to manually select a communication medium for your application. (You can identify the service providers installed on the system by using the **DirectPlayEnumerate** function.) If any other error value is returned, your application should report an error that indicates that lobbying the application failed.

The following example shows how to connect to a session:

```
// Connect to the session. Returns an ANSI IDirectPlay2A interface.
hr = lpDirectPlayLobbyA->Connect(0, &lpDirectPlay2A, NULL);
if FAILED(hr)
    goto FAILURE;
```

## Step 5: Creating a Player

If the application was successfully started by using the **IDirectPlayLobby::Connect** method, it can now create one or more players. It can use the **IDirectPlay2::CreatePlayer** method to create a player with the name specified in the **DPNAME** structure (which was filled in by the **IDirectPlayLobby::GetConnectionSettings** method).

The following example shows how to create a player:

```
// create a player with the name returned in the connection settings
hr = lpDirectPlay2A->CreatePlayer(&dpidPlayer,
    lpConnectionSettings->lpPlayerName,
    lpDPInfo->hPlayerEvent, NULL, 0, 0);
if FAILED(hr)
    goto FAILURE;
```

Now your application is connected and you are ready to play.

# Tutorial 2: Connecting by Using a Dialog Box

If a lobby did not start your application, you should include code that allows the user to manually enter the connection information. To demonstrate how to manually connect to the session and create one or more players, the DIALOG sample performs the following steps:

- *Step 1: Enumerating the Service Providers*
- *Step 2: Creating the DirectPlay Object*
- *Step 3: Joining a Session*
- *Step 4: Creating a Session*
- *Step 5: Creating a Player*

## Step 1: Enumerating the Service Providers

The first step in creating a manual connection is to request that the user select a communication medium for the application. Your application can identify the service providers installed on a personal computer by using the **DirectPlayEnumerate** function.

The following example shows how to enumerate the service providers:

```
DirectPlayEnumerate(DirectPlayEnumerateCallback, hWnd);
```

The first parameter in the **DirectPlayEnumerate** function is a callback that enumerates service providers registered with DirectPlay. The following example shows one possible way of implementing this callback function:

```
BOOL FAR PASCAL DirectPlayEnumerateCallback(
    LPGUID lpSPGuid, LPTSTR lpszSPName, DWORD dwMajorVersion,
    DWORD dwMinorVersion, LPVOID lpContext)
{
HWND    hWnd = lpContext;
LRESULT iIndex;
LPGUID  lpGuid;
```

```
// Store the service provider name in a combo box.
iIndex = SendDlgItemMessage(hWnd, IDC_SPCOMBO, CB_ADDSTRING,
    0, (LPARAM) lpszSPName);
if (iIndex == CB_ERR)
    goto FAILURE;

// Make space for the application GUID.
lpGuid = (LPGUID) GlobalAllocPtr(GHND, sizeof(GUID));
if (lpGuid == NULL)
    goto FAILURE;

// Store the pointer to the GUID in a combo box.
*lpGuid = *lpSPGuid;
SendDlgItemMessage(hWnd, IDC_SPCOMBO, CB_SETITEMDATA,
    (WPARAM) iIndex, (LPARAM) lpGuid);

FAILURE:
    return (TRUE);
}
```

## Step 2: Creating the DirectPlay Object

After the user has selected which service provider to use, your application can create a DirectPlay object based on the selection by calling the **DirectPlayCreate** function and specifying the appropriate service provider's globally unique identifier (GUID). Calling this function causes DirectPlay to load the library for the selected service provider and returns an **IDirectPlay** interface.

Although you could use the **IDirectPlay** interface to create new games, a better approach would be to use the newer DirectPlay interfaces, *IDirectPlay2* and **IDirectPlay2A**, with all their increased functionality. Your application can obtain these interfaces by calling the **QueryInterface** method on the **IDirectPlay** interface returned by **DirectPlayCreate**.

The following example shows how the create the **IDirectPlay** interface, and then use **QueryInterface** to create an **IDirectPlay2A** interface:

```
HRESULT CreateDirectPlayInterface(LPGUID lpguidServiceProvider,
    LPDIRECTPLAY2A *lplpDirectPlay2A)
{
LPDIRECTPLAY   lpDirectPlay1 = NULL;
LPDIRECTPLAY2A lpDirectPlay2A = NULL;
HRESULT        hr;

// Retrieve a DirectPlay 1.0 interface.
hr = DirectPlayCreate(lpguidServiceProvider, &lpDirectPlay1, NULL);
if FAILED(hr)
    goto FAILURE;
```

```
// Query for an ANSI DirectPlay2 interface.
hr = lpDirectPlay1->QueryInterface(IID_IDirectPlay2A,
    (LPVOID *) &lpDirectPlay2A);
if FAILED(hr)
    goto FAILURE;

// Return the created interface.
*lplpDirectPlay2A = lpDirectPlay2A;

FAILURE:
if (lpDirectPlay1)
    lpDirectPlay1->Release();

return (hr);
}
```

## Step 3: Joining a Session

If the user wants to join an existing session, enumerate the available sessions by using the **IDirectPlay2::EnumSessions** method, present the choices to the user, and then connect to that session by using the **IDirectPlay2::Open** method, specifying the DPOPEN_JOIN flag. The service provider might display a dialog box requesting some information from the user before it can enumerate the sessions.

The following example shows how to enumerate the available sessions:

```
// Search for this kind of session.
ZeroMemory(&sessionDesc, sizeof(DPSESSIONDESC2));
sessionDesc.dwSize = sizeof(DPSESSIONDESC2);
sessionDesc.guidApplication = DPCHAT_GUID;

hr = lpDirectPlay2A->EnumSessions(&sessionDesc, 0, EnumSessionsCallback,
    hWnd, DPENUMSESSIONS_AVAILABLE);
if FAILED(hr)
    goto FAILURE;
```

The third parameter in the **IDirectPlay2A::EnumSessions** method is a callback that enumerates the available sessions. The following example shows one way to implement this callback function:

```
BOOL FAR PASCAL EnumSessionsCallback(
    LPCDPSESSIONDESC2 lpSessionDesc, LPDWORD lpdwTimeOut,
    DWORD dwFlags, LPVOID lpContext)
{
HWND   hWnd = lpContext;
LPGUID lpGuid;
LONG   iIndex;

// Determine if the enumeration has timed out.
```

```
if (dwFlags & DPESC_TIMEDOUT)
    return (FALSE);            // Do not try again

// Store the session name in the list.
iIndex = SendDlgItemMessage(hWnd, IDC_SESSIONLIST, LB_ADDSTRING,
    (WPARAM) 0, (LPARAM) lpSessionDesc->lpszSessionNameA);
if (iIndex == CB_ERR)
    goto FAILURE;

// Make space for the session instance GUID.
lpGuid = (LPGUID) GlobalAllocPtr(GHND, sizeof(GUID));
if (lpGuid == NULL)
    goto FAILURE;

// Store the pointer to the GUID in the list.
*lpGuid = lpSessionDesc->guidInstance;
SendDlgItemMessage(hWnd, IDC_SESSIONLIST, LB_SETITEMDATA,
    (WPARAM) iIndex, (LPARAM) lpGuid);

FAILURE:
    return (TRUE);
}
```

After the user has selected a session, your application can allow the user to join an existing session. The following example shows how to join an existing session:

```
// Join an existing session.
ZeroMemory(&sessionDesc, sizeof(DPSESSIONDESC2));
sessionDesc.dwSize = sizeof(DPSESSIONDESC2);
sessionDesc.guidInstance = *lpguidSessionInstance;

hr = lpDirectPlay2A->Open(&sessionDesc, DPOPEN_JOIN);
if FAILED(hr)
    goto OPEN_FAILURE;
```

## Step 4: Creating a Session

If the user wants to create a new session, your application can create it by using the **IDirectPlay2::Open** method and specifying the DPOPEN_CREATE flag. Again, the service provider might display a dialog box requesting information from the user before it can create the session.

The following example shows how to create a new session:

```
// Host a new session.
ZeroMemory(&sessionDesc, sizeof(DPSESSIONDESC2));
sessionDesc.dwSize = sizeof(DPSESSIONDESC2);
sessionDesc.dwFlags = DPSESSION_MIGRATEHOST | DPSESSION_KEEPALIVE;
sessionDesc.guidApplication = DPCHAT_GUID;
sessionDesc.dwMaxPlayers = MAXPLAYERS;
```

```
sessionDesc.lpszSessionNameA = lpszSessionName;

hr = lpDirectPlay2A->Open(&sessionDesc, DPOPEN_CREATE);
if FAILED(hr)
    goto OPEN_FAILURE;
```

## Step 5: Creating a Player

After a session has been created or joined, your application can create one or more players by using the **IDirectPlay2::CreatePlayer** method. The following example shows one way to create a player:

```
// Fill out the name structure.
ZeroMemory(&dpName, sizeof(DPNAME));
dpName.dwSize = sizeof(DPNAME);
dpName.lpszShortNameA = lpszPlayerName;
dpName.lpszLongNameA = NULL;

// Create a player with this name.
hr = lpDirectPlay2A->CreatePlayer(&dpidPlayer, &dpName,
    lpDPInfo->hPlayerEvent, NULL, 0, 0);
if FAILED(hr)
    goto CREATEPLAYER_FAILURE;
```

Your application can determine a player's communication capabilities by using the **IDirectPlay2::GetCaps** and **IDirectPlay2::GetPlayerCaps** methods. Your application can find other players by using the **IDirectPlay2::EnumPlayers** method.

Now your application is connected and you are ready to play.

# DirectPlay Reference

## Functions

### DirectPlayCreate

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

Creates an instance of a DirectPlay object.

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

  **CLASS_E_NOAGGREGATION**

  **DPERR_EXCEPTION**

  **DPERR_INVALIDPARAMS**

**DPERR_UNAVAILABLE**

*lpGUID*
Address of the globally unique identifier (GUID) that represents the service provider that should be created.

*lplpDP*
Address of a pointer to be initialized with a valid DirectPlay interface. The application will need to use the **QueryInterface** method to obtain an *IDirectPlay2* (Unicode strings) or **IDirectPlay2A** (ANSI strings) interface.

*pUnkOuter*
Address of the containing *IUnknown* interface. This parameter is provided for future compatibility with COM aggregation features. Presently, however, the **DirectPlayCreate** function returns 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. Your application should call the **DirectPlayEnumerate** function immediately before initialization to determine what types of service providers are available (the **DirectPlayEnumerate** function fills in the *lpGUID* parameter of **DirectPlayCreate**).

This function returns a pointer to an **IDirectPlay** interface. The current interfaces for DirectX 3 are **IDirectPlay2** and **IDirectPlay2A**, which need to be obtained through a call to the **QueryInterface** method on the **IDirectPlay** interface returned by **DirectPlayCreate**.

See also **DirectPlayEnumerate**

# DirectPlayEnumerate

```
HRESULT WINAPI 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**

  **DPERR_INVALIDPARAMS**

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

*lpContext*
Address of an application-defined structure that will be passed to the callback function each time the function is called.

This function will enumerate service providers installed in the system even though the system might not be capable of using those service providers. For example, a TAPI service provider will be part of the enumeration even though the system might not have a modem installed.

## DirectPlayLobbyCreate

```
HRESULT WINAPI DirectPlayLobbyCreate(
    LPGUID lpguidSP, LPDIRECTPLAYLOBBY *lplpDPL,
    IUnknown *lpUnk, LPVOID lpData, DWORD dwDataSize);
```

Creates an instance of a DirectPlayLobby object. This function attempts to initialize a DirectPlayLobby object and set a pointer to it.

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

  **CLASS_E_NOAGGREGATION**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

*lpguidSP*
Reserved for future use; must be set to NULL.

*lplpDPL*
Address of a pointer to be initialized with a valid *IDirectPlayLobby* interface.

*lpUnk*
Address of the containing *IUnknown* interface. This parameter is provided for future compability with COM aggregation features. Presently, however, **DirectPlayLobbyCreate** returns an error if this parameter is anything but NULL.

*lpData*
Extra data needed to create the DirectPlayLobby object. This parameter must be set to NULL.

*dwDataSize*
This parameter must be set to zero.

# Callback Functions

## EnumAddressCallback

```
BOOL WINAPI EnumAddressCallback(REFGUID guidDataType,
    DWORD dwDataSize, LPCVOID lpData,
    LPVOID lpContext);
```

Application-defined callback function for the **IDirectPlayLobby::EnumAddress** method.

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

*guidDataType*
Globally unique identifier (GUID) indicating the type of this data chunk.

*dwDataSize*
Size, in bytes, of the data chunk.

*lpData*
Address of the constant data.

*lpContext*
Context passed to the callback function.

The service provider should examine the GUID in the *guidDataType* parameter and process or store the value specified in *lpData*. Unrecognized values in *guidDataType* can be ignored.

Any pointers returned in a callback function are temporary and are valid only in the body of the callback function. If the application needs to save pointer information, it must allocate memory to hold the data, copy the data, and then store the pointer to this new data. In this function, *lpData* is temporary.

## EnumAddressTypeCallback

```
BOOL WINAPI EnumAddressTypeCallback(
    REFGUID guidDataType, LPVOID lpContext,
    DWORD dwFlags);
```

Application-defined callback function for the **IDirectPlayLobby::EnumAddressTypes** method.

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

*guidDataType*
Globally unique identifier (GUID) indicating the address type. Predefined address types are DPAID_Phone, DPAID_Inet, and DPAID_ComPort. For more information about these address types, see *DirectPlay Address*.

*lpContext*
Context passed to the callback function.

*dwFlags*
Reserved; do not use.

## EnumDPCallback

```
BOOL WINAPI EnumDPCallback(LPGUID lpguidSP,
    LPSTR/LPWSTR lpSPName, DWORD dwMajorVersion,
```

```
    DWORD dwMinorVersion, LPVOID lpContext);
```

Application-defined callback function for the **DirectPlayEnumerate** function. Depending on whether UNICODE is defined or not, the prototype for the callback function will have *lpSPName* defined as either the **LPWSTR** type (for Unicode) or the **LPSTR** type (for ANSI).

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

*lpguidSP*
Address of the unique identifier of the DirectPlay service provider.

*lpSPName*
Address of a string containing the driver description. Depending on whether the UNICODE symbol is defined or not, the parameter will be of the **LPWSTR** type (Unicode) or the **LPSTR** type (ANSI).

*dwMajorVersion* and *dwMinorVersion*
Major and minor version numbers of the driver.

*lpContext*
Address of an application-defined context.

Any pointers returned in a callback function are temporary and are valid only in the body of the callback function. If the application needs to save pointer information, it must allocate memory to hold the data, copy the data, and then store the pointer to this new data. In this function, *lpguidSP* and *lpSPName* are temporary.

# EnumLocalApplicationsCallback

```
BOOL WINAPI EnumLocalApplicationsCallback(
    LPCDPLAPPINFO lpAppInfo, LPVOID lpContext, DWORD dwFlags);
```

Application-defined callback function for the **IDirectPlayLobby::EnumLocalApplications** method.

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

*lpAppInfo*
Address of a read-only **DPLAPPINFO** structure containing information about the application being enumerated.

*lpContext*
Context passed from the **IDirectPlayLobby::EnumLocalApplications** call.

*dwFlags*
Reserved; do not use.

Any pointers returned in a callback function are temporary and are valid only in the body of the callback function. If the application needs to save pointer information, it must allocate memory to hold the data, copy the data, and then

store the pointer to this new data. In this function, *lpAppInfo* is temporary. Also note that the pointers inside the structure specified in the *lpAppInfo* parameter—**lpszAppNameA** and **lpszAppName**—are also temporary.

## EnumPlayersCallback2

```
BOOL WINAPI EnumPlayersCallback2(DPID dpId,
    DWORD dwPlayerType, LPCDPNAME lpName,
    DWORD dwFlags, LPVOID lpContext);
```

Application-defined callback function for the **IDirectPlay2::EnumGroups**, **IDirectPlay2::EnumGroupPlayers**, and **IDirectPlay2::EnumPlayers** methods.

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

*dpId*
ID of the player or group being enumerated.

*dwPlayerType*
Type of player, either DPPLAYERTYPE_GROUP or DPPLAYERTYPE_PLAYER.

*lpName*
Address of a constant **DPNAME** structure containing the name of the player or group.

*dwFlags*
Specifies the flags that were passed in the **IDirectPlay2::EnumGroups**, **IDirectPlay2::EnumGroupPlayers**, or **IDirectPlay2::EnumPlayers** method.

*lpContext*
Address of an application-defined context.

Any pointers returned in a callback function are temporary and are valid only in the body of the callback function. If the application needs to save pointer information, it must allocate memory to hold the data, copy the data, and then store the pointer to this new data. In this function, *lpName* is temporary. Also note that the pointers inside the structure specified in the *lpName* parameter—**lpszShortName** / **lpszShortNameA** and **lpszLongName** / **lpszLongNameA**—are also temporary.

## EnumSessionsCallback2

```
BOOL EnumSessionsCallback2(LPDPSESSIONDESC2 lpThisSD,
    LPDWORD lpdwTimeOut, DWORD dwFlags
    LPVOID lpContext);
```

Application-defined callback function for the **IDirectPlay2::EnumSessions** method.

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

*lpThisSD*
Address of a **DPSESSIONDESC2** structure describing the enumerated session. This parameter will be set to NULL if the enumeration has timed out.

*lpdwTimeOut*
Address of a variable containing the current time-out value. This parameter can be reset when the DPESC_TIMEDOUT flag is returned if you want to wait longer for sessions to reply.

*dwFlags*
Typically, this flag is set to zero.

**DPESC_TIMEDOUT**

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

*lpContext*
Address of an application-defined context.

Any pointers returned in a callback function are temporary and are valid only in the body of the callback function. If the application needs to save pointer information, it must allocate memory to hold the data, copy the data, and then store the pointer to this new data. In this function, *lpThisSD* is temporary. Also note that the pointers inside the structure specified in the *lpThisSD* parameter—**lpszSessionName** / **lpszSessionNameA** and **lpszPassword** / **lpszPasswordA**—are also temporary.

# IDirectPlay2

Applications use the methods of the **IDirectPlay2** interface to create DirectPlay objects and work with system-level variables. (The **IDirectPlay2A** interface is the same as the **IDirectPlay2** interface, except that **IDirectPlay2A** uses ANSI characters, and **IDirectPlay2** uses Unicode.) This section is a reference to the methods of this interface.

The methods of the **IDirectPlay2** interface can be organized into the following groups:

| | |
|---|---|
| **Data management** | **GetGroupData** |
| | **GetPlayerData** |
| | **SetGroupData** |
| | **SetPlayerData** |
| **Group management** | **AddPlayerToGroup** |
| | **CreateGroup** |

| | |
|---|---|
| | **DeletePlayerFromGroup** |
| | **DestroyGroup** |
| | **EnumGroupPlayers** |
| | **EnumGroups** |
| | **GetGroupName** |
| | **SetGroupName** |
| **Initialization** | **Initialize** |
| **Message management** | **GetMessageCount** |
| | **Receive** |
| | **Send** |
| **Player management** | **CreatePlayer** |
| | **DestroyPlayer** |
| | **EnumPlayers** |
| | **GetPlayerAddress** |
| | **GetPlayerCaps** |
| | **GetPlayerName** |
| | **SetPlayerName** |
| **Session management** | **Close** |
| | **EnumSessions** |
| | **GetCaps** |
| | **GetSessionDesc** |
| | **Open** |
| | **SetSessionDesc** |

The **IDirectPlay2** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The **IUnknown** interface supports the following three methods:

**AddRef**

**QueryInterface**

**Release**

# IDirectPlay2::AddPlayerToGroup

```
HRESULT AddPlayerToGroup(DPID idGroup, DPID idPlayer);
```

Adds an existing player to an existing group.

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

  **DPERR_CANTADDPLAYER**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idGroup*
Group ID of the group to be augmented.

*idPlayer*
Player ID of the player to be added to the group.

Groups cannot be added to other groups, but players can be members of multiple groups. DPSYS_ADDPLAYERTOGROUP system message will be generated and sent to all the other players. For a list of system messages, see *Using System Messages*.

See also **IDirectPlay2::CreateGroup**, **IDirectPlay2::DeletePlayerFromGroup**, **DPMSG_ADDPLAYERTOGROUP**

# IDirectPlay2::Close

```
HRESULT Close();
```

Closes a previously opened session.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_NOSESSIONS**

All locally created players will be destroyed, with corresponding DPSYS_DESTROYPLAYERORGROUP system messages sent to other session participants. However, no groups will be destroyed (use **IDirectPlay2::DestroyGroup** to destroy the group). For a list of system messages, see *Using System Messages*.

See also **IDirectPlay2::DestroyPlayer**, **DPMSG_DESTROYPLAYERORGROUP**, **IDirectPlay2::Open**

# IDirectPlay2::CreateGroup

```
HRESULT CreateGroup(LPDPID lpidGroup,
    LPDPNAME lpGroupName, LPVOID lpData,
```

```
    DWORD dwDataSize, DWORD dwFlags);
```

Creates a logical group of players in the current session.

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

  **DPERR_CANTADDPLAYER**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

*lpidGroup*
Address of a variable that will be filled with the DirectPlay group ID. This value is defined by DirectPlay.

*lpGroupName*
Address of a **DPNAME** structure that holds the name of the group. NULL indicates that the group has no initial name.

*lpData*
Address of a block of application-defined remote data to associate with the group ID. NULL indicates that the group has no initial data. The data specified here is assumed to be remote data that will be propagated to all the other applications in the session as if **IDirectPlay2::SetGroupData** were called.

*dwDataSize*
Size, in bytes, of the data block that *lpData* points to.

*dwFlags*
Reserved; do not use.

Messages can be sent to a group, and DirectPlay will forward the message to every player in the group. The group ID returned to the application should be used to identify the group for message passing and data association. Player and group IDs assigned by DirectPlay will always be unique within the session. This method will generate a DPSYS_CREATEPLAYERORGROUP system message that will be sent to all the other players. For a list of system messages, see *Using System Messages*.

The application can associate an initial name with the group at creation by using the **IDirectPlay2::SetGroupName** method. The names in *lpGroupName* are provided for human use only; they are not used internally and need not be unique. The application can also associate initial data with the group creation by using the **IDirectPlay2::SetGroupData** method.

See also **DPNAME**, **DPMSG_CREATEPLAYERORGROUP**, **IDirectPlay2::DestroyGroup**, **IDirectPlay2::EnumGroups**, **IDirectPlay2::EnumGroupPlayers**, **IDirectPlay2::Send**, **IDirectPlay2::SetGroupData**, **IDirectPlay2::SetGroupName**

# IDirectPlay2::CreatePlayer

```
HRESULT CreatePlayer(LPDPID lpidPlayer,
    LPDPNAME lpPlayerName, HANDLE hEvent,
    LPVOID lpData, DWORD dwDataSize, DWORD dwFlags);
```

Creates a local player for the current 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**

*lpidPlayer*
Address of a variable that will be filled with the DirectPlay player ID. This value is defined by DirectPlay.

*lpPlayerName*
Address of a **DPNAME** structure that holds the name of the player. NULL indicates that the player has no initial name information.

*hEvent*
An event object created by the application that will be triggered by DirectPlay when a message addressed to this player is received.

*lpData*
Address of a block of application-defined data to associate with the player ID. NULL indicates that the player has no initial data. The data specified in this parameter is assumed to be remote data that will be propagated to all the other applications in the session, as if **IDirectPlay2::SetPlayerData** were called.

*dwDataSize*
Size, in bytes, of the data block that *lpData* points to.

*dwFlags*
Reserved; do not use.

A single process can have multiple local players that communicate through a DirectPlay object with any number of other local or remote players running on multiple computers. Your application should use the player ID returned to the application to identify the player for message passing and data association. Player and group IDs assigned by DirectPlay will always be unique within the session.

The application can associate an initial name with the player at creation by using the **IDirectPlay2::SetPlayerName** method. The names in *lpPlayerName* are provided for human use only; they are not used internally and need not be unique.

The application can also associate initial data with the player at creation by using the **IDirectPlay2::SetPlayerData** method.

Upon successful completion, this method sends a DPSYS_CREATEPLAYERORGROUP system message to all the other players in the session announcing that a new player has joined the session. For a list of system messages, see *Using System Messages*.

If the application uses a separate thread to retrieve DirectPlay messages, it may supply a synchronization event by using the *hEvent* parameter. This event will be set when this player receives a message. Multiple players can use the same event object specified in *hEvent*.

See also **DPNAME**, **DPMSG_CREATEPLAYERORGROUP**, **IDirectPlay2::DestroyPlayer**, **IDirectPlay2::EnumPlayers**, **IDirectPlay2::Receive**, **IDirectPlay2::Send**, **IDirectPlay2::SetPlayerData**, **IDirectPlay2::SetPlayerName**

## IDirectPlay2::DeletePlayerFromGroup

```
HRESULT DeletePlayerFromGroup(DPID idGroup,
    DPID idPlayer);
```

Removes a player from a group.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idGroup*
Group ID of the group to be adjusted.

*idPlayer*
Player ID of the player to be removed from the group.

A DPSYS_DELETEPLAYERFROMGROUP system message is generated to inform the other players of this change. For a list of system messages, see *Using System Messages*.

See also **IDirectPlay2::AddPlayerToGroup**, **DPMSG_DELETEPLAYERFROMGROUP**

## IDirectPlay2::DestroyGroup

```
HRESULT DestroyGroup(DPID idGroup);
```

Deletes a group from the session. The 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_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idGroup*
The 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. This method will generate a DPSYS_DELETEPLAYERFROMGROUP system message for each player in the group, and then a DPSYS_DESTROYPLAYERORGROUP system message. For a list of system messages, see *Using System Messages*.

See also **IDirectPlay2::CreateGroup**, **DPMSG_DESTROYPLAYERORGROUP**

## IDirectPlay2::DestroyPlayer

```
HRESULT DestroyPlayer(DPID idPlayer);
```

Deletes a player from the 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**

*idPlayer*
Player ID of the player that is being removed from the session.

This method will generate a DPSYS_DELETEPLAYERFROMGROUP system message for each group that the player belongs to, and then a DPSYS_DESTROYPLAYERORGROUP system message. For a list of system messages, see *Using System Messages*.

See also **IDirectPlay2::CreatePlayer**, **DPMSG_DESTROYPLAYERORGROUP**

## IDirectPlay2::EnumGroupPlayers

```
HRESULT EnumGroupPlayers(DPID idGroup,
    LPGUID lpguidInstance,
    LPDPENUMPLAYERSCALLBACK2 lpEnumPlayersCallback2,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates all the players of a group 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**

*idGroup*
ID of the group whose players are to be enumerated.

*lpguidInstance*
DirectPlay session instance of interest. This parameter must be set to NULL unless the DPENUMPLAYERS_SESSION flag is specified.

*lpEnumPlayersCallback2*
Address of the **EnumPlayersCallback2** function to be called for every player in the group.

*lpContext*
Address of an application-defined context that is passed to each enumeration callback.

*dwFlags*
Flag to be passed in the *dwFlags* parameter to the callback function.

**DPENUMPLAYERS_SESSION**

Enumerates the players in the group in the session identified by *lpguidInstance*.

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 instance GUID, to request that a session's host provide its list for enumeration. This method cannot be called from within an **IDirectPlay2::EnumSessions** enumeration. Furthermore, use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay2::EnumSessions** method has been called, and before any calls to the **IDirectPlay2::Close** or **IDirectPlay2::Open** methods.

See also **IDirectPlay2::CreatePlayer**, **IDirectPlay2::DestroyPlayer**, **IDirectPlay2::AddPlayerToGroup**, **IDirectPlay2::DeletePlayerFromGroup**

# IDirectPlay2::EnumGroups

```
HRESULT EnumGroups(LPGUID lpguidInstance,
    LPDPENUMPLAYERSCALLBACK2 lpEnumPlayersCallback2,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the groups available to a session.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_UNSUPPORTED**

*lpguidInstance*
DirectPlay session instance of interest. This parameter must be set to NULL unless the DPENUMPLAYERS_SESSION flag is specified.

*lpEnumPlayersCallback2*
Address of the **EnumPlayersCallback2** function to be called for every group in the session.

*lpContext*
Address of an application-defined context that is passed to each enumeration callback.

*dwFlags*
Flag to be passed in the *dwFlags* parameter to the callback function.

**DPENUMPLAYERS_SESSION**

Enumerates the groups in the session identified by *lpguidInstance*.

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 instance GUID, to request that a session's host provide its list for enumeration. This method cannot be called from within an **IDirectPlay2::EnumSessions** enumeration. Furthermore, use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay2::EnumSessions** method has been called, and before any calls to the **IDirectPlay2::Close** or **IDirectPlay2::Open** methods.

See also **IDirectPlay2::CreateGroup**, **IDirectPlay2::DestroyGroup**, **IDirectPlay2::EnumSessions**

# IDirectPlay2::EnumPlayers

```
HRESULT EnumPlayers(LPGUID lpguidInstance,
    LPDPENUMPLAYERSCALLBACK2 lpEnumPlayersCallback2,
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the players in a session.

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

  **DPERR_EXCEPTION**

  **DPERR_GENERIC**

**DPERR_INVALIDOBJECT**

**DPERR_UNSUPPORTED**

*lpguidInstance*

DirectPlay session instance of interest. This parameter must be set to NULL unless the DPENUMPLAYERS_SESSION flag is specified.

*lpEnumPlayersCallback2*

Address of the **EnumPlayersCallback2** function that will be called for every group in the session.

*lpContext*

Address of an application-defined context that is passed to each enumeration callback.

*dwFlags*

Flags to be passed in the *dwFlags* parameter to the callback function.

**DPENUMPLAYERS_GROUP**

Includes groups in the enumeration of players.

**DPENUMPLAYERS_LOCAL**

Enumerates only those players that were created locally by this DirectPlay object.

**DPENUMPLAYERS_REMOTE**

Enumerates only those players that were created by remote DirectPlay objects.

**DPENUMPLAYERS_SESSION**

Enumerates the players for the session identified by *lpguidInstance*.

By default, this method will enumerate players in the current open session. Groups can also be included in the enumeration by using the DPENUMPLAYERS_GROUP flag. The DPENUMPLAYERS_SESSION flag can be used, along with a session instance GUID, to request that a session's host provide its list for enumeration. This method cannot be called from within an **IDirectPlay2::EnumSessions** enumeration. Furthermore, use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay2::EnumSessions** method has been called, and before any calls to the **IDirectPlay2::Close** or **IDirectPlay2::Open** methods.

See also **IDirectPlay2::CreatePlayer**, **IDirectPlay2::DestroyPlayer**, **IDirectPlay2::EnumSessions**

# IDirectPlay2::EnumSessions

```
HRESULT EnumSessions(LPDPSESSIONDESC2 lpsd,
    DWORD dwTimeout,
    LPDPENUMSESSIONSCALLBACK2 lpEnumSessionsCallback2,
```

```
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates the sessions available to this DirectPlay object.

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

  **DPERR_EXCEPTION**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

*lpsd*

Address of a **DPSESSIONDESC2** structure describing the sessions to be enumerated. Only those sessions that meet the criteria set in this structure will be enumerated. The **guidApplication** member of the **DPSESSIONDESC2** structure should be set to the globally unique identifier (GUID) of the application of interest, or NULL for all applications. The **guidInstance** member may be set to a specific GUID of a session instance if it is known; otherwise, it should be set to NULL to obtain all sessions. If a password is required, then the **lpszPassword** member should be set accordingly.

*dwTimeout*

Total amount of time, in milliseconds, that DirectPlay will wait for replies to the enumeration message (not the time between each enumeration). It is recommended that this parameter be set to zero so DirectPlay can compute the default timeout appropriate for the service provider.

*lpEnumSessionsCallback2*

Address of the **EnumSessionsCallback2** function to be called for each DirectPlay session responding.

*lpContext*

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

*dwFlags*

If this parameter is set to 0, only the available sessions will be enumerated (DPENUMSESSIONS_AVAILABLE).

**DPENUMSESSIONS_AVAILABLE**

Enumerates all sessions this application can join.

**DPENUMSESSIONS_ALL**

Enumerates all active sessions, whether they are available to join or not. Sessions in which the player limit has been reached, new players have been disabled, or joining has been disabled will be enumerated. The application can examine the **dwFlags** member of this structure to determine if the session will allow new applications to join or not.

This method is typically called immediately after the DirectPlay object is created by using the **DirectPlayCreate** function. It cannot be called while connected to a session or after an application has created a session.
**IDirectPlay2::EnumSessions** works by requesting that the service provider

locate one or more hosts on the network and send them an enumeration request. The replies that are received make up the sessions that are enumerated. The amount of time DirectPlay spends waiting for these replies is controlled by the *dwTimeout* parameter. When this time interval has expired, your callback will be notified by using the DPESC_TIMEDOUT flag, and a NULL value will be passed for the *lpThisSD* parameter. At this point, you can continue the enumeration by setting *dwTimeout* to a new value and returning TRUE, or you can cancel the enumeration by returning FALSE. It is recommended that *dwTimeout* be set to 0. In that case, DirectPlay will compute a time-out that is appropriate for the service provider.

Typically, only sessions that can be joined are enumerated. If the DPENUMSESSIONS_ALL flag is specified, sessions will be enumerated even if the creation of new players has been disabled. Note the application will still not be able to join these sessions.

If the application was not started by a lobby, the service provider may display a dialog to obtain information from the user to perform the enumeration. For example, the Microsoft serial service provider will ask for COM port settings, the modem service provider will ask for a phone number, and the Internet service provider will ask for an IP address of the host.

Password-protected sessions will not be enumerated unless you supply a correct password.

See also **DPSESSIONDESC2**, **IDirectPlay2::Open**

## IDirectPlay2::GetCaps

```
HRESULT GetCaps(LPDPCAPS lpDPCaps,
    DWORD dwFlags);
```

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 of a **DPCAPS** structure that will be filled with the capabilities of the DirectPlay object. The **dwSize** member of the **DPCAPS** structure must be filled in before using **IDirectPlay2::GetCaps**.

*dwFlags*
If this parameter is set to 0, the capabilities will be computed for nonguaranteed messaging.

**DPGETCAPS_GUARANTEED**

Retrieves the capabilities for a guaranteed message delivery.

This method returns the capabilities of the current session, while the **IDirectPlay2::GetPlayerCaps** method returns the capabilities of the requested player.

See also **DPCAPS**, **IDirectPlay2::GetPlayerCaps**, **IDirectPlay2::Send**

# IDirectPlay2::GetGroupData

```
HRESULT GetGroupData(DPID idGroup,
    LPVOID lpData, LPDWORD lpdwDataSize,
    DWORD dwFlags);
```

Retrieves an application-specific data block that was associated with a group ID by using **IDirectPlay2::SetGroupData**.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idGroup*
Group ID for which data is being requested.

*lpData*
Address of a buffer where the application-specific group data is to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling the method. After the method returns, this parameter will be set to the size, in bytes, of the group data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size required.

*dwFlags*
If this parameter is set to 0, the remote data will be retrieved.

**DPGET_REMOTE**

Retrieves the current data from the remote shared data space.

**DPGET_LOCAL**

Retrieves the local data set by this application

DirectPlay can maintain two types of group data: local and remote. The application must specify which type of data to retreive. Local data was set by this

DirectPlay object by using the DPSET_LOCAL flag. Remote data might have been set by any application in the session by using the DPSET_REMOTE flag.

See also **IDirectPlay2::SetGroupData**

# IDirectPlay2::GetGroupName

```
HRESULT GetGroupName(DPPID idGroup,
    LPVOID lpData, LPDWORD lpdwDataSize);
```

Returns the name associated with a group.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idGroup*
ID of the group whose name is being requested.

*lpData*
Address of a buffer where the name data is to be written. Set this parameter to NULL to request only the size of data. *lpdwDataSize* will be set to the size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling the method. After the method returns, this parameter will be set to the size, in bytes, of the name data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size that is required.

After the function returns, the pointer *lpData* should be cast to the **DPNAME** structure to read the group name data.

See also **DPNAME**, **IDirectPlay2::SetGroupName**

# IDirectPlay2::GetMessageCount

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

Queries for the number of messages in the receive queue for a specific local player.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

**DPERR_INVALIDPLAYER**

*idPlayer*
ID of the player whose message count is requested. The player must be local.

*lpdwCount*
Address of a variable that will be set to the message count when this method returns.

See also **IDirectPlay2::Receive**

# IDirectPlay2::GetPlayerAddress

```
HRESULT GetPlayerAddress(DPID idPlayer,
    LPVOID lpAddress, LPDWORD lpdwAddressSize);
```

Retrieves the DirectPlay Address for a player. The DirectPlay Address is a network address for a player using a specific service provider.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idPlayer*
Player ID that the address is being requested for.

*lpAddress*
Address of a buffer where the DirectPlay Address is to be written. Set this parameter to NULL to request only the size of data. The *lpdwAddressSize* parameter will be set to the size required to hold the data.

*lpdwAddressSize*
Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the group data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size that is required.

# IDirectPlay2::GetPlayerCaps

```
HRESULT GetPlayerCaps(DPID idPlayer,
    LPDPCAPS lpPlayerCaps, DWORD dwFlags);
```

Retrieves the current capabilities of a specified player.

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

  **DPERR_INVALIDOBJECT**

**DPERR_INVALIDPARAMS**

**DPERR_INVALIDPLAYER**

*idPlayer*
Player ID for which the capabilities should be computed.

*lpPlayerCaps*
Address of a **DPCAPS** structure that will be filled with the capabilities. The **dwSize** member of the **DPCAPS** structure must be filled in before using **IDirectPlay2::GetPlayerCaps**.

*dwFlags*
If this parameter is set to 0, the capabilities will be computed for nonguaranteed messaging.

**DPGETCAPS_GUARANTEED**

Retrieves the capabilities for a guaranteed message delivery.

This method returns the capabilities of the requested player, while the **IDirectPlay2::GetCaps** method returns the capabilities of the current session.

See also **DPCAPS**, **IDirectPlay2::GetCaps**, **IDirectPlay2::Send**

# IDirectPlay2::GetPlayerData

```
HRESULT GetPlayerData(DPID idPlayer,
    LPVOID lpData, LPDWORD lpdwDataSize,
    DWORD dwFlags);
```

Retrieves an application-specific data block that was associated with a player ID by using **IDirectPlay2::SetPlayerData**.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDFLAGS**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idPlayer*
ID of the player for which data is being requested.

*lpData*
Address of a buffer where the application-specific player data is to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the group data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size required.

*dwFlags*
If this parameter is set to 0, the remote data will be retrieved.

**DPGET_REMOTE**

Retrieves the current data from the remote shared data space.

**DPGET_LOCAL**

Retrieves the local data set by this application.

DirectPlay can maintain two types of player data: local and remote. The application must specify which type of data to retrieve. Local data was set by this DirectPlay object by using the DPSET_LOCAL flag. Remote data might have been set by any application in the session by using the DPSET_REMOTE flag.

See also **IDirectPlay2::SetPlayerData**

# IDirectPlay2::GetPlayerName

```
HRESULT GetPlayerName(DPID idPlayer,
    LPVOID lpData, LPDWORD lpdwDataSize);
```

Retrieves the name associated with a player.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idPlayer*
ID of the player whose name is requested.

*lpData*
Address of a buffer where the name data is to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the name data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size required.

After this method returns, the pointer *lpData* should be cast to the **DPNAME** structure to read the group name data.

See also **DPNAME**, **IDirectPlay2::SetPlayerName**

## IDirectPlay2::GetSessionDesc

```
HRESULT GetSessionDesc(LPVOID lpData,
    LPDWORD lpdwDataSize);
```

Retrieves the properties of the current open session.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDOBJECT**

  **DPERR_NOCONNECTION**

*lpData*
Address of a buffer where the session description data is to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the group data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size required.

After this method returns, the pointer *lpData* should be cast to the **DPSESSIONDESC2** structure to read the session description data.

See also **DPSESSIONDESC2**, **IDirectPlay2::EnumSessions**, **IDirectPlay2::Open**

## IDirectPlay2::Initialize

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

This method is provided for compliance with the COM protocol.

- Returns **DPERR_ALREADYINITIALIZED**.

*lpGUID*
Address of the globally unique identifier (GUID) used as the interface identifier.

Because the DirectPlay object is initialized when it is created, this method always returns the DPERR_ALREADYINITIALIZED return value.

See also **IUnknown::AddRef**, **IUnknown::QueryInterface**

## IDirectPlay2::Open

```
HRESULT Open(LPDPSESSIONDESC2 lpsd,
    DWORD dwFlags);
```

Establishes a gaming session instance.

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

  **DPERR_ACTIVEPLAYERS**

  **DPERR_ALREADYINITIALIZED**

  **DPERR_GENERIC**

  **DPERR_INVALIDFLAGS**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_UNAVAILABLE**

  **DPERR_UNSUPPORTED**

  **DPERR_USERCANCEL**

*lpsd*
Address of the **DPSESSIONDESC2** structure describing the session to be created or joined.

*dwFlags*
One of the following flags:

**DPOPEN_CREATE**
Creates a new instance of a gaming session.

**DPOPEN_JOIN**
Joins an existing instance of a gaming session.

An application can either create a new session (which other remote applications join) or join an existing session. Your application must call **IDirectPlay2::Open** before any local players are created. Before an application can join an existing session, it should use **IDirectPlay2::EnumSessions** to obtain a list of what sessions can be joined and their session descriptions. Attempting to join a session where new players are disabled, joining is disabled, or the player limit has been reached will result in a DPERR_UNAVAILABLE error.

See also **DPSESSIONDESC2**, **IDirectPlay2::Close**, **IDirectPlay2::EnumSessions**

# IDirectPlay2::Receive

```
HRESULT Receive(LPDPID lpidFrom, LPDPID lpidTo,
    DWORD dwFlags, LPVOID lpData, LPDWORD lpdwDataSize);
```

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_INVALIDPLAYER**

  **DPERR_NOMESSAGES**

*lpidFrom*

Address of a variable that will be set to the sender's player ID when this method returns. If the DPRECEIVE_FROMPLAYER flag is specified, this variable must be initialized with the player ID before calling this method.

*lpidTo*

Address of a variable that will be set to the receiver's player ID when this method returns. If the DPRECEIVE_TOPLAYER flag is specified, this variable must be initialized with the player ID before calling this method.

*dwFlags*

One or more of the following control flags can be set. Both DPRECEIVE_TOPLAYER and DPRECEIVE_FROMPLAYER can be specified, in which case this method returns whichever message is encountered first.

**DPRECEIVE_ALL**

Returns the first available message. This is the default.

**DPRECEIVE_FROMPLAYER**

Returns the first message from the player ID that the *lpidFrom* parameter points to. System messages come from player ID DPID_SYSMSG.

**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 *lpidTo* parameter points to.

*lpData*

Address of a buffer where the message data is to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the size required to hold the data. If the message came from player ID

DPID_SYSMSG, the application should cast *lpData* to **DPMSG_GENERIC** and check the **dwType** member to see what type of system message it is before processing it.

*lpdwDataSize*

Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the group data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the buffer size required. The message order in the receive queue can change between calls to **IDirectPlay2::Receive**. Therefore, it is possible to get a DPERR_BUFFERTOOSMALL error again even after the application has allocated the memory requested from the previous call to **IDirectPlay2::Receive**. It is best to keep reallocating memory until a DPERR_BUFFERTOOSMALL error is not received.

Any message received from player ID DPID_SYSMSG is a system message generated by the host. In those cases, the *lpData* of system messages should be cast to **DPMSG_GENERIC** and the **dwType** member should be examined to see what specific system message it is.

Messages that were sent to player ID DPID_SYSMSG as a way to broadcast them to all players or to a group ID to send them to all the players in the group still appear to come from the sending player ID. An application will receive only messages directed to a local player. A player cannot receive a message in which the values pointed to by *lpidFrom* and *lpidTo* are equal.

If DPSESSION_NOMESSAGEID is specified in the session description, the *lpidFrom* and *lpidTo* parameters are meaningless.

All the service providers shipped with DirectPlay perform integrity checks on the data to protect against corruption. Any message received will be verified, and if data corruption is detected, it will either be thrown away (if it was sent nonguaranteed) or it will be retransmitted (if it was sent guaranteed).

See also **DPMSG_GENERIC**, **IDirectPlay2::Send**

# IDirectPlay2::Send

```
HRESULT Send(DPID idFrom, DPID idTo, DWORD dwFlags,
    LPVOID lpData, DWORD dwDataSize);
```

Sends messages to other players, to a group 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_INVALIDPARAMS**

**DPERR_INVALIDPLAYER**

**DPERR_SENDTOOBIG**

*idFrom*

ID of the sending player. The player ID must correspond to one of the local players on this computer.

*idTo*

ID of the player to send the message to, the group ID of the group of players to send the message to, or DPID_ALLPLAYERS to send the message to all players in the session. If the DPSEND_OPENSTREAM or DPSEND_CLOSESTREAM flags are used, then this parameter must be a player ID.

*dwFlags*

Indicates how the message should be sent. If this parameter is set to 0, the message is sent nonguaranteed and at normal priority. DPSEND_OPENSTREAM and DPSEND_CLOSESTREAM are used to let DirectPlay and the service provider know that there will be a large number of guaranteed messages being sent to the player specified in *idTo*. If some efficiency can be gained by not opening and closing a guaranteed communication pipe (stream) for each message to that player, then the service provider may leave the stream open until the **IDirectPlay2::Send** method with a DPSEND_CLOSESTREAM flag for that player ID is called. DPSEND_OPENSTREAM and DPSEND_CLOSESTREAM are valid only for messages where *idTo* is a valid player ID. It is not required that a service provider support DPSEND_OPENSTREAM and DPSEND_CLOSESTREAM.

**DPSEND_GUARANTEED**

Sends the message by using a guaranteed method of delivery if it is available.

**DPSEND_HIGHPRIORITY**

Sends the message high-priority. This will force the message to the front of the send queue for immediate transmission. The message will also go to the top of the receiving application's receive buffer.

**DPSEND_OPENSTREAM**

Indicates an optimization hint to the service provider that there will be a lot of guaranteed messages being sent to this player.

**DPSEND_CLOSESTREAM**

Indicates that there will no longer be a lot of guaranteed messages being sent to this player.

*lpData*

Address of the data being sent. Set this parameter to NULL if there is no actual message to send. An application can do this if the DPSEND_OPENSTREAM or DPSEND_CLOSESTREAM flag is specified.

*dwDataSize*
Length of the data being sent.

To send a message to another player, specify the target player's ID. To send a message to a group of players, send the message to the ID assigned to the group. To send messages to the entire session, specify the DPID_ALLPLAYERS player ID. You cannot use the **IDirectPlay2::Send** method inside an **IDirectDrawSurface2::Lock** / **IDirectDrawSurface2::Unlock** or **IDirectDrawSurface2::GetDC** / **IDirectDrawSurface2::ReleaseDC** method pair.

A player cannot send a message to itself. If a player sends a message to a group that it is part of or to DPID_ALLPLAYERS, it will not receive a copy of that message.

If DPSESSION_NOMESSAGEID was specified in the session description, it is possible for a player to receive a message that it sent to a group. Because there is no DirectPlay message ID header on the message (indicating who sent the message), it cannot filter out messages from itself when the service provider implements group sends, and the application will need to be able to evaluate this based on the content of the message.

When DPSESSION_NOMESSAGEID is used, the message is sent to one of the local players on the target computer.

Messages can be sent guaranteed or nonguaranteed. By default, messages are sent nonguaranteed, which means that DirectPlay does not verify that the message reached the intended recipient. Sending a guaranteed message takes a minimum of two to three times longer than a nonguaranteed message. You should try to minimize the number of times your application sends guaranteed messages, and your application should be able to tolerate lost messages.

All the service providers shipped with DirectPlay perform integrity checks on the data to protect against corruption. Any message received will be verified, and if data corruption is detected, it will either be thrown away (if it was sent nonguaranteed) or it will be retransmitted (if it was sent guaranteed).

**Note**

In this version of DirectPlay, DPSEND_GUARANTEED will guarantee delivery only if the service provider supports it. An application can find out if delivery will be guaranteed by calling the **IDirectPlay2::GetCaps** method and checking for the DPCAPS_GUARANTEEDSUPPORTED flag. If this flag is not set, then the DPSEND_GUARANTEED flag will be ignored and the message will be sent nonguaranteed. The next version of DirectPlay will implement guaranteed delivery on nonguaranteed service providers so the guaranteed supported CAPS flag will always be present. If the application implements its own method of guaranteeing message delivery, it must be sure not to use the DPSEND_GUARANTEED flag. When testing the performance of your application, it is important to know if the service provider supports guaranteed messaging or not. If it does not, then every place you specified DPSEND_GUARANTEED will run slower with the next version of DirectPlay.

See also **IDirectPlay2::Receive**

# IDirectPlay2::SetGroupData

```
HRESULT SetGroupData(DPID idGroup,
    LPVOID lpData, DWORD dwDataSize,
    DWORD dwFlags);
```

Associates an application-specific data block with a group ID.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idGroup*
Group ID for which data is being set.

*lpData*
Address of the data to be set. Set to NULL to clear any existing group data.

*dwDataSize*
Size of the data buffer.

*dwFlags*
If this parameter is set to 0, the remote group data will be set and propagated using nonguaranteed messaging.

**DPSET_REMOTE**

This data is for use by all the applications, and will be propagated to all the other applications in the session.

**DPSET_LOCAL**

This data is for local use only and will not be propagated.

**DPSET_GUARANTEED**

Propagates the data by using guaranteed messaging (if available). This flag can only be used with DPSET_REMOTE.

DirectPlay can maintain two types of group data: local and remote. Local data is available only to the application on the local computer. Remote data is propagated to all the other applications in the session. A DPSYS_SETPLAYERORGROUPDATA system message will be sent to all the other players notifying them of the change unless DPSESSION_NODATAMESSAGES is set in the session description. It is safe to store pointers to resources in the local data; the local data block is available (in the **DPMSG_DESTROYPLAYERORGROUP** system message) when the group is being destroyed, so the application can free those resources. For a list of system messages, see *Using System Messages*.

See also **DPMSG_SETPLAYERORGROUPDATA**, **IDirectPlay2::GetGroupData**, **IDirectPlay2::Send**

# IDirectPlay2::SetGroupName

```
HRESULT SetGroupName(DPID idGroup,
    LPDPNAME lpGroupName, DWORD dwFlags);
```

Sets the name of a group after it has been created. A DPSYS_SETPLAYERORGROUPNAME system message will be sent to all the other players notifying them of the change unless DPSESSION_NODATAMESSAGES is set in the session description. For a list of system messages, see *Using System Messages*.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_INVALIDPLAYER**

*idGroup*

ID of the group for which the name is being set.

*lpGroupName*

Address of a **DPNAME** structure containing the name information for the group. Set this parameter to NULL if the group has no name information.

*dwFlags*

If this parameter is set to 0, the name will be propagated to all the remote systems by using nonguaranteed message passing.

**DPSET_GUARANTEED**

Propagates the data using guaranteed messaging (if available).

See also **DPNAME**, **DPMSG_SETPLAYERORGROUPNAME**, **IDirectPlay2::GetGroupName**, **IDirectPlay2::Send**

## IDirectPlay2::SetPlayerData

```
HRESULT SetPlayerData(DPID idPlayer, LPVOID lpData,
    DWORD dwDataSize, DWORD dwFlags);
```

Associates an application-specific data block with a player ID.

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

  **DPERR_INVALIDFLAGS**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idPlayer*
ID of the player for which data is being set.

*lpData*
Address of the data to be set. Set this parameter to NULL to clear out any existing player data.

*dwDataSize*
Size of the data buffer

*dwFlags*
If this parameter is set to 0, the remote player data will be set and propagated by using nonguaranteed messaging.

**DPSET_REMOTE**

This data is for use by all the applications, and will be propagated to all the other applications in the session.

**DPSET_LOCAL**

This data is for local use only and will not be propagated.

**DPSET_GUARANTEED**

Propagates the data by using guaranteed messaging (if available). This flag can only be used with DPSET_REMOTE.

DirectPlay can maintain two types of player data: local and remote. Local data is available only to the application on the local computer. Remote data is propagated to all the other applications in the session. A DPSYS_SETPLAYERORGROUPDATA system message will be sent to all the other players notifying them of the change unless DPSESSION_NODATAMESSAGES is set in the session description. It is safe to store pointers to resources in the local data; the local data block is available (in

the **DPMSG_DESTROYPLAYERORGROUP** system message) when the player is being destroyed, so the application can free those resources. For a list of system messages, see *Using System Messages*.

See also **DPMSG_SETPLAYERORGROUPDATA**, **IDirectPlay2::GetPlayerData**, **IDirectPlay2::Send**

## IDirectPlay2::SetPlayerName

```
HRESULT SetPlayerName(DPID idPlayer,
    LPDPNAME lpPlayerName, DWORD dwFlags);
```

Sets the name of the player after it has been changed. A DPSYS_SETPLAYERORGROUPNAME system message will be sent to all the other players notifying them of the change unless DPSESSION_NODATAMESSAGES is set in the session description. For a list of system messages, see *Using System Messages*.

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

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPLAYER**

*idPlayer*
ID of the player for which data is being sent.

*lpPlayerName*
Address of a **DPNAME** structure containing the name information for the player. Set this parameter to NULL if the player has no name information.

*dwFlags*
If this parameter is set to 0, the name will be propagated to all the remote systems using nonguaranteed message passing.

**DPSET_GUARANTEED**
Propagates the data by using guaranteed messaging (if available).

See also **DPNAME**, **DPMSG_SETPLAYERORGROUPNAME**, **IDirectPlay2::GetPlayerName**, **IDirectPlay2::Send**

## IDirectPlay2::SetSessionDesc

```
HRESULT SetSessionDesc(LPDPSESSIONDESC2 lpSessDesc,
    DWORD dwFlags);
```

This method is currently not supported.

Changes the properties of the current session. This method works only when called on the computer that is the host of the session.

- Returns **DPERR_UNSUPPORTED**.

*lpSessDesc*
Address of the session description structure containing the new settings.

*dwFlags*
No flags are currently used by this method.

See also **DPSESSIONDESC2**, **IDirectPlay2::GetSessionDesc**

# IDirectPlayLobby

Applications use the methods of the **IDirectPlayLobby** interface to manage applications and their associated data. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirectPlayLobby Interface*.

| | |
|---|---|
| **Address management** | **CreateAddress** |
| | **EnumAddress** |
| | **EnumAddressTypes** |
| **Application management** | **Connect** |
| | **EnumLocalApplications** |
| | **RunApplication** |
| **Data management** | **GetConnectionSettings** |
| | **ReceiveLobbyMessage** |
| | **SendLobbyMessage** |
| | **SetConnectionSettings** |
| | **SetLobbyMessageEvent** |

# IDirectPlayLobby::Connect

```
HRESULT WINAPI Connect(DWORD dwFlags,
    LPDIRECTPLAY2 FAR *lplpDP, IUnknown FAR *pUnk);
```

Connects an application to a session by using the connection data supplied by the lobby client in the **IDirectPlayLobby::RunApplication** method, or by calling the **IDirectPlayLobby::SetConnectionSettings** method.

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

  **CLASS_E_NOAGGREGATION**

  **DPERR_INVALIDFLAGS**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

**DPERR_INVALIDPARAMS**

**DPERR_NOTLOBBIED**

**DPERR_OUTOFMEMORY**

*dwFlags*
Reserved; must be zero.

*lplpDP*
Address of a pointer to be initialized with a valid inteface—either *IDirectPlay2* (if called on *IDirectPlayLobby*) or **IDirectPlay2A** (if called on **IDirectPlayLobbyA**).

*pUnk*
Address of the containing *IUnknown* interface. This parameter is provided for future compability with COM aggregation features. Presently, however, **IDirectPlayLobby::Connect** returns an error if this parameter is anything but NULL.

Executing this method successfully creates a DirectPlay object with the correct service provider, and opens the correct session without asking the user to fill in any dialog boxes. If this method fails with the DPERR_NOTLOBBIED error value, then the application should perform the normal steps of calling **DirectPlayEnumerate**, **DirectPlayCreate**, **IDirectPlay2::EnumSessions** and **IDirectPlay2::Open**. If it fails on any other error value, then there is a problem connecting to the session.

Before calling this method, the application can examine the connection settings that will be used to start the application by using the **IDirectPlayLobby::GetConnectionSettings** method. The application then can modify these settings and set them by using the **IDirectPlayLobby::SetConnectionSettings** method. The application should pay particular attention to the **DPSESSIONDESC2** structure to ensure that the proper session properties are set, especially **dwFlags**, **dwMaxPlayers**, and the **dwUser** members.

See also **DirectPlayCreate**

## IDirectPlayLobby::CreateAddress

```
HRESULT CreateAddress(REFGUID guidSP,
    REFGUID guidDataType, LPCVOID lpData,
    DWORD dwDataSize, LPVOID lpAddress,
    LPDWORD lpdwAddressSize);
```

Creates a DirectPlay Address, given a service provider-specific network address. The resulting address contains the globally unique identifier (GUID) of the

service provider and data that the service provider can interpret as a network address.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_INVALIDPARAMS**

*guidSP*
Address of the GUID of the service provider. (In C++, it is a reference to the GUID.)

*guidDataType*
Address of a GUID identifying the specific network address type being used. For information about predefined network address types, see *DirectPlay Address*. (In C++, it is a reference to the GUID.)

*lpData*
Address of a buffer containing the specific network address.

*dwDataSize*
Size, in bytes, of the network address in *lpData*.

*lpAddress*
Address of a buffer in which the constructed DirectPlay Address is to be written.

*lpdwAddressSize*
Address of a variable containing the size of the DirectPlay Address buffer. Before calling this method, the service provider must initialize *lpdwAddressSize* to the size of the buffer. After the method has returned, this parameter will contain the number of bytes written to *lpAddress*. If the buffer was too small (DPERR_BUFFERTOOSMALL), this parameter will be set to the size required to store the DirectPlay Address.

See also **IDirectPlayLobby::EnumAddress**

# IDirectPlayLobby::EnumAddress

```
HRESULT EnumAddress(LPDPENUMADDRESS lpEnumAddressCallback,
    LPCVOID lpAddress, DWORD dwAddressSize,
    LPVOID lpContext);
```

Parses out chunks from the DirectPlay Address buffer.

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

  **DPERR_EXCEPTION**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

*lpEnumAddressCallback*
Address of a **EnumAddressCallback** function that will be called for each information chunk in the DirectPlay Address.

*lpAddress*
Address of the start of the DirectPlay Address buffer.

*dwAddressSize*
Size of the DirectPlay Address.

*lpContext*
Context that will be passed to the callback function.

See also **DirectPlay Address**, **IDirectPlayLobby::CreateAddress**

## IDirectPlayLobby::EnumAddressTypes

```
HRESULT EnumAddressTypes(
    LDDPLENUMADDRESSTYPESCALLBACK lpEnumAddressTypeCallback,
    REFGUID guidSP, LPVOID lpContext,
    DWORD dwFlags);
```

Enumerates all the address types that a given service provider needs to build the DirectPlay Address.

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

  **DPERR_EXCEPTION**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

*lpEnumAddressTypeCallback*
Address of the **EnumAddressTypeCallback** function that will be called for each address type for a service provider. If the service provider takes no address type, the callback will not be called.

*guidSP*
Address of the GUID of the service provider whose address types are to be enumerated. (In C++, it is a reference to the GUID.)

*lpContext*
Context that will be passed to the callback function.

*dwFlags*
Reserved; must be zero.

See also *DirectPlay Address*, **IDirectPlayLobby::CreateAddress**

## IDirectPlayLobby::EnumLocalApplications

```
HRESULT EnumLocalApplications(
    LPDPENUMLOCALAPPLICATIONS lpEnumLocalAppCallback,
```

```
    LPVOID lpContext, DWORD dwFlags);
```

Enumerates what applications are registered with DirectPlay.

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

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

*lpEnumLocalAppCallback*
Address of the **EnumLocalApplicationsCallback** function that will be called for each enumerated application.

*lpContext*
Context passed to the callback function.

*dwFlags*
Reserved; must be zero.

See also **DPLAPPINFO**

# IDirectPlayLobby::GetConnectionSettings

```
HRESULT GetConnectionSettings(DWORD dwAppID,
    LPVOID lpData, LPDWORD lpdwDataSize);
```

Retrieves the **DPLCONNECTION** structure that contains all the information needed to start and connect an application. The data returned is the same data that was passed to the **IDirectPlayLobby::RunApplication** method by the lobby client, or set by calling the **IDirectPlayLobby::SetConnectionSettings** method.

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

  **DPERR_BUFFERTOOSMALL**

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_NOTLOBBIED**

  **DPERR_OUTOFMEMORY**

*dwAppID*
Identifies which application's connection settings to retrieve when called from a lobby client (that communicates with several applications). When called from an

application (that only communicates with one lobby client), this parameter must be zero. This ID number is obtained from **IDirectPlayLobby::RunApplication**.

*lpData*
Address of a buffer in which the connection settings are to be written. Set this parameter to NULL to request only the size of data. The *lpdwDataSize* parameter will be set to the minimum size required to hold the data.

*lpdwDataSize*
Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the data. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the minimum buffer size required.

The *lpData* member should be cast to the **LPDPLCONNECTION** structure when the function returns to read the data from it.

See also **DPLCONNECTION**, **IDirectPlayLobby::RunApplication**, **IDirectPlayLobby::SetConnectionSettings**

## IDirectPlayLobby::ReceiveLobbyMessage

```
HRESULT ReceiveLobbyMessage(DWORD dwFlags,
    DWORD dwAppID, LPDWORD lpdwMessageFlags,
    LPVOID lpData, LPDWORD lpdwDataSize);
```

Retrieves the message sent between a lobby client application and a DirectPlay application. Messages are queued, so there is no danger of losing data if it is not read in time.

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

  **DPERR_APPNOTSTARTED**

  **DPERR_BUFFERTOOSMALL**

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_NOMESSAGES**

  **DPERR_OUTOFMEMORY**

*dwFlags*
Reserved; must be zero.

*dwAppID*
Identifies which application's message to retrieve when called from a lobby client (that communicates with several applications). When called from an application (that communicates only with one lobby client), this parameter must be set to

zero. This ID number is obtained by using the **IDirectPlayLobby::RunApplication** method.

*lpdwMessageFlags*

Flags indicating what type of message is being returned.

**DPLAD_SYSTEM**

Indicates that this is a system message informing the application of an event. To determine what type of event occurred, cast the *lpData* pointer to the **DPLMSG_GENERIC** system message and switch on the **dwType** member to see exactly what type of system message it is.

*lpData*

Address of a buffer in which the message is to be written. Set this parameter to NULL to request only the size of message. The *lpdwDataSize* parameter will be set to the minimum size required to hold the message.

*lpdwDataSize*

Address of a variable that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the size, in bytes, of the message. If the buffer was too small (DPERR_BUFFERTOOSMALL), then this parameter will be set to the minimum buffer size required.

See also **IDirectPlayLobby::RunApplication**, **IDirectPlayLobby::SendLobbyMessage**

# IDirectPlayLobby::RunApplication

```
HRESULT RunApplication(DWORD dwFlags,
    LPDWORD lpdwAppID, LPDPLCONNECTION lpConn,
    HANDLE hReceiveEvent);
```

Starts an application and passes to it all the information necessary to connect it to a session. This method is used by a lobby client.

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

  **DPERR_CANTCREATEPROCESS**

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

  **DPERR_UNKNOWNAPPLICATION**

*dwFlags*

Reserved; must be zero.

*lpdwAppId*
Address of a variable that will be filled with an ID identifying the application that was started. The lobby client must save this application ID for use on with any calls to the **IDirectPlayLobby::SendLobbyMessage** and **IDirectPlayLobby::ReceiveLobbyMessage** methods.

*lpConn*
Address of a **DPLCONNECTION** structure that contains all the information necessary to specify which application to start and how to get it connected to a session instance without displaying any user dialog boxes.

*hReceiveEvent*
Specifies a synchronization event that will be set when a lobby message is received. This event can be changed later by using the **IDirectPlayLobby::SetLobbyMessageEvent** method.

This method will return after the application process has been created. The lobby client will receive a system message indicating the status of the application. If the lobby client is starting an application that will be hosting a session, it should wait until it receives a DPLSYS_SESSIONCREATED system message before starting the other applications that will be joining the session. If the application was unable to create or join a session, a DPLSYS_DPLAYCONNECTFAILED message will be generated. The lobby client will also receive a DPLSYS_CONNECTIONSETTINGSREAD system message when the application has read the connection settings and a DPLSYS_APPTERMINATED system message when the application terminates.

It is important that the lobby client not release its *IDirectPlayLobby* interface before it receives a DPLSYS_CONNECTIONSETTINGSREAD system message. The lobby client can either check **IDirectPlayLobby::ReceiveLobbyMessage** in a loop until it is received, or supply a synchronization event.

See also **IDirectPlayLobby::ReceiveLobbyMessage**, **IDirectPlayLobby::GetConnectionSettings**, **IDirectPlayLobby::SetLobbyMessageEvent**

## IDirectPlayLobby::SendLobbyMessage

```
HRESULT SendLobbyMessage( DWORD dwFlags,
    DWORD dwAppID, LPVOID lpData,
    DWORD dwDataSize);
```

Sends a message between the application and the lobby client.

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

  **DPERR_APPNOTSTARTED**

  **DPERR_BUFFERTOOLARGE**

**DPERR_GENERIC**

**DPERR_INVALIDINTERFACE**

**DPERR_INVALIDOBJECT**

**DPERR_INVALIDPARAMS**

**DPERR_OUTOFMEMORY**

**DPERR_TIMEOUT**

*dwFlags*
Reserved; must be zero.

*dwAppID*
Identifies which application to send a message to when called from a lobby client (that communicates with several applications). When called from an application (that communicates with only one lobby client), this parameter must be zero. This ID is obtained by using the **IDirectPlayLobby::RunApplication** method.

*lpData*
Address of the buffer containing the message to send.

*dwDataSize*
Size, in bytes, of the buffer.

See also **IDirectPlayLobby::RunApplication**, **IDirectPlayLobby::ReceiveLobbyMessage**

## IDirectPlayLobby::SetConnectionSettings

```
HRESULT SetConnectionSettings(DWORD dwFlags,
    DWORD dwAppID, LPDPLCONNECTION lpConn);
```

Modifies the **DPLCONNECTION** structure, which contains all the information needed to start and connect an application.

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

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

*dwFlags*
Reserved; must be zero.

*dwAppID*
When called from a lobby client (that communicates with several applications), this parameter identifies which application's connection settings to retrieve.

When called from an application (that communicates with only one lobby client), this parameter must be zero. This ID is obtained by using the **IDirectPlayLobby::RunApplication** method.

*lpConn*

Address of a **DPLCONNECTION** structure that contains all the information necessary to specify which application to start and how to get it connected to a session instance without displaying any user dialog boxes.

See also **IDirectPlayLobby::GetConnectionSettings**

## IDirectPlayLobby::SetLobbyMessageEvent

```
HRESULT SetLobbyMessageEvent(DWORD dwFlags,
    DWORD dwAppID, HANDLE hReceiveEvent);
```

Registers an event that will be set when a lobby message is received. The application must call this method if it needs to synchronize with messages. The lobby client can call this method to change the events specified in the call to the **IDirectPlayLobby::RunApplication** method.

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

  **DPERR_GENERIC**

  **DPERR_INVALIDINTERFACE**

  **DPERR_INVALIDOBJECT**

  **DPERR_INVALIDPARAMS**

  **DPERR_OUTOFMEMORY**

*dwFlags*

Reserved; must be zero.

*dwAppID*

Identifies which application the event is associated with when called from a lobby client (that communicates with several applications). When called from an application (that communicates with only one lobby client), this parameter must be zero. This ID number is obtained from **IDirectPlayLobby::RunApplication**.

*hReceiveEvent*

Event handle to be set when a message is received.

See also **IDirectPlayLobby::ReceiveLobbyMessage**, **IDirectPlayLobby::SendLobbyMessage**

# Structures

## DPCAPS

```
typedef struct {
```

```
    DWORD dwSize;
    DWORD dwFlags;
    DWORD dwMaxBufferSize;
    DWORD dwMaxQueueSize;
    DWORD dwMaxPlayers;
    DWORD dwHundredBaud;
    DWORD dwLatency;
    DWORD dwMaxLocalPlayers;
    DWORD dwHeaderLength;
    DWORD dwTimeout;
} DPCAPS, FAR *LPDPCAPS;
```

Contains the capabilities of a DirectPlay object after a call to the **IDirectPlay2::GetCaps** or **IDirectPlay2::GetPlayerCaps** methods. Any of these capabilities can differ depending on whether guaranteed or nonguaranteed capabilities are requested. This structure is read-only.

**dwSize**
Size, in bytes, of this structure. Your application must set this member before it uses this structure; otherwise, an error will result.

**dwFlags**
Indicates the properties of the DirectPlay object.

**DPCAPS_GROUPOPTIMIZED**
Indicates that the service provider bound to this DirectPlay object can optimize group (multicast) messaging.

**DPCAPS_GUARANTEEDOPTIMIZED**
Indicates that the service provider bound to this DirectPlay object supports guaranteed message delivery.

**DPCAPS_GUARANTEEDSUPPORTED**
Indicates that the DirectPlay object supports guaranteed message delivery, either because the service provider supports it or because DirectPlay implements it on a nonguaranteed service provider.

**DPCAPS_ISHOST**
Indicates that the DirectPlay object created by the calling application is the session host.

**DPCAPS_KEEPALIVEOPTIMIZED**
The service provider can detect when the connection to the session has been lost.

**dwMaxBufferSize**
Maximum number of bytes that can be sent in a single packet by this service provider. Larger messages will be sent by using more than one packet.

**dwMaxQueueSize**
This member is no longer used.

**dwMaxPlayers**
Maximum number of local and remote players supported in a session by this DirectPlay object.

**dwHundredBaud**
Bandwidth specified in multiples of 100 bits per second. For example, a value of 24 specifies 2400 baud.

**dwLatency**
Estimate of latency by the 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. Latency can differ depending on whether the application uses guaranteed or nonguaranteed message delivery.

**dwMaxLocalPlayers**
Maximum number of local players supported in a session.

**dwHeaderLength**
Size, in bytes, of the header that will be added to player messages by this DirectPlay object. Note that the header size depends on which service provider is in use.

**dwTimeout**
Service provider's suggested timeout value. By default, DirectPlay will use this timeout value when waiting for replies to messages.

See also **IDirectPlay2::Send**

# DPCOMPORTADDRESS

```
typedef struct DPCOMPORTADDRESS{
    DWORD dwComPort;
    DWORD dwBaudRate;
    DWORD dwStopBits;
    DWORD dwParity;
    DWORD dwFlowControl;
} DPCOMPORTADDRESS;

typedef DPCOMPORTADDRESS FAR* LPDPCOMPORTADDRESS;
```

Contains information about the configuration of the COM port.

**dwComPort**
Indicates the number of the COM port to use. The value for this member can be 1, 2, 3, or 4.

**dwBaudRate**
Indicates the baud of the COM port. The value for this member can be one of the following:

CBR_110 CBR_300 CBR_600

| | | |
|---|---|---|
| CBR_1200 | CBR_2400 | CBR_4800 |
| CBR_9600 | CBR_14400 | CBR_19200 |
| CBR_38400 | CBR_56000 | CBR_57600 |
| CBR_115200 | CBR_128000 | CBR_256000 |

**dwStopBits**
Indicates the number of stop bits. The value for this member can be ONESTOPBIT, ONE5STOPBITS, or TWOSTOPBITS.

**dwParity**
Indicates the parity used on the COM port. The value for this member can be NOPARITY, ODDPARITY, EVENPARITY, or MARKPARITY.

**dwFlowControl**
Indicates the method of flow control used on the COM port. The following values can be used for this member:

| | |
|---|---|
| **DPCPA_DTRFLOW** | Indicates hardware flow control with DTR. |
| **DPCPA_NOFLOW** | Indicates no flow control. |
| **DPCPA_RTSDTRFLOW** | Indicates hardware flow control with RTS and DTR. |
| **DPCPA_RTSFLOW** | Indicates hardware flow control with RTS. |
| **DPCPA_XONXOFFFLOW** | Indicates software flow control (xon/xoff). |

The constants that define baud, stop bits, and parity are defined in Winbase.h.

# DPLAPPINFO

```
typedef struct DPLAPPINFO {
DWORD dwSize;
GUID guidApplication;
union
{
    LPSTR  lpszAppNameA;
    LPWSTR lpszAppName;
};
} DPLAPPINFO, * LPDPLAPPINFO;
```

Contains information about the application from the registry and is passed to the **IDirectPlayLobby::EnumLocalApplications** callback function.

**dwSize**
Size, in bytes, of this structure. Your application must set this member before it uses this structure; otherwise, an error will result.

**guidApplication**
Globally unique identifier (GUID) of the application.

**lpszAppNameA**, **lpszAppName**
Name of the application in ANSI or Unicode, depending on what interface is in use.

# DPLCONNECTION

```
typedef struct {
    DWORD            dwSize;
    DWORD            dwFlags;
    LPDPSESSIONDESC2 lpSessionDesc;
    LPDPNAME         lpPlayerName;
    GUID             guidSP;
    LPVOID           lpAddress;
    DWORD            dwAddressSize;
} DPLCONNECTION, *LPDPLCONNECTION;
```

Contains the information needed to connect an application to a session.

**dwSize**
Indicates the size, in bytes, of this structure. Your application must set this member before it uses this structure; otherwise, an error will result.

**dwFlags**
Indicates how the connection should be made.

**DPLCONNECTION_CREATESESSION**
Create a new session as described in the session description.

**DPLCONNECTION_JOINSESSION**
Join the existing session as described in the session description.

**lpSessionDesc**
Address of a **DPSESSIONDESC2** structure describing the session to be created or the session to join.

**lpPlayerName**
Address of a **DPNAME** structure with the name that the player should be created with. This will be the name of the person registered in the lobby. The application can ignore this name.

**guidSP**
Globally unique identifier (GUID) of the service provider to use to connect to the session.

**lpAddress**
Address of a DirectPlay Address that contains the information that the service provider needs to connect to a session.

**dwAddressSize**
Size, in bytes, of the address data.

# DPNAME

```
typedef struct {
    DWORD dwSize;
    DWORD dwFlags;
    union {
        LPWSTR lpszShortName;
        LPSTR  lpszShortNameA;
    };
    union {
        LPWSTR lpszLongName;
        LPSTR  lpszLongNameA;
    };
} DPNAME,FAR *LPDPNAME;
```

Contains name information for a DirectPlay entity, such as a player or group.

**dwSize**
Size, in bytes, of this structure. Your application must set this member before it uses this structure; otherwise, an error will result.

**dwFlags**
Structure-specific flags. Currently set to zero.

**lpszShortName** and **lpszLongName**
Addresses of Unicode strings containing the short (friendly) and long (formal) names of a player or group. Use these members only if the *IDirectPlay2* interface is in use.

**lpszShortNameA** and **lpszLongNameA**
Addresses of ANSI strings containing the short (friendly) and long (formal) names of a player or group. Use these members only if the **IDirectPlay2A** interface is in use.

See also **IDirectPlay2::CreateGroup**, **IDirectPlay2::CreatePlayer**, **IDirectPlay2::GetGroupName**, **IDirectPlay2::GetPlayerName**, **IDirectPlay2::SetGroupName**, **IDirectPlay2::SetPlayerName**

# DPSESSIONDESC2

```
typedef struct {
    DWORD dwSize;
    DWORD dwFlags;
    GUID  guidInstance;
    GUID  guidApplication;
    DWORD dwMaxPlayers;
    DWORD dwCurrentPlayers;
    union  {
        LPWSTR lpszSessionName;
        LPSTR  lpszSessionNameA;
    };
```

```
    union  {
        LPWSTR lpszPassword;
        LPSTR  lpszPasswordA;
    };
    DWORD dwReserved1;
    DWORD dwReserved2;
    DWORD dwUser1;
    DWORD dwUser2;
    DWORD dwUser3;
    DWORD dwUser4;
} DPSESSIONDESC2, FAR *LPDPSESSIONDESC2;
```

Contains a description of an *IDirectPlay2* session's capabilities. (The **DPSESSIONDESC** structure is no longer used in the **IDirectPlay2** interface.)

**dwSize**

Size of this structure, in bytes. Your application must set this member before it uses this structure; otherwise, an error will result.

**dwFlags**

A combination of the following flags.

**DPSESSION_JOINDISABLED**

No new applications can join this session. Any call to the **IDirectPlay2::Open** method with the DPOPEN_JOIN flag and the globally unique identifier (GUID) of this session instance will cause an error. If this flag is not specified, new remote applications can join the session until the session player limit is reached.

**DPSESSION_KEEPALIVE**

Automatically detect when remote players drop out of the game abnormally. Those players will be deleted from the session. If a temporary network outage caused the loss of the players, they will be informed when they return that they were dropped from the session. For more information, see the description for the DPSYS_SESSIONLOST system message in *Using System Messages* . If this flag is not specified, DirectPlay will not support this feature.

**DPSESSION_MIGRATEHOST**

If the current host exits, the host will attempt to migrate to another computer so that new players can continue to join. If this flag is not specified, the host will not migrate and new players cannot be created.

**DPSESSION_NEWPLAYERSDISABLED**

Indicates that new players cannot be created in the session. Any call to the **IDirectPlay2::CreatePlayer** method by an application in the session will result in an error. Also, new applications cannot join the session. If this flag is not specified, players can be created until the session player limit is reached.

**DPSESSION_NODATAMESSAGES**

Do not send system messages when remote player or group data changes, by using the **IDirectPlay2::SetPlayerData**, **IDirectPlay2::SetGroupData**, **IDirectPlay2::SetPlayerName**, or **IDirectPlay2::SetGroupName** method. If this flag is not specified, messages will be generated indicating that the data changed.

**DPSESSION_NOMESSAGEID**

Do not attach data to messages indicating who the message is from and who it is to. Saves message overhead if this information is not relevant. (For more information, see the **IDirectPlay2::Receive** method.) If this flag is not specified, the message ID will be added.

**guidInstance**
GUID of the session instance.

**guidApplication**
GUID for the application running in the session instance. It uniquely identifies the application so that DirectPlay connects only to other computers running the same application. This member can be set to GUID_NULL to enumerate sessions for any application.

**dwMaxPlayers**
Maximum number of players allowed in this session.

**dwCurrentPlayers**
Number of players currently in the session.

**lpszSessionName** and **lpszPassword**
Addresses of Unicode strings containing the name and password of the session. Use these members only if the *IDirectPlay2* interface is in use.

**lpszSessionNameA** and **lpszPasswordA**
Addresses of ANSI strings containing the session's name and password. Use these members only if the **IDirectPlay2A** interface is in use.

**dwReserved1** and **dwReserved2**
Reserved for future use.

**dwUser1**, **dwUser2**, **dwUser3**, and **dwUser4**
Application-specific data for the session.

See also **IDirectPlay2::EnumSessions**, **IDirectPlay2::GetSessionDesc**

# System Messages

## DPLMSG_GENERIC

```
typedef struct {
    DWORD dwType;
} DPL_GENERIC, *LPDPLMSG_GENERIC;
```

Generic structure of system messages passed between the lobby client and an application.

**dwType**
Identifies what type of system message has been received.

**DPLSYS_APPTERMINATED**

Indicates the application started by **IDirectPlayLobby::RunApplication** has terminated.

**DPLSYS_CONNECTIONSETTINGSREAD**

Indicates the application started by the **IDirectPlayLobby::RunApplication** method has read the connection settings.

**DPLSYS_DPLAYCONNECTFAILED**

Indicates the application started by **IDirectPlayLobby::RunApplication** failed to connect to a session.

**DPLSYS_DPLAYCONNECTSUCCEEDED**

Indicates the application started by **IDirectPlayLobby::RunApplication** has created a session and is ready for other applications to join or successfully join a session.

# DPMSG_ADDPLAYERTOGROUP

```
typedef struct{
    DWORD dwType;
    DPID  dpIdGroup;
    DPID  dpIdPlayer;
} DPMSG_ADDPLAYERTOGROUP, *LPDPMSG_ADDPLAYERTOGROUP;
```

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

**dwType**
Identifies the message. This member can either be DPSYS_ADDPLAYERTOGROUP or DPSYS_DELETEPLAYERFROMGROUP.

**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.

See also **IDirectPlay2::AddPlayerToGroup**, **IDirectPlay2::DeletePlayerFromGroup**

# DPMSG_CREATEPLAYERORGROUP

```
typedef struct{
    DWORD  dwType;
    DWORD  dwPlayerType;
    DPID   dpId;
    DWORD  dwCurrentPlayers;
    LPVOID lpData;
    DWORD  dwDataSize;
    DPNAME dpnName;
} DPMSG_CREATEPLAYERORGROUP, *LPDPMSG_CREATEPLAYERORGROUP;
```

Contains information for the DPSYS_CREATEPLAYERORGROUP system message. The system sends this message when players and groups are created in a session.

**dwType**
Identifies the message. This member must be set to DPSYS_CREATEPLAYERORGROUP.

**dwPlayerType**
Indicates whether the message applies to a player (DPPLAYERTYPE_PLAYER) or a group (DPPLAYERTYPE_GROUP).

**dpId**
Indicates whether the ID of a player or group has been created.

**dwCurrentPlayers**
Current number of players and groups in the session including the one that has just been added.

**lpData**
Address of any application-specific remote data associated with this player or group. If this member is NULL, there is no remote data.

**dwDataSize**
Size of the data contained in the buffer referenced by **lpData**.

**dpnName**
Structure containing the name of the player or group.

See also **IDirectPlay2::CreateGroup**, **IDirectPlay2::CreatePlayer**

# DPMSG_DELETEPLAYERFROMGROUP

```
typedef DPMSG_ADDPLAYERTOGROUP     DMSG_DELETEPLAYERFROMGROUP;
typedef DPMSG_DELETEPLAYERFROMGROUP     *LPDPMSG_DELETEPLAYERFROMGROUP;
```

Contains information for the DPSYS_DELETEPLAYERFROMGROUP system message. For a description of the structure members, see the **DPMSG_ADDPLAYERTOGROUP** structure.

# DPMSG_DESTROYPLAYERORGROUP

```
typedef struct{
    DWORD  dwType;
    DWORD  dwPlayerType;
    DPID   dpId;
    LPVOID lpLocalData;
    DWORD  dwLocalDataSize;
    LPVOID lpRemoteData;
    DWORD  dwRemoteDataSize;
} DPMSG_DESTROYPLAYERORGROUP, *LPDPMSG_DESTROYPLAYERORGROUP;
```

Contains information for the DPSYS_DESTROYPLAYERORGROUP system message. The system sends this message when players and groups are deleted from a session.

**dwType**
Identifies the message. This member is DPSYS_DESTROYPLAYERORGROUP.

**dwPlayerType**
Identifies whether the message applies to a player (DPPLAYERTYPE_PLAYER) or group (DPPLAYERTYPE_GROUP).

**dpId**
ID of a player or group that has been destroyed.

**lpLocalData**
Address of the local data associated with this player/group.

**dwLocalDataSize**
Size, in bytes, of the local data.

**lpRemoteData**
Address of the remote data associated with this player/group.

**dwRemoteDataSize**
Size, in bytes, of the remote data.

See also **IDirectPlay2::DestroyGroup**, **IDirectPlay2::DestroyPlayer**

# DPMSG_GENERIC

```
typedef struct{
    DWORD dwType;
} DPMSG_GENERIC, *LPDPMSG_GENERIC;
```

This structure is provided for message processing.

**dwType**
Identifies the system message type.

When a system message is received (that is, the value pointed to by the *lpidFrom* parameter equals DPID_SYSMSG), first cast the unknown message data to the

**DPMSG_GENERIC** type, and then perform further processing based on the value of **dwType**. After the message type has been determined, the message can cast to one of the known types of system messages for further processing.

## DPMSG_HOST

```
typedef DPMSG_GENERIC    DPMSG_HOST;
typedef DPMSG_HOST    *LPDPMSG_HOST;
```

When the current session host exits the session, this message is sent to all the players on the computer that inherits the host duties.

## DPMSG_SESSIONLOST

```
typdef DPMSG_GENERIC    DPMSG_SESSIONLOST;
typedef DPMSG_SESSIONLOST    *LPDPMSG_SESSIONLOST;
```

This message is generated by DirectPlay when the connection to all the other players in the session is lost. After the session is lost, messages cannot be sent to remote players, but all data at the time the session was lost is still available. Your applications should try to recover gracefully and exit if this message is received.

## DPMSG_SETPLAYERORGROUPDATA

```
typedef struct {
    DWORD  dwType;
    DWORD  dwPlayerType;
    DPID   dpId;
    LPVOID lpData;
    DWORD  dwDataSize;
} DPMSG_SETPLAYERORGROUPDATA, *LPDPMSG_SETPLAYERORGROUPDATA;
```

Contains information for the DPSYS_SETPLAYERORGROUPDATA system message.

**dwType**
Identifies the message. This member is always DPSYS_SETPLAYERORGROUPDATA.

**dwPlayerType**
Identifies whether the message applies to a player (DPPLAYERTYPE_PLAYER) or a group (DPPLAYERTYPE_GROUP).

**dpId**
ID of the player or group whose data changed.

**lpData**
Address of an application-specific block of data.

**dwDataSize**
Size of the data contained in the buffer referenced by **lpData**.

The system sends this message when an application changes remote player or group data. It is not necessary for the application to save the data from this message; it can be retrieved at any time by using the **IDirectPlay2::GetGroupData** or **IDirectPlay2::GetPlayerData** method with the DPGET_REMOTE flag. This message will not be generated if the **DPSESSION_NODATAMESSAGES** flag is specified in the session description.

See also **IDirectPlay2::GetGroupData**, **IDirectPlay2::GetPlayerData**, **IDirectPlay2::SetGroupData**, **IDirectPlay2::SetPlayerData**

# DPMSG_SETPLAYERORGROUPNAME

```
typedef struct {
    DWORD  dwType;
    DWORD  dwPlayerType;
    DPID   dpId;
    DPNAME dpnName;
} DPMSG_SETPLAYERORGROUPNAME, *LPDPMSG_SETPLAYERORGROUPNAME;
```

Contains information for the DPSYS_SETPLAYERORGROUPNAME system message.

**dwType**
Identifies the message. This member is always DPSYS_SETPLAYERORGROUPNAME.

**dwPlayerType**
Identifies whether the message applies to a player (DPPLAYERTYPE_PLAYER) or a group (DPPLAYERTYPE_GROUP).

**dpId**
ID of the player or group whose name changed.

**dpnName**
Structure containing the new name information for the player or group.

The system sends this message when the name of a player or group has changed. It is not necessary for the application to save the data from this message; it can be retrieved at any time by using the **IDirectPlay2::GetGroupName** or **IDirectPlay2::GetPlayerName** method. This message will not be generated if the DPSESSION_NODATAMESSAGES flag is specified in the session description.

See also **IDirectPlay2::GetGroupName**, **IDirectPlay2::GetPlayerName**, **IDirectPlay2::SetGroupName**, **IDirectPlay2::SetPlayerName**

# Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all *IDirectPlay2* and *IDirectPlayLobby* methods. For a list of the error values each method can return, see the individual method descriptions.

**CLASS_E_NOAGGREGATION**

A non-NULL value was passed for the *pUnkOuter* parameter in **DirectPlayCreate**, **DirectPlayLobbyCreate**, or **IDirectPlayLobby::Connect**.

**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_APPNOTSTARTED**

The application has not been started yet.

**DPERR_BUFFERTOOLARGE**

The data buffer is too large to store.

**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_CANTCREATEGROUP**

A new group cannot be created.

**DPERR_CANTCREATEPLAYER**

A new player cannot be created.

**DPERR_CANTCREATEPROCESS**

Cannot start the application.

**DPERR_CANTCREATESESSION**

A new session 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_INVALIDINTERFACE**

The interface parameter is 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 that DirectPlay is attempting to use is not capable of this function.

**DPERR_NOCONNECTION**

No communication link was established.

**DPERR_NOINTERFACE**

The interface is not supported.

**DPERR_NOMESSAGES**

There are no messages to be received.

**DPERR_NONAMESERVERFOUND**

No name server (host) could be found or created. A host must exist 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_NOTLOBBIED**

Returned by the **IDirectPlayLobby::Connect** method if the application was not started by using the **IDirectPlayLobby::RunApplication** method.

**DPERR_OUTOFMEMORY**

There is insufficient memory to perform the requested operation.

**DPERR_PLAYERLOST**

A player has lost the connection to the session.

**DPERR_SENDTOOBIG**

The message buffer passed to the **IDirectPlay2::Send** method is larger than allowed.

**DPERR_SESSIONLOST**

The connection to the session has been lost.

**DPERR_TIMEOUT**

The operation could not be completed in the specified time.

**DPERR_UNAVAILABLE**

The requested function is not available at this time.

**DPERR_UNKNOWNAPPLICATION**

An unknown application was specified.

**DPERR_UNSUPPORTED**

The function is not available in this implementation.

**DPERR_USERCANCEL**

The user canceled the connection process during a call to the **IDirectPlay2::Open** method.