CHAPTER 4

DirectPlay

About DirectPlay

The Microsoft® DirectPlay® application programming interface (API) for Microsoft Windows® 95 is a software interface that simplifies application access to communication services. DirectPlay has become a technology family that not only provides a way for applications to communicate with each other, independent of the underlying transport, protocol, or online service, but also provides this independence for matchmaking servers and game 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 the developer to deal with the differences that each connectivity solution represents, DirectPlay provides well-defined, generalized communication capabilities. DirectPlay shields developers from the underlying complexities of diverse connectivity implementations, freeing them to concentrate on producing a great application.

What's New in DirectPlay 5?

This section discusses new features in DirectPlay 5. For the most recent updates including new features, additional samples, and further technical information, consult the Microsoft DirectX web site at http://www.microsoft.com/DirectX.

DirectPlay 5 has a new interface, **IDirectPlay3**. This interface inherits directly from **IDirectPlay2** and by default behaves as **IDirectPlay2**. All new functionality is enabled through new methods or new flags.

DirectPlay 5 supports the following new features and methods:

- DirectPlay interface objects can be created directly by using the **CoCreateInstance** method. This eliminates the need to link directly to the dplayx.dll.

- **IDirectPlay3::EnumConnections** enumerates the service providers and lobby providers available to the application. This method supersedes **DirectPlayEnumerate**.
- **IDirectPlay3::InitializeConnection** initializes a DirectPlay connection. This method supersedes **DirectPlayCreate**.
- The new **IDirectPlayLobby2::CreateCompoundAddress** method creates an address to pass to the **InitializeConnection** method.
- **IDirectPlay3::SecureOpen** creates or joins a session on a server that requires security.
- **IDirectPlay3::CreateGroupInGroup**, **IDirectPlay3::AddGroupToGroup**, **IDirectPlay3::DeleteGroupFromGroup**, and **IDirectPlay3::EnumGroupsInGroup** add richer group functionality and navigation. The new **IDirectPlay3::GetGroupFlags** and **IDirectPlay3::GetGroupParent** methods give ready access to additional group information.
- **IDirectPlay3::SendChatMessage** enables players to chat with other players in a session or connected to a lobby server, using standardized messages.
- **IDirectPlay3::SetGroupConnectionSettings**, **IDirectPlay3::GetGroupConnectionSettings**, and **IDirectPlay3::StartSession** enable synchronized application launching from a lobby server.
- The new **DPCREDENTIALS** structure holds the user name, password, and domain to use when connecting to a secure server. The **DPSECURITYDESC** structure describes the security properties of a DirectPlay session instance.
- The new **DPCOMPOUNDADDRESSELEMENT** structure describes a *DirectPlay Address* data chunk that can be used to create longer DirectPlay Addresses.
- The new **IDirectPlay3::GetPlayerAccount** method and new **DPACCOUNTDESC** can be used by a session host to obtain account information for a player logged into a secure session.
- The new **IDirectPlay3::GetPlayerFlags** method gives access to a player's flag settings.

DirectPlay 5 also supports new functionality for existing DirectPlay 3 methods:

- An application can create multiple DirectPlay objects.
- The **IDirectPlay3::SetSessionDesc** method enables the host to change the session description.
- The **IDirectPlay3::EnumSessions** method can now be called asynchronously and will maintain a constantly refreshing list of sessions available on the session
- Password protection of sessions has been improved. Specify the **DPENUMSESSIONS_PASSWORDREQUIRED** flag in **EnumSessions** to enumerate password-protected sessions (in addition to nonpassword-protected sessions). The **DPSESSIONDESC2** structure will contain a flag indicating that

the session needs a password. Put the password in the **DPSESSIONDESC2** structure passed to the **Open** method to join the session.

- Applications can override the service provider dialog boxes that prompt users for information. To prevent these dialog boxes from appearing, create a *DirectPlay Address* using the **IDirectPlayLobby2::CreateAddress** or **IDirectPlayLobby2::CreateCompoundAddress** methods and then call **IDirectPlay3::InitializeConnection** with this DirectPlay Address. A subsequent call to **IDirectPlay3::EnumSessions** will not display a dialog box prompting the user for address information.
- A new multicast server option improves group messaging.
- Support has been added for scalable client/server architecture applications.
- Sessions can be hosted securely and require users to log in with a name and password.
- Members of a secure session can send digitally signed or encrypted messages, by using the **DPSEND_SIGNED** and **DPSEND_ENCRYPTED** flags in the **Send** method.

Updates to DirectPlay

Be sure to consult the Microsoft DirectX web site for the latest information about new features and updates to DirectPlay, additional samples, and further technical information about using DirectPlay.

The web site is http://www.microsoft.com/DirectX.

Writing a Network Application

The fundamental concern when writing a networked, multiplayer application is how to propagate state information to all the computers in the session. The state of the session defines what users see on their computers and what actions they can perform. Generally, two things make up the session state: the environment, and the individual users or players.

The environment consists of the objects that the players can manipulate or interact with. This can include a map of a dungeon, a racecourse, or even a document. The environment can also include computer-controlled players.

Within the environment, each player also has a state indicating the player's current properties. These can include position, velocity, energy, armor, and so on.

When a player first joins the session, the session's current state must be downloaded, including the environment and the state of the other players.

Actions performed by users or the natural progression of a game change the state of the session. When this happens, the change must be propagated to the other computers in the session through messages. There are many techniques for updating the session state, but they all depend on sending messages between computers.

When two players perform an action on the same object (like opening a door) or on each other (like swinging a sword) a conflict can arise. That is, two players are trying to do something where only one of them can succeed. In these cases, arbitration or conflict resolution is needed. There are several ways to resolve conflicts like this: the two players can communicate with each other and decide which one wins, or a special player can arbitrate all conflicts.

The actions of players change the state of a game, and conflicting actions must be resolved before the game's state can be updated. There are two fundamental ways of maintaining state in a session. The first way is called peer-to-peer. In peer-to-peer sessions, all the computers in the session have the complete state of the environment and all the players. Any change in state that happens on one computer must be propagated to all the other computers in the session. No one computer is really in charge.

The second way to maintain the session state is called client/server. In client/server sessions, one computer is designated the server, and it maintains the complete state of the game and performs conflict resolution. All the other computers in the session are clients, and they download some portion of the state from the server. When they need to change the state, they tell the server, and the server propagates the change to other clients as needed.

With DirectPlay, you can create and manage both peer-to-peer and client/server applications. DirectPlay provides all the tools needed to write a networked, multiplayer application by providing the services to manage players, send messages between players, and automatically propagate the state among all the computers.

DirectPlay Overview

This section contains the following topics, which provide general information about the DirectPlay component.

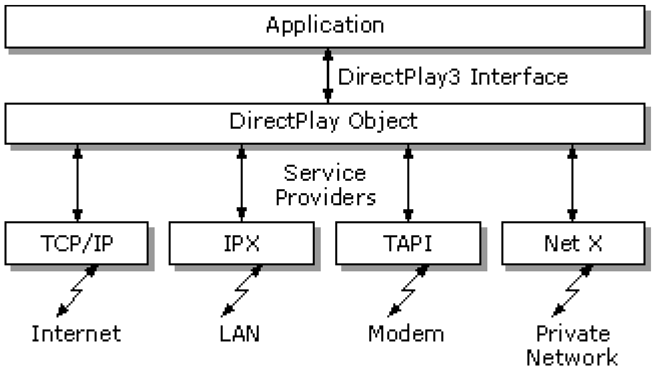
- *Architecture*
- *Session Management*
- *Player Management*
- *Message Management*
- *Group Management*
- *Overview of DirectPlay Communications*

Architecture

The DirectPlay API is a network abstraction and distributed object system that applications can be written to. The API defines the functionality of the abstract DirectPlay network and all the functionality is available to your application regardless of whether the actual underlying network supports it or not. In cases where the underlying network does not support a method, DirectPlay contains all the code necessary to emulate it. Examples include group messaging and guaranteed messaging.

DirectPlay's service provider architecture insulates the application from the underlying network it is running on. The application can query DirectPlay for specific capabilities of the underlying network, such as latency and bandwidth, and adjust its communications accordingly.

The following diagram illustrates the DirectPlay service provider architecture.



The first step in using DirectPlay is to select which service provider to use. The service provider determines what type of network or protocol will be used for communications. The protocol can range from TCP/IP over the Internet to an IPX local area network to a serial cable connection between two computers.

Use the service provider to make a connection to a point on the network. The user may need to provide additional information to make a connection, or the application can specify the connection parameters.

Connection Management Methods

DirectPlay provides two very useful connection management methods:

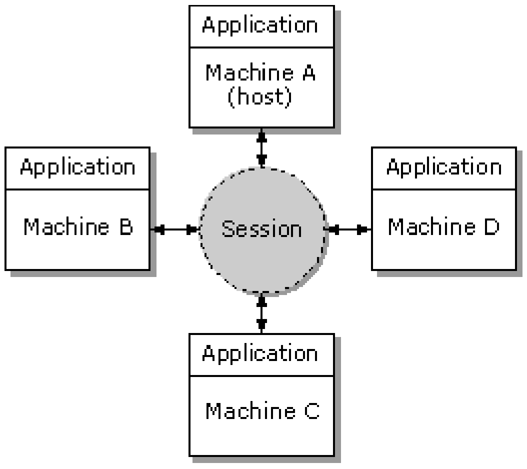
- **IDirectPlay3::EnumConnections** enumerates all the connections that are available to the application.
- **IDirectPlay3::InitializeConnection** initializes a specific connection.

Session Management

A DirectPlay session is a communications channel between several computers. Before an application can start communicating with other computers it must be part of a session. An application can do this in one of two ways: it can enumerate all the existing sessions on a network and join one of them, or it can create a new session and wait for other computers to join it. Once the application is part of a session, it can create a player and exchange messages with all the other players in the session.

Each session has one computer that is designated as the host. The host is the owner of the session and is the only computer that can change the session's properties.

The following diagram illustrates the DirectPlay session model.



Session Management Methods

DirectPlay provides several session management methods:

- **IDirectPlay3::EnumSessions** enumerates all the available sessions.
- **IDirectPlay3::Open** joins one of the enumerated sessions or creates a new session.
- **IDirectPlay3::SecureOpen** performs the same function as **IDirectPlay3::Open**, but enables the application to alter the default opening behavior.
- **IDirectPlay3::Close** leaves the currently open session.
- **IDirectPlay3::GetSessionDesc** gets the properties of the current session.
- **IDirectPlay3::SetSessionDesc** changes the properties of the current session.
- **IDirectPlay3::GetCaps** gets the communications capabilities of the underlying network.

Player Management

The most basic entity within a DirectPlay session is a player. A player represents a logical object within the session that can send and receive messages. DirectPlay does not have any representation of a physical computer in the session. Each player is identified as being either a local player (one that exists on your computer) or a remote player (one that exists on another computer). Each computer must have at least one local player before it can start sending and receiving messages. Individual computers can have more than one local player.

When an application sends a message, it is always directed to another player — not another computer. The destination player can be another local player (in which case the message will not go out over the network) or a remote player. Similarly, when an application receives messages they are always addressed to a specific (local) player and marked as being from some other player (except system messages, which are always marked as being from DPID_SYSMSG).

DirectPlay provides some additional player management methods that an application can use. These methods can save the application from having to implement a list of players and data associated with each one. You do not need these methods to use DirectPlay successfully. They enable an application to associate a name with a player and automatically propagate that name to all the computers in the session. Similarly, the application can associate some arbitrary data with a player that will be propagated to all the other computers in the session. The application can also associate private local data with a player that is available only to the local computer.

Basic Player Management Methods

DirectPlay provides several basic player management methods:

- **IDirectPlay3::EnumPlayers** enumerates all the players in the sessions.
- **IDirectPlay3::CreatePlayer** creates a local player.
- **IDirectPlay3::DestroyPlayer** destroys a local player.

Additional Player Management Methods

DirectPlay provides these additional methods to manage player information:

- **IDirectPlay3::GetPlayerCaps** gets a player's communications capabilities.
- **IDirectPlay3::GetPlayerName** gets a player's name.
- **IDirectPlay3::SetPlayerName** changes a player's name.
- **IDirectPlay3::GetPlayerData** gets the application-specific data associated with a player.

- **IDirectPlay3::SetPlayerData** changes the application-specific data associated with a player.
- **IDirectPlay3::GetPlayerAddress** gets a player's network-specific address.
- **IDirectPlay3::GetPlayerAccount** gets a player's account information.
- **IDirectPlay3::GetPlayerFlags** gets a player's flag settings.

Message Management

Once an application has created a player within a session, it can start exchanging messages with other players in the session. DirectPlay imposes no message format or extra bytes on the message. A message can be sent to an individual player, to all the players in the sessions, or to a subset of players that have been defined as a group (see *Group Management*). When sending a message it must also be marked as being from a specific local player.

All the messages received by an application are put into a receive queue. The application must retrieve individual messages from the queue and act on them as appropriate. The application can either poll the receive queue for messages or use a separate thread that waits on a synchronization event for notification that a new message has arrived.

There are two types of messages. Player messages are messages that another player in the session sent. This type of message is directed to a specific player and is marked as being from the sending player. System messages are sent to all the players in a session and are all marked as being from the system (DPID_SYSMSG). DirectPlay generates system messages to notify the application of some change in state of the session; for example, when a new player has been created. See *Using System Messages* for more information.

Basic Message Management Methods

DirectPlay provides several message management methods:

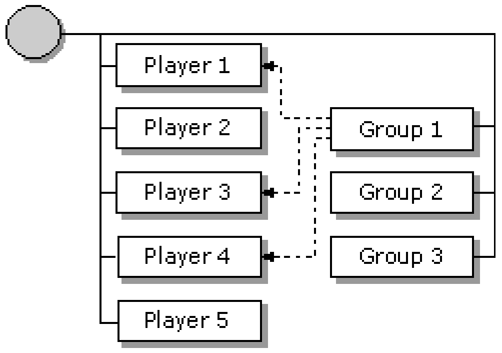
- **IDirectPlay3::Send** sends a message from a local player to another player in the session.
- **IDirectPlay3::SendChatMessage** enables players to chat with other players using standardized messages.
- **IDirectPlay3::Receive** retrieves a message from the incoming message queue.
- **IDirectPlay3::GetMessageCount** gets the number of messages currently in the incoming message queue.

Group Management

DirectPlay supports groups within a session. A group is logical collection of players. By creating a group of players, an application can send a single message to the group and all the players in the group will receive the message. A group is the means by which a network's multicast capabilities are exposed to the application.

Groups can also be used as a general means to organize players in a session. A player can belong to more than one group. DirectPlay provides methods for administering groups and their membership. Additional methods associate names and data with individual groups as a convenience, but you don't need them to use groups.

The following diagram shows a logical representation of the contents of a DirectPlay session.



Basic Group Management Methods

DirectPlay provides several group management methods:

- **IDirectPlay3::EnumGroups** enumerates all the groups in the session.
- **IDirectPlay3::CreateGroup** creates a new group.
- **IDirectPlay3::DestroyGroup** destroys a group that this computer created.
- **IDirectPlay3::EnumGroupPlayers** enumerates the players that are in a group.
- **IDirectPlay3::AddPlayerToGroup** adds a player to a group.
- **IDirectPlay3::DeletePlayerFromGroup** removes a player from a group.

Additional Group Management Methods

DirectPlay provides these additional methods to manage group information:

- **IDirectPlay3::GetGroupName** gets the group's name.
- **IDirectPlay3::SetGroupName** changes the name of a group created by this computer.
- **IDirectPlay3::GetGroupData** gets the application-specific data associated with a group.

- **IDirectPlay3::SetGroupData** changes the application-specific data associated with a group created by this computer.
- **IDirectPlay3::GetGroupFlags** gets the flags describing a group.
- **IDirectPlay3::GetGroupParent** gets the DPID of the parent of the group.

New Group Management Methods

DirectPlay 5 has added these methods that manage groups within groups and shortcuts to groups:

- **IDirectPlay3::CreateGroupInGroup** creates a group within an existing group.
- **IDirectPlay3::EnumGroupsInGroup** enumerates all the groups within another group.
- **IDirectPlay3::AddGroupToGroup** adds a shortcut from a group to an already existing group.
- **IDirectPlay3::DeleteGroupFromGroup** removes a group previously added to another with **AddGroupToGroup**, but doesn't destroy the deleted group.
- **IDirectPlay3::GetGroupConnectionSettings** retrieves a group's connection settings.
- **IDirectPlay3::SetGroupConnectionSettings** sets a group's connection settings.

Overview of DirectPlay Communications

DirectPlay's default mode of communications is peer-to-peer. In this model, the session's complete state is replicated on all the computers in the session. This means that the session description data, the list of players and groups, and the names and remote data associated with each session are duplicated on every computer. When one computer changes something, it is immediately propagated to all the other computers.

DirectPlay's alternative mode of communications is client/server. In this model, only the server stores the session's complete state and each client has only a subset of the session's state. Each client has only the information that is relevant to that computer and receives that information from the server. When one computer changes something, it propagates the change to the server. The server then determines which clients it must inform of the change.

An application can manage its own data using either a client/server model or peer-to-peer model, but this will not change how the underlying DirectPlay session state is managed.

The following sections discuss the two modes of communications within DirectPlay sessions:

- *Peer-to-Peer Session*

- *Client/Server Session*

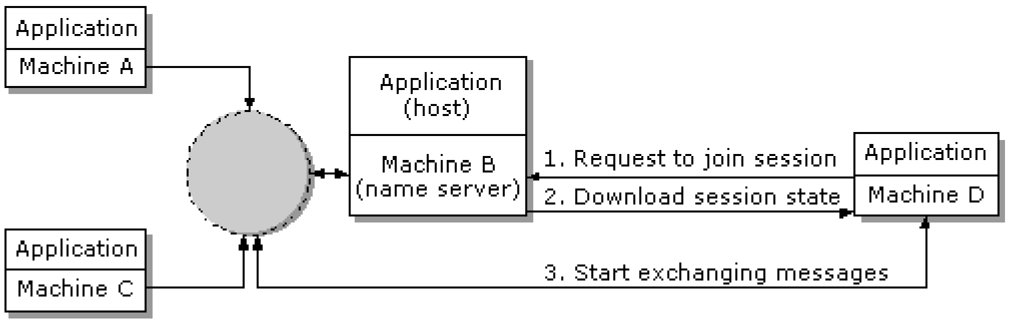
Peer-to-Peer Session

In a peer-to-peer DirectPlay session, one computer is designated the name server. This computer responds to enumeration requests, regulates computers trying to join the session, downloads the session's state to new computers that have joined, and generates ID numbers as players and groups are created. Beyond that, the name server is just another peer in the session and runs the same application as all the other peers. Messages are not routed through the name server and the name server does not generate all the system messages.

DirectPlay automatically performs the name server functions, which are not exposed to the application in any way. The DirectPlay application running on the name server is also the session's host. Only the host application can change the session description data. Each peer application in the session has access to the complete list of players and groups in the session (see *Session Management*) and can send and receive messages from any other player in the session or send a message to any group in the session. See *Session Management* for more details.

The session's network address corresponds to the name server's network address. When a computer needs to join the session, it sends the join request to the name server. In response, the name server downloads the session's state to new computers. If the name server leaves the session, a new computer is elected name server. When the name server migrates, the session's network address also changes.

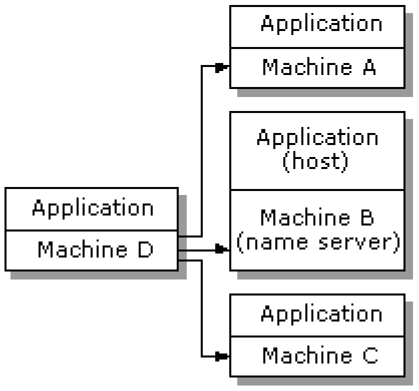
The following diagram illustrates joining a session. To join a session, a machine sends the join request to the name server of the session.



Peer-to-peer sessions generally have a limitation on the number of computers that can participate. Every change in state on every computer (like player movements) must be broadcast to all the other computers in the session. Because there is a limit on how much data a computer can receive (especially if connected through a phone line) there is a limit to how many computers can generate data. Minimizing the quantity and frequency of data exchanged can help increase the number of computers that can be in a session before performance degrades.

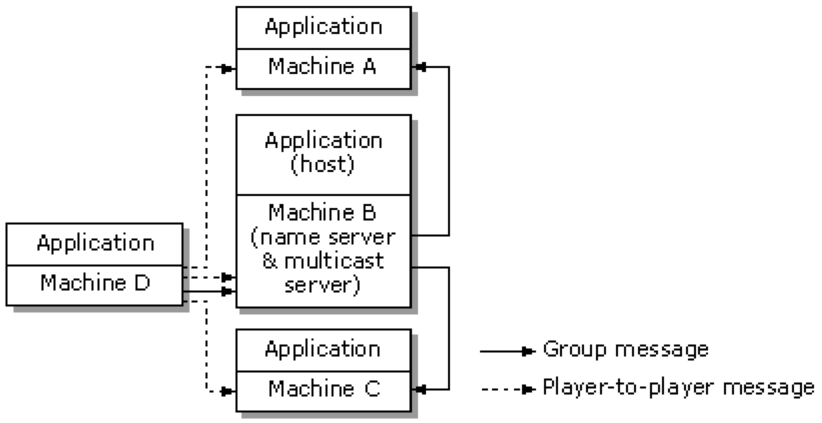
Computers in a peer-to-peer DirectPlay session communicate directly with each other. That is, messages are not sent through an intermediate computer to reach their destination (although they can go through a router). When DirectPlay needs to send the same message to a set of computers, it attempts to use any multicast capabilities in the service provider. If multicast is not supported, then individual unicast messages are sent to each destination computer.

The following diagram illustrates group messaging without multicast. To broadcast a message, individual messages are sent to each machine.



This can be a source of inefficiency in sessions with a large number of players (more than four). Adding a multicast server to the session can alleviate this. A multicast server is a computer that will forward a single message to multiple destinations. A computer needing to broadcast a message to all the other computers in the session can send one message to the multicast server, which in turn sends individual messages to all the other computers. This is more efficient, because the multicast server is connected to the network through a high-speed link and can therefore pump out unicast messages faster than any individual computer. The assumption is that the server has a high-speed link to the network (such as a T1 line) while the other computers have slow-speed links (such as phone lines).

The following diagram illustrates group messaging using a multicast server. Player-to-player messages are sent directly to the destination machine. Group messages are sent to the multicast server, which forwards them to all the destination machines.



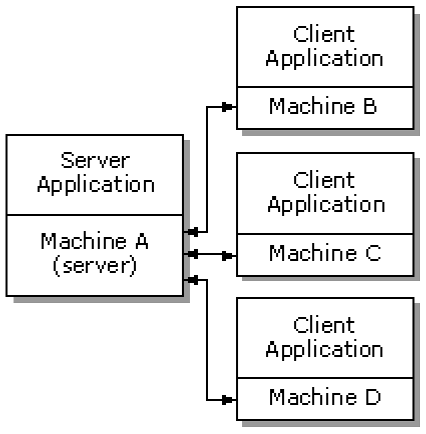
DirectPlay supports the creation of a multicast server in a session. If the name server is on a sufficiently high-speed link, it can choose to be a multicast server for the session as well. Adding a multicast server to a session only changes how group messages (including broadcast) are routed — the application's behavior doesn't change. Unicast player-to-player messages are still sent directly to the destination computers.

Client/Server Session

In a DirectPlay client/server session, one computer is designated the server. Like the peer-to-peer name server, this computer responds to enumeration requests, regulates computers trying to join the session, downloads the session's state to new computers that have joined, and generates ID numbers as players and groups are created. Unlike the peer-to-peer name server, all messages in the session are routed through the server.

The server computer runs a special version of the application (the application server) that can maintain the master state of the game, updating that state according to actions that the client computers take, and notifying individual clients of relevant events. The application server creates a special player called the server player. Any client computer can send a message directly to the server player or receive a message from the server player. When the server player receives a message it can send messages to other client players to inform them of a change in state.

The following diagram illustrates client\server communications. Each machine communicates directly with the server only. The server can forward messages to other clients.

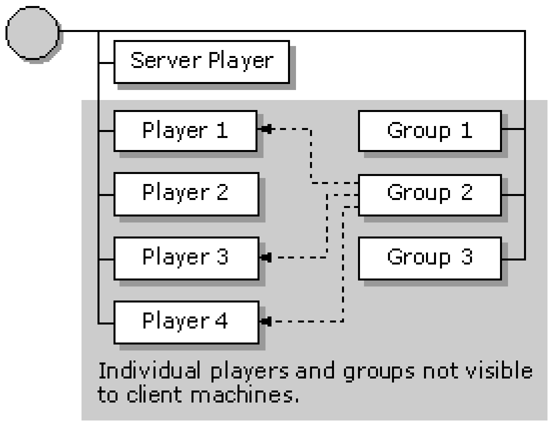


The client computers run client versions of the application that can maintain the local state, updating the state in response to messages from the server, and sending messages to the server when some local action has occurred.

You can implement a client/server application in two ways. One mode is a hybrid peer-to-peer and client/server session, where every client application has the complete player and group list available to it (including the server player). Any client player can send a message directly to any player in the session, and each player or group can have data associated with it (such as name or remote data) that will be updated on all the computers whenever it changes. All the benefits of peer-to-peer state propagation are available, and, in addition, a server player is available. However, you can't control the propagation of the player-to-player data. Depending on the frequency with which players and groups are created and destroyed or their data changes, client computers could be overloaded with in-coming messages. This type of a session is not scalable beyond about 16 players. It isn't very different from a peer-to-peer session, except that it has a server player, all messages are routed through the server, and the server player will receive a copy of all the messages that pass through the server.

The second mode is pure client/server. Here, each client application sees only the server player and its own local players in the session. A client player can send messages only to the server player and receive messages from only the server player. The application server can see all the players on all clients. When the server receives a message from a client, it can intelligently decide how to update the master application state and which clients it must inform of this update. In this mode, the client application must manage the player list and the data associated with each player.

The following diagram shows a logical representation of the contents of a DirectPlay client/server session.



Like a peer-to-peer session, the session's network address corresponds to the server's network address. When a computer needs to join the session, it sends the join request to the server. In response, the server downloads the session's state to the new computers. If the server leaves the session, the session is terminated.

Because all messages are routed through the server, it automatically behaves as a multicast server.

Security

Using DirectPlay security features, an application running on a server can create secure sessions. DirectPlay implements security through the Security Support Provider Interface (SSPI) on Windows. Key features supported by DirectPlay in a secure session are:

- Authentication to verify the identity of the user. Once a user has been authenticated, communications between the client and server can be done securely either by digitally signing or encrypting the messages.
- Digitally signed system messages to verify the identity of the sender.
- Encryption of sensitive system messages.

For more information, including how to set up a secure server, start a secure session, and specify security packages, see *Security and Authentication*.

DirectPlay Lobby Overview

A DirectPlay lobby server is a common place on a network that (at a minimum) tracks DirectPlay application sessions in progress, and users that are connected to the server. Users can navigate around the lobby server to find areas of interest. At any location on the lobby server the user can chat with other users, join sessions that are in progress, or gather a group of players to start a new session.

A lobby server's main advantage is that it acts as a central, well-known location where users can go to find sessions and other people to interact with. The lobby server manages all the network addresses of the various players and sessions and can automatically manage launching applications and connecting them to the correct address without user intervention. The management of sessions, players, and their network addresses is especially useful on the Internet, where users generally can't find opponents or get applications connected in a session easily. It also has the advantage of launching application sessions for the user without the user having to enter any network configuration information.

Lobby servers can be greatly enhanced to provide more services to the user, such as tournaments, individual score tracking and high scores, personal profiles, avatars, message boards, news, a broader user interface, authenticated membership, software updates, and so on.

A lobby-aware application is one that has been specifically developed to interoperate with lobby servers. There are two types of lobbies that a user can experience. An external lobby is a client application whose sole purpose is to interact with a lobby server and the other users connected to it. When the time comes to start or join an application session, the lobby client launches the application in a separate process and gives it all the information necessary to establish a connection to the session. Adding support for external lobby launching to a DirectPlay application is quite straightforward and explained further in *Supporting External Lobby Launching*.

An internal lobby is a lobby user interface that is integrated into the application itself. This is more difficult to implement because the application must implement the user interface for the lobby as well as that for the game. However, it has the advantage that you can customize the lobby experience to match the application's theme.

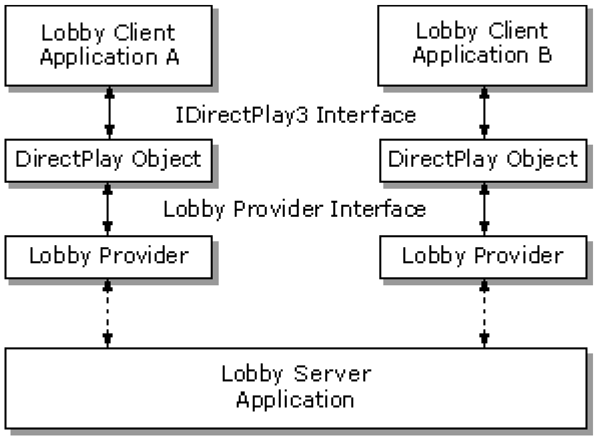
This section discusses the following topics.

- *DirectPlay Lobby Architecture*
- *Lobby Sessions*
- *Lobby Navigation*
- *Synchronized Launching*

DirectPlay Lobby Architecture

The DirectPlay lobby architecture consists of a client API that all applications, whether they are external lobby clients or applications that have a lobby client built in, use to connect to any DirectPlay-compliant lobby server. This is done through a lobby provider interface that, like a service provider, abstracts the interaction with the lobby server. The author of the lobby server application must write the lobby provider that resides on the client computer. The application calls the standard DirectPlay APIs, and the lobby provider's dynamic-link library (DLL) services these methods by communicating with the lobby server software.

The following diagram illustrates the DirectPlay lobby architecture. Different lobby client applications can connect to the same lobby server through the DirectPlay API.



At the very least, the lobby server must be able to track all the users currently connected to it, organize those players by grouping them, and synchronize the launch of an application session.

The DirectPlay API defines a common level of functionality for all lobby servers. Any generic lobby client application can connect to any generic lobby server and, through the DirectPlay API and the lobby provider architecture, they can interoperate successfully. An application can extend the basic functionality with **Send** and **Receive**. A lobby client designed to work with a specific lobby server can implement extended functionality that is not defined by the DirectPlay API.

There is no separate API to communicate with a DirectPlay lobby server. Interaction with a lobby server has been abstracted so that the same **IDirectPlay3** interface used to communicate in an application session can be used to communicate with the lobby server. Additional methods and messages have been added to the interface to support the additional functionality that a lobby server requires.

By using the same DirectPlay methods to interact with the lobby server, it is simple to add a lobby client interface to an application, because the same APIs and concepts are being leveraged.

Lobby Sessions

A lobby session closely resembles a DirectPlay client/server session (see the illustration in *Client/Server Session*). The term *lobby session* refers to a connection to a lobby server where clients and the server have not been specifically written to interoperate. The term *application session* refers to a traditional DirectPlay session in which all the clients and the server (if any) have been specifically written to interoperate.

A DirectPlay lobby session is used to represent a connection to a lobby server. Like a DirectPlay application, the first step in using DirectPlay to communicate with a lobby server is to select which lobby provider to use and which lobby server to connect to. Like a DirectPlay application, the **IDirectPlay3::EnumConnections** and **IDirectPlay3::InitializeConnection** methods are used to do this.

The lobby client uses the same session management methods as an application client to locate and to join a lobby session; for example, *EnumSessions*, *Open*, *SecureOpen*, and *Close*. Joining a lobby session gives the client application access to all the information on the lobby server and enables the user to interact with other users on the lobby server.

Like a DirectPlay application session, the player is the basic entity in a lobby session. Each player represents a user on a client computer connected to the lobby server. There is also the server player representing the lobby server. Once a lobby client connects to a lobby server (by joining the lobby session), it must establish the user's presence by creating a player and adding the player to an initial group before the player can start communicating with the server and other players. In fact, other users connected to the session won't even be aware of the new user's presence until this happens.

The same player management methods used by application sessions are used to manage players in a lobby session; for example, *EnumPlayers*, *CreatePlayer*, *DestroyPlayer*, *GetPlayerName*, *SetPlayerName*, and *GetPlayerCaps*.

Unlike a DirectPlay application session, *Send* and *Receive* generally cannot be used to exchange messages with other players. In the DirectPlay lobby architecture, any application that uses the DirectPlay API can connect to any lobby server. This means that many different lobby client applications might be present in the same lobby session, all of which were written by different developers. Simply sending a message to another client player or the server player does not guarantee that it will be interpreted correctly by the recipient application.

Common functions that require communications between clients and the server have new methods created for them. DirectPlay determines the message's format. The recipient receives a system message containing the content of the message in a well-defined data structure. For example, the **IDirectPlay3::SendChatMessage** method sends a text chat message to another player.

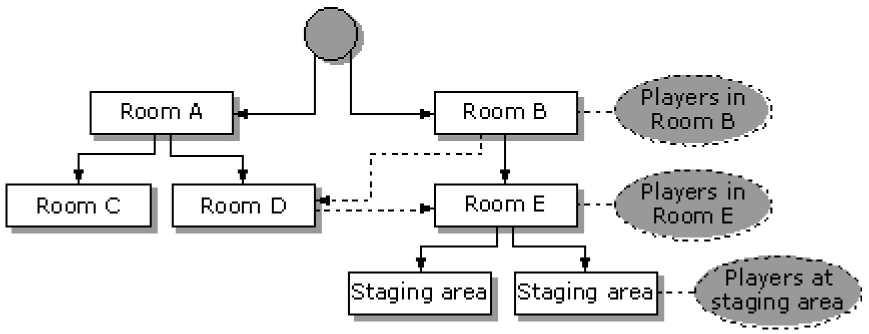
Groups take on more significance in a lobby session, because they are used to define the organization of the entire lobby server and for launching application sessions. You can create a complete hierarchy of groups to manage and organize all the players and sessions that the lobby server tracks.

For the purposes of the lobby server, groups are classified as one of two types. A standard group can contain players and other groups and is called a **room**. A room is primarily used as a meeting place for players to interact with other players in the context of the lobby. The room contains links to other rooms and links to the second type of group — a **staging area**.

A staging area group typically contains only players. A staging area is used to marshal players together in order to launch a new session. Once the session has been launched, the staging area can remain in existence so that new players can join the session in progress.

Room groups are primarily organized in a hierarchical structure. A few top-level groups are created and other groups are created within existing groups. The lobby client can enumerate all the top-level groups as well as the groups contained within a group. The lobby server can also organize the groups in a web structure. Once a group is created, you can add it to another group, which creates a link between the two groups. When a lobby client enumerates groups within a group, all subgroups and linked groups are returned.

The following diagram illustrates the organization of a lobby session.



Players can only be seen in groups. They cannot be seen until they are part of a group. Players can belong to more than one group at once.

Lobby Navigation

Players can navigate through the lobby server space simply by deleting themselves from their current group and adding themselves to a new group. At any time, the **IDirectPlay3::EnumGroupsInGroup** method can be called to determine what groups are linked to the current group.

In general, the scope of the lobby session visible to a player is limited by the groups the player is part of. This is done to limit the amount of information that the server must download to the client.

Synchronized Launching

Application sessions are marshaled and launched from a staging area. One player creates a staging area and waits for other players to join it. A player can set the properties of the application session to be created through the **IDirectPlay3::SetGroupConnectionSettings** method. Other players who are considering joining the group can find out what the application session properties are by calling the **IDirectPlay3::GetGroupConnectionSettings** method. Once enough players have joined the staging area, any player can call the **IDirectPlay3::StartSession** method to initiate the synchronized launch sequence. Every player in the staging area groups receives a **DPMSG_STARTSESSION** system message with a **DPLCONNECTION** structure. If the player is using an external lobby client, it can launch the application using the **IDirectPlayLobby2::RunApplication** method. If the player is using an internal lobby, the application can use the information to establish a connection to the appropriate session by calling **IDirectPlayLobby2::Connect**.

DirectPlay Providers

This section describes the DirectPlay interface and the DirectPlay service providers and how they interact with each other. See:

- *Service Providers*
- *Lobby Providers*

The DirectPlay interface is a common interface to the application for simple send and receive type messaging as well as higher-level services like player management, groups for multicast, data management and propagation, and lobby services for locating other players on a network.

The service providers furnish network-specific communications services as requested by DirectPlay. Online services and network operators can supply service providers to use specialized hardware, protocols, communications media, and network resources. A service provider can simply be a layer between DirectPlay and an existing transport like Winsock, or it can use specialized resources on an online service such as multicast servers, enhanced quality of service, or billing services. Microsoft includes four generic service providers with DirectPlay: *head-to-head modem (TAPI)*, *serial connection*, *Internet TCP/IP* (using Winsock), and *IPX* (also using Winsock).

The DirectPlay interface hides the complexities and unique tasks required to establish an arbitrary communications link inside the DirectPlay provider implementation. An application using DirectPlay need only concern itself with the performance and capabilities of the virtual network presented by DirectPlay. It need not know whether a modem, network card, or online service is providing the medium.

DirectPlay will dynamically bind to any DirectPlay 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, protocol, and other network resources.

The DirectPlay API is exposed to the application through several COM interfaces. (See *DirectPlay Interfaces* for a discussion of COM and DirectPlay.) Application sessions can talk to someone else on the network through the **IDirectPlay3** and **IDirectPlay3A** interfaces. The first uses Unicode strings in all the DirectPlay structures, while the second uses ANSI strings. **IDirectPlay**, **IDirectPlay2**, and **IDirectPlay2A** still exist for backward compatibility with applications written to a previous version of the DirectPlay SDK.

Lobby clients can talk to another application on the same machine through the **IDirectPlayLobby2** and **IDirectPlayLobby2A** interfaces. The first interface uses Unicode strings while the second uses ANSI strings. The **IDirectPlayLobby** interface still exists for backward compatibility.

Service Providers

The service provider furnishes network-specific communication services as requested by DirectPlay. Online services and network operators can supply service providers for specialized hardware and communications media. Microsoft includes the following service providers with DirectPlay:

- *TCP/IP*
- *IPX*
- *Modem-to-Modem*
- *Serial Link*

Individual service providers are identified using a GUID. GUIDs for the standard service providers are listed in the header file DPLAY.H. Third-party service providers will have their own GUIDs.

This section outlines what to expect from the default service providers that Microsoft ships. To obtain information about the behavior of third-party service providers, contact the network operator.

TCP/IP

The TCP/IP service provider uses Winsock to communicate over the Internet or local area network (LAN) using the TCP/IP protocol. It uses UDP (User Datagram Protocol) packets for nonguaranteed messaging and TCP for guaranteed messaging. A single computer can host multiple DirectPlay sessions using TCP/IP.

When the **IDirectPlay3::EnumSessions** method is called, TCP/IP displays a dialog box asking the user for the session's IP address. The user must enter the IP address of the computer hosting the session to be joined. If the computer has a name (such as microsoft.com), the name can be used instead of the IP address, and DirectPlay will use Domain Name System (DNS) lookup to find it. The **IDirectPlay3::EnumSessions** method will return the sessions that the computer is hosting. The user can also leave the address blank and hit OK. In this case, DirectPlay will broadcast a message looking for sessions. This will generally only work on a LAN and only on the same subnet.

Note: A Windows 95 user can determine his or her IP address by choosing Run from the Start menu and typing WINIPCFG as the program to run. A Windows NT user can determine his or her IP address by running IPConfig from the command line. If the user is connected to both a LAN and a dial-up Internet service provider (ISP), the computer can have two IP addresses and the correct one must be selected. Most dial-up ISPs assign a dynamic IP address that changes each time the user logs on.

An application can call **IDirectPlay3::InitializeConnection**, or can call **IDirectPlayLobby2::SetConnectionSettings** followed by a call to **Connect**, to supply an IP address to the service provider in a *DirectPlay Address*. The address must be a null-terminated ANSI or Unicode string (each has a different data type GUID). If a broadcast enumeration of sessions is desired, the address must be a zero-length string; that is, a string consisting of just the null terminator.

The DirectPlay TCP/IP service provider does not generally work through firewalls.

Adding DirectPlay lobby support can eliminate the need for users to enter an IP address if they start the game from a lobby server.

This service provider can be identified using the symbol definition DPSPGUID_TCPIP.

Note: A Windows 95 user can configure his or her computer connections to display or not display a dialog box requesting connection information when DirectPlay tries to initiate a TCP/IP connection. To suppress the display of this dialog box, follow these steps:

1 Open Control Panel.

2 Double-click the Internet icon.
3 Choose the Connection tab.
4 Clear the checkbox next to **Connect to the Internet as needed**.

IPX

The IPX service provider uses Winsock to communicate over a local area network (LAN) using the Internet Packet Exchange (IPX) protocol. The service provider only supports nonguaranteed messaging. A single computer can host only one DirectPlay session using IPX.

IPX always uses a broadcast to find sessions on the network, so the **IDirectPlay3::EnumSessions** method will not display a dialog box requesting IP addresses.

IPX will not enumerate sessions on another subnet.

Once a session is established, packets are sent directly between computers (they are not broadcast).

This service provider can be identified using the symbol definition DPSPGUID_IPX.

Modem-to-Modem

Modem-to-modem communication uses TAPI (Telephony Application Programming Interface) to communicate with another modem.

Creating a session (by using the **IDirectPlay3::Open** method) causes a dialog box to appear, asking the user which modem to wait for a call on. The **IDirectPlay3::EnumSessions** method will also display a dialog box asking the user what phone number to call and which modem to use. Once the information is entered, DirectPlay will dial the modem and try to find sessions hosted by the computer on the other end. In both cases, dialogs are displayed to show the progress.

The current list of available modems can be obtained from the service provider by initializing it and calling **IDirectPlay3::GetPlayerAddress** with a player ID of zero. The *DirectPlay Address* returned will contain a data chunk with the ANSI modem names (DPAID_Modem) and the Unicode modem names (DPAID_ModemW). The list of modems is a series of NULL-terminated strings with a zero-length string at the end of the list.

If you insert too many delays into the message processing, you may lose packets; for example, if you print a lot of debug information.

This service provider can be identified using the symbol definition DPSPGUID_MODEM.

Serial Link

A serial link is used to communicate with another computer through the serial ports.

Creating a session (using the **IDirectPlay3::Open** method) causes a dialog box to appear asking the user to configure the serial port. The **IDirectPlay3::EnumSessions** method will also display a dialog box asking the user to configure the serial port. You must configure the serial port the same way on both computers.

If you insert too many delays into the message processing, you may lose packets; for example, if you print a lot of debug information.

This service provider can be identified using the symbol definition DPSPGUID_SERIAL.

Lobby Providers

The lobby provider is a client component (DLL) supplied by the developer of a lobby server. It implements communications functions with the lobby server as requested by DirectPlay. A lobby client written using the DirectPlay API can interoperate with any lobby server for which a lobby provider DLL is present on the system.

The DirectX SDK installs a lobby provider for use with the test lobby server (LSERVER.EXE) included with the SDK. This lobby provider is used by the BELLHOP sample lobby client and together they can be used to test the lobby-aware functions your applications.

Using DirectPlay

This section contains the following topics that explain how to use different aspects of DirectPlay.

- *Debug versus Retail DLLs*
- *Working with GUIDs*
- *DirectPlay Interfaces*
- *Using Callback Functions*
- *Building Lobby-Aware Applications*
- *DirectPlay Messages*
- *DirectPlay Address (Optional)*

- *Migrating from Previous Versions of DirectPlay*
- *DirectPlay Tools and Samples*
- *Security and Authentication*

Debug versus Retail DLLs

The SDK has the option to install debug or retail builds of the DirectPlay DLLs. When developing software, it best to install the debug versions of the DLLs. The debug DLLs have addition code in them which will validate internal data structures and output debug error messages (using the Win32 **OutputDebugString** API) while your program is executing. When an error occurs, the debug output will give you a more detailed description of what the problem is. The debug DLLs will execute more slowly than the retail DLLs but are much more useful for debugging an application. Be sure to ship the retail version with your application.

In order to see the debug messages, it is necessary to configure your computer so that debug output will be displayed in a window or on a remote computer. An interactive development environment like Microsoft Visual C++ allows you to do this. Consult the environment's documentation for exactly how to set this up.

Working with GUIDs

Globally unique identifiers are 16-byte data structures that you can use to identify objects in a globally unique way. Whenever a GUID is required in an API, a symbol representing that GUID should be used. The symbols are either defined in one of the DirectPlay header files or the application developer must generate them. You can generate GUIDs by using the Guidgen.exe utility that comes with Microsoft Visual C++. For example, every application must define an application GUID that identifies the application that is running in a session.

Note If there are different versions of an application that cannot interoperate in the same session, they should have different application GUIDs to distinguish them.

To use DirectPlay-defined GUIDs successfully in your application, you must either define the INITGUID symbol prior to all other include and define statements, or you must link to the Dxguid.lib library. If you define INITGUID, you should define it in only one of your source modules.

DirectPlay Interfaces

All the functionality in DirectPlay is accessed through member functions on COM (Component Object Model) interfaces. To use them, an application must obtain the appropriate COM interface.

The standard method of obtaining COM interfaces is to use the Win32 **CoCreateInstance** API. To use it successfully, the application must first call the Win32 **CoInitialize** API to initialize COM and then call **CoCreateInstance**, specifying the GUID of the desired interface. For example, use the following code to obtain an **IDirectPlay3A** interface:

```
// C++ example
hr = CoCreateInstance( CLSID_DirectPlay, NULL, CLSCTX_INPROC_SERVER,
                       IID_IDirectPlay3A, (LPVOID*)&lpDirectPlay3A);
// C example
hr = CoCreateInstance( &CLSID_DirectPlay, NULL, CLSCTX_INPROC_SERVER,
                       &IID_IDirectPlay3A, (LPVOID*)&lpDirectPlay3A);
```

When the program is finished, all the COM interfaces must be freed by calling the **Release** method on each interface. Finally, the Win32 **CoUninitialize** method should be called to uninitialize COM.

If you call **CoCreateInstance** without first calling **CoInitialize** you will get a CO_E_NOTINITIALIZED error, with the error text "CoInitialize has not been called."

DirectPlay has several COM interfaces. Each interface represents a revision of an earlier version of DirectPlay in which new methods are added. COM interfaces are numbered sequentially with each revision. The latest COM interface will have all the latest functionality of DirectPlay. To access the new functionality (for example, **IDirectPlay3::SendChatMessage**), you must use the latest COM interface. Source code written for a earlier COM interface will work fine.

Once a COM interface is obtained, an alternate interface can be used on the same object by calling the **QueryInterface** method on the interface. For example, **DirectPlayCreate** will create a DirectPlay object and return an **IDirectPlay** interface. If your application requires an **IDirectPlay3** interface, it can call **QueryInterface** on the **IDirectPlay** interface. Be sure to release the original **IDirectPlay** interface if it is no longer needed.

Using Callback Functions

The enumeration methods in DirectPlay are used to return a list of items to the application. The application calls an enumeration method (such as **IDirectPlay3::EnumPlayers**) and supplies a pointer to a callback function that it has implemented. DirectPlay will call the callback function once for each item in the list. The enumeration method will not return until all the items in the list have been returned to the application through the callback function.

It is extremely important that all callbacks be declared correctly. For example:

```
BOOL FAR PASCAL EnumConnectionsCallback(LPCGUID lpguidSP, LPVOID
lpConnection, DWORD dwConnectionSize,
LPCDPNAME lpName, DWORD dwFlags, LPVOID pContext);
```

The FAR PASCAL symbol will define the function as **_stdcall**. That means it will clean up the stack before returning to DirectPlay. Do not cast function pointers when passing them to a DirectPlay enumeration method. If there is a compiler warning about the function pointer, fix the function declaration.

Building Lobby-Aware Applications

A lobby-aware application is one that, at a minimum, supports being launched from a lobby. A matchmaking lobby is a site on the Internet where end users can find other people to play games with. Once a group of people has decided to start an application session, the lobby software can launch the application on each person's computer and have them all connect to a session. The main benefit for end users is the ease with which they can establish a session with other players. Not only does it allow a user to easily find opponents, but there is also no need for the user to:

- Select a service provider. The lobby will specify which service provider to use.
- Decide whether to host or join a session. The lobby will specify whether to create or join a session.
- Enter a network address or configure the network. The lobby will supply this information if it is needed.
- Enter the name of the player. The lobby will pass in the same name that the user connected to the lobby with.

Other benefits of the lobby are:

- It can keep track of sessions in progress and enable users to join them.
- It can receive status messages from the session and display the progress to other users on a scoreboard.
- It can obtain final scores and maintain player rankings for tournament play.

For a DirectPlay application to be lobby-aware, at a minimum it must support being launched from a lobby. You can add other features to make it integrate better with the lobby. For additional information see the following topics within this section.

- *Registering Lobby-Aware Applications*
- *Supporting External Lobby Launching*
- *LobbyMessaging (Optional)*

Registering Lobby-Aware Applications

To enable a lobby to launch a DirectPlay application, the application must add the following entries to the system registry. Once an application has been registered, it will be recognized as a lobby-aware application and lobbies can launch it. These registry entries must be deleted when the application is removed.

The following example shows the registry entries for the DPCHAT sample application included in the SDK.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DirectPlay\Applications\DPChat]
"Guid"="{5BFDB060-06A4-11D0-9C4F-00A0C905425E}"
"File"="dpchat.exe"
"CommandLine"=""
"Path"="C:\DXSDK\sdk\bin"
"CurrentDirectory"="C:\DXSDK\sdk\bin"
```

The keys and values are:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DirectPlay\Applications\	The key's name will be the name of the application that the **IDirectPlayLobby2::EnumLocalApplications** method returns. It is **DPChat** in the preceding example.
Guid	This is the 16-byte GUID that identifies the application. It is formatted as shown in the example. This should be the same GUID that is put in the **guidApplication** member of the **DPSESSIONDESC2** structure when creating a session. A globally unique identifier (GUID) can be generated using the Guidgen.exe utility.
File	This is the file name of the application executable.
CommandLine	This lists any command-line parameters that are to be specified when a lobby launches the application.
Path	This is the directory that the application executable resides in.
CurrentDirectory	This is the directory to set as the current directory after launching the application executable.

Supporting External Lobby Launching

Once the application has been registered, the DirectPlay application must be able to recognize whether a lobby launched it or not. If a lobby launched it, it must follow a slightly different code path to set up the network connection. Consult the DUEL (Lobby.c) and DPCHAT (Lobby.cpp) samples in the SDK for the code necessary to support external lobby launching. See *Tutorial 1: Connecting by Using the Lobby* for a demonstation on how to connect an application by using a DirectPlay lobby.

Here are the basic steps necessary:

1. At startup, create an **IDirectPlayLobby2** interface using the **CoCreateInstance** API.
2. The application can examine the connection information that was passed in by the lobby and modify some of the connection settings if necessary. The **IDirectPlayLobby2::GetConnectionSettings** method returns a **DPLCONNECTION** structure with the connection settings. This method returns **DPERR_NOTLOBBIED** if a lobby did not launch the application. New settings can be set with **IDirectPlayLobby2::SetConnectionSettings**.
3. **IDirectPlayLobby2::Connect** creates or joins the specified application session using the specified service provider, and returns an **IDirectPlay2** interface. **Connect** returns an error if the session could not be created/joined, or if a lobby didn't launch the application.
4. Use **QueryInterface** to obtain an **IDirectPlay3** interface and then call **Release** on the **IDirectPlay2** interface.
5. Create a player using the name information supplied in the **DPLCONNECTION** structure obtained in step 2.

At this point, the application can continue on the same code path as if the user had manually selected a connection, joined or created a session, and entered the name of the player to create.

The **IDirectPlayLobby2** interface can be saved if the game will pass information back to the lobby (see *Lobby Messaging*), or it can be discarded by using the **Release** method if no messages will be sent.

Many lobbies will launch the application and then go into a suspend mode waiting for the application to terminate. DirectPlay will notify the lobby when the application that it launched has terminated. For this reason it is very important that the application launched by the lobby not launch another application.

Lobby Messaging (Optional)

After registering a lobby-aware application and adding the code to support external lobby launching, the next step in integrating the application with the lobby is to send information back to the lobby or request information from the lobby. This is done by exchanging messages with the lobby through the **IDirectPlayLobby2::SendLobbyMessage** and **IDirectPlayLobby2::ReceiveLobbyMessage** methods on the **IDirectPlayLobby2** interface. Standard message structures have been defined by DirectPlay to facilitate this functionality.

Sending information to the lobby is done through setting properties. The application must create and fill in a **DPLMSG_SETPROPERTY** structure and send it to the lobby by using the **SendLobbyMessage** method. Each property identifies a distinct type of data. The application developer should generate GUIDs (using Guidgen.exe) for every property that will be set. The lobby operator needs to obtain this list of GUIDs from the application developer along with the description of each property and the data structure of the property.

Properties can take the form of:

- Current individual player scores.
- Final individual player scores.
- Player configuration for later retrieval.
- Current game status (level or mission).

The lobby can store this information or display it to other users in the lobby so they can monitor the game's progress. An application can request confirmation that the property was set correctly by supplying a nonzero request ID with the set property message.

Requesting information from the lobby is done through requesting properties. The application must create and fill in a **DPLMSG_GETPROPERTY** structure and send it to the lobby by using the **SendLobbyMessage** method. At some later time, the lobby will send a message back to the application — a **DPLMSG_GETPROPERTYRESPONSE** structure that contains the property data that was requested. The application can retrieve the message from the lobby message queue using the **ReceiveLobbyMessage** method. As before, the application developer generates the GUIDs for each property and the lobby operator must obtain them from the developer.

The following list shows examples of some properties that can be requested from the lobby:

- Game configuration settings — enables players to configure the game in the lobby before launching the game.
- Player configuration settings — enables players to configure their players in the lobby or use a stored configuration from a previous session.
- Other information stored from a previous session.

Not all lobbies will be able to support these standard lobby messages. The application can determine if the lobby that it was launched from supports standard lobby messages by sending a **DPLMSG_GETPROPERTY** message requesting the **DPLPROPERTY_MessagesSupported** property.

DirectPlay Messages

The following sections describe how to use messages in DirectPlay.

- *Synchronization*
- *Using System Messages*
- *Using Lobby Messages*

Synchronization

An application can use two methods to process DirectPlay messages. The first is to check the receive queue during the main loop of the application. Typically, this means the application is single threaded.

Alternatively, an application can have a separate thread to wait for messages and process them. In this case the application should supply a non-NULL auto-reset synchronization event handle (see the Win32 **CreateEvent** API) when it creates players. DirectPlay will set this event whenever a message arrives for that player. All the local players in an application can share the same event or each can have his or her own event.

The message processing thread should then use the Win32 **WaitForSingleObject** API to wait until the event is set. Keep calling **Receive** until there are no more messages in the message queue.

Using System Messages

Messages returned by the **IDirectPlay3::Receive** method from player ID DPID_SYSMSG are system messages. All system messages begin with a double-word value specified by the **dwType** DWORD value. You can cast the buffer returned by the **IDirectPlay3::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	Cause
DPSYS_ADDGROUPTOGROUP	**DPMSG_ADDGROUPTOGROUP**	An existing group has been added to an existing group.

DPSYS_ADDPLAYERTOGROUP	**DPMSG_ADDPLAYERTOGROUP**	An existing player has been added to an existing group.
DPSYS_CHAT	**DPMSG_CHAT**	A chat message has been received.
DPSYS_CREATEPLAYERORGROUP	**DPMSG_CREATEPLAYERORGROUP**	A new player or group has been created.
DPSYS_DELETEGROUPFROMGROUP	**DPMSG_DELETEGROUPFROMGROUP**	A group has been removed from a group.
DPSYS_DELETEPLAYERFROMGROUP	**DPMSG_DELETEPLAYERFROMGROUP**	A player has been removed from a group.
DPSYS_DESTROYPLAYERORGROUP	**DPMSG_DESTROYPLAYERORGROUP**	An existing player or group has been destroyed.
DPSYS_HOST	**DPMSG_HOST**	The current host has left the session and this application is the new host.
DPSYS_SECUREMESSAGE	**DPMSG_SECUREMESSAGE**	A digitally signed or encrypted message has been received.
DPSYS_SESSIONLOST	**DPMSG_SESSIONLOST**	The connection with the session has been lost.
DPSYS_SETPLAYERORGROUPDATA	**DPMSG_SETPLAYERORGROUPDATA**	Player or group data has changed.
DPSYS_SETPLAYERORGROUPNAME	**DPMSG_SETPLAYERORGROUPNAME**	Player or group name has changed.
DPSYS_SETSESSIONDESC	**DPMSG_SETSESSIONDESC**	The session description has changed.
DPSYS_STARTSESSION	**DPMSG_STARTSESSION**	The lobby server is requesting that a session be started.

Using Lobby Messages

Messages returned by the **IDirectPlayLobby2::ReceiveLobbyMessage** method fall into two categories: DirectPlay-defined messages and custom-defined messages. The message category can be identified by the *lpdwMessageFlags* parameter of **ReceiveLobbyMessage**. The flags indicate that the message is either a system message (the **DPLMSG_SYSTEM** flag) or a standard message (the **DPLMSG_STANDARD** flag). If neither of these flags is set, the message is custom-defined. System messages are generated automatically by DirectPlay and sent only to the lobby to inform it of changes in the status of the application. Standard messages can be generated by either the lobby or the application and sent to the other.

The advantage of standard messages over custom-defined messages is that the receiver can positively interpret the message. It is not required that all applications or lobbies act on standard messages.

DirectPlay-defined messages all start with a DWORD value that identifies the type of the message. After retrieving a message using **ReceiveLobbyMessage**, the *lpData* pointer to the message data should be cast to the **DPLMSG_GENERIC** structure and the structure's **dwType** member examined. Based on the value of **dwType**, the *lpData* pointer should then be cast to the appropriate message structure for further processing.

Lobbies should be prepared to handle all the following message types. Applications need to handle the **DPLMSG_GETPROPERTYRESPONSE** message if they generate **DPLMSG_GETPROPERTY** messages.

Messages returned by the **IDirectPlayLobby2::ReceiveLobbyMessage** method that have a *dwFlags* parameter set to DPLMSG_SYSTEM are system messages. All system messages begin with a double-word value specified by **dwType**. You can cast the buffer returned by the **IDirectPlayLobby2::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.

The following list shows the possible values of the *dwType* data member, and the message structure and message cause associated with each value.

Value of dwType	Message structure	Cause
DPLSYS_APPTERMINATED	**DPLMSG_GENERIC**	The application has terminated.
DPLSYS_CONNECTIONSETTINGSREAD	**DPLMSG_GENERIC**	The application has read the connection settings.
DPLSYS_DPLAYCONNECTFAILED	**DPLMSG_GENERIC**	The application failed to connect to the DirectPlay session.
DPLSYS_DPLAYCONNECTSUCCEEDED	**DPLMSG_GENERIC**	The application successfully connected to the DirectPlay session.
DPLSYS_GETPROPERTY	**DPLMSG_GETPROPERTY**	The application is requesting a property from the lobby.
DPLSYS_GETPROPERTYRESPONSE	**DPLMSG_GETPROPERTYRESPONSE**	The lobby is responding to a prior DPLMSG_GETPROPERTY message.
DPLSYS_SETPROPERTY	**DPLMSG_SETPROPERTY**	The application is setting a property on the lobby.
DPLSYS_SETPROPERTYRESPONSE	**DPLMSG_SETPROPERTYRESPONSE**	The lobby is responding to a prior DPLMSG_SETPROPERTY message.

DirectPlay Address (Optional)

A DirectPlay Address is a data format that DirectPlay uses to pass addressing information between lobby servers, applications, DirectPlay, and service providers. The format is flexible enough to hold any number of variable-length data fields that make up a network address. It is not necessary to understand DirectPlay Addresses in order to use DirectPlay. In fact, it is possible to successfully write a DirectPlay application without understanding DirectPlay Addresses at all.

It is only necessary to learn about DirectPlay Addresses if you want to do the following things.

- Override the standard service provider dialog boxes. You need to create a DirectPlay Address that specifies which service provider to use and contains all the information the service provider needs to establish a connection. You then pass the DirectPlay Address to the **IDirectPlay3::InitializeConnection** method.
- Write a lobby client that will launch an external DirectPlay application. The lobby client needs to create a DirectPlay Address that specifies which service provider to use and contains all the information the service provider needs to establish a connection and automatically connect the client to a session. You need to create a DirectPlay Address of the session to be joined and put it in the **DPLCONNECTION** structure before calling the **IDirectPlayLobby2::RunApplication** method.

A DirectPlay Address is a sequence of variable-length data chunks tagged with a GUID that supply all the address information needed by DirectPlay. Examples are the network address of a server, the network address of a player, the email name of a player, or the network address of a session.

DirectPlay Address Data Types

Microsoft has predefined the following general data types for each data chunk. Based on the data type, the data must be cast to the appropriate type or structure to interpret the data.

GUID	**Type of data.**
DPAID_ComPort	A **DPCOMPORTADDRESS** structure that contains all the settings for the COM port. The *serial connection* service provider will use this information to configure the serial port.
DPAID_INet	A null-terminated ANSI string (LPSTR) containing an IP address ("137.55.100.173") or a domain net ("gameworld.com"). The length in bytes must include the terminator. A blank address is a string that contains only the ANSI terminator (0x00) and has a length of 1 byte. If a blank

	address is supplied, the Internet TCP/IP Connection service provider will use this information to enumerate sessions on the specified network address or broadcast on the subnet.
DPAID_INetW	A null-terminated Unicode string (LPWSTR) containing an IP address ("137.55.100.173") or a domain net ("gameworld.com"). The length in bytes must include the terminator. A blank address is a string that contains only the Unicode terminator (0x0000) and has a length of 2 bytes. If a blank address is supplied, the Internet TCP/IP Connection service provider will use this information to enumerate sessions on the specified network address or broadcast on the subnet.
DPAID_Modem	A variable-length null-terminated ANSI string (LPSTR) specifying which installed modem to use. The length in bytes must include the terminator. The modem service provider will use this modem without displaying a dialog box asking the user which modem to use. Use the **IDirectPlay3::GetPlayerAddress** method to determine which modems are available.
DPAID_ModemW	A variable-length null-terminated Unicode string (LPWSTR) specifying which installed modem to use. The length in bytes must include the terminator. The modem service provider will use this modem without displaying a dialog box asking the user which modem to use. Use the **IDirectPlay3::GetPlayerAddress** method to determine which modems are available.
DPAID_Phone	A variable-length null-terminated ANSI string (LPSTR) containing a phone number. The length in bytes must include the terminator. The modem service provider will call this phone number on the **IDirectPlay3::EnumSessions** method. If no modem is specified, the first modem will be used.
DPAID_PhoneW	A variable-length null-terminated Unicode string (LPWSTR) containing a phone number. The length in bytes must include the terminator. The modem service provider will call this phone number on the **IDirectPlay3::EnumSessions** method. If no modem is specified, the first modem will be used.
DPAID_ServiceProvider	The 16-byte GUID that specifies the service provider this DirectPlay Address applies to.

Using DirectPlay Addresses

You can use a DirectPlay Address to encapsulate all the information necessary to initialize a DirectPlay object. At a minimum, this is the GUID of a DirectPlay provider, but can also include the network address of an application or lobby server and even a specific session instance GUID.

A DirectPlay Address can be used to supply enough information to DirectPlay (and the DirectPlay provider) when it is initialized so that no dialog boxes appear later during the process of establishing a session or connecting to a session.

The DirectPlay Addresses returned by *EnumConnections* are simply registered DirectPlay providers that the user can choose from. When one of these is initialized, dialog boxes can appear asking the user for more information.

The application can create a DirectPlay Address directly using the **IDirectPlayLobby2::CreateAddress** and **IDirectPlayLobby2::CreateCompoundAddress** methods on the **IDirectPlayLobby2** interface and pass the connection to the **IDirectPlay3::InitializeConnection** method to initialize the DirectPlay object. If enough valid information is supplied, then no DirectPlay dialog boxes will appear.

Examples of Using DirectPlay Addresses

This topic contains examples of DirectPlay Addresses and the data they contain for different connection types.

A DirectPlay Address describing an IPX connection

Initializing this connection will bind the DirectPlay object to the IPX service provider.

guidDataType	dwDataSize	Data
DPAID_ServiceProvider	16	{685BC400-9D2C-11cf-A9CD-00AA006886E3}

A DirectPlay Address describing a modem connection

Initializing this connection will bind DirectPlay to the modem service provider and store the phone number. A subsequent call the **IDirectPlay3::EnumSessions** will dial the number without asking the user for a phone number.

guidDataType	dwDataSize	Data
DPAID_ServiceProvider	16	{44EAA760-CB68-11cf-9C4E-00A0C905425E}
DPAID_Phone	9 (including NULL terminator)	"555-8237"

A DirectPlay Address describing a TCP/IP connection

Initializing this connection will bind DirectPlay to the TCP/IP service provider and store the IP address. A subsequent call **IDirectPlay3::EnumSessions** will enumerate sessions on this server without asking the user for an IP address.

guidDataType	dwDataSize	Data
DPAID_ServiceProvider	16	{36E95EE0-8577-11cf-960C-0080C7534E82}

DPAID_INet	10 (including NULL terminator)	"127.0.0.1"

Migrating from Previous Versions of DirectPlay

The DirectPlay version 5 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 5 SDK without making any changes to the code. DirectPlay supplied with the DirectX 5 SDK supports all the APIs and behavior of earlier DirectPlay versions.

For specific information, see:

- *Migrating from DirectPlay 3*
- *Migrating from DirectPlay 2 or Earlier*

Migrating from DirectPlay 3

If you are migrating to DirectPlay version 5 from DirectPlay version 3, you do not have to make any changes. However, you should upgrade your application to use the new *IDirectPlay3* or **IDirectPlay3A** interfaces.

For a list of new features in version 5 of DirectPlay, see *What's New in DirectPlay 5*.

Some system messages have changed for DirectPlay 5, with new data members added to the end. Applications that need to be backward compatible with older run times should either:

- compile with the DirectX 3 header files and libraries
- not reference new data members
- test with both the DirectX 3 and the DirectX 5 runtime

The system message structures that have new members for DirectPlay 5 are:

Structure	**New Members**
DPMSG_CREATEPLAYERORGROUP	DPID dpIdParent
	DWORD dwFlags
DPMSG_DESTROYPLAYERORGROUP	DPNAME dpnName
	DPID dpIdParent
	DWORD dwFlags

Migrating from DirectPlay 2 or Earlier

The names of the DirectPlay DLLs in version 3 and later are different from those in previous DirectPlay versions. 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 also recommended that you add the code necessary to make the application lobby aware. 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 were added in the DirectPlay version 5 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 other features in DirectPlay 5 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 added to DirectPlay 5 are not directly related to applications, 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.

Migrating to the IDirectPlay3 Interface

To migrate an application created with DirectPlay version 2 or earlier, 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*.
2. If your application is enumerating service providers, use the **EnumConnections** method to determine if a service provider is available. If so, call the **InitializeConnection** method on the service provider. If **InitializeConnection** 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 **IDirectPlay3** (Unicode) or **IDirectPlay3A** (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;
   lpDP3A->SetPlayerName(pidPlayer, lpPlayerName, 0)
   ```

   where *lpDP3A* is an **IDirectPlay3A** interface. If the application is using Unicode strings (and therefore instantiates an **IDirectPlay3** interface), use the following:

   ```
   lpPayerName->lpszShortName = lpwszFriendlyName;
   lpPlayerName->lpszLongName = lpwszFormalName;
   ```

```
lpDP3->SetPlayerName(pidPlayer, lpPlayerName, 0)
```

where *lpDP3* is an **IDirectPlay3** 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 **IDirectPlay3::EnumSessions**, **IDirectPlay3::EnumGroups**, **IDirectPlay3::EnumGroupPlayers**, and **IDirectPlay3::EnumPlayers**.

9 Update the manner in which the *hEvent* parameter is supplied to the **IDirectPlay3::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 **IDirectPlay3** interface and use them in your application. Pay particular attention to the **IDirectPlay3::SetPlayerData** and **IDirectPlay3::GetPlayerData** methods. You might be able to substitute the code where you broadcast player state information to all the other players by using the **IDirectPlay3::Send** and **IDirectPlay3::Receive** methods.

DirectPlay Tools and Samples

The following samples can be examined for examples of how to use DirectPlay. These DirectPlay samples are included on the DirectX 5 CD. The Control Panel tool can be used to find out about DirectPlay applications.

DirectX Control Panel Tool

- Found in Control Panel. Double-click to open the DirectX Properties dialog box.
- The DirectPlay tab displays all the registered DirectPlay service providers and applications.
- Open the DirectX Properties dialog box before starting a DirectPlay application to see on-the-fly statistics about DirectPlay communications such as bytes/second and messages/second.

SDK/SAMPLES/BELLHOP

- A lobby client program the uses the IDirectPlay3 interface to communicate with a lobby server.
- Uses the LSERVER test lobby server so that a real lobby client/server environment can be set up.
- BELLHOP can be used to test external lobby support in your application.
- LSERVER can be used as a test server to write your own lobby client or integrate a lobby client into your application.

SDK/SAMPLES/DPCHAT

- A simple Windows-based chat program that uses IDirectPlay3
- Uses IDirectPlayLobby to make it lobby-able

SDK/SAMPLES/DPLAUNCH

- A stand-alone application that demonstrates how a DirectPlay 5 application can be launched from an external source using the IDirectPlayLobby2 interface
- Can also be used to test the lobby support in your application

SDK/SAMPLES/DPSLOTS

- A DirectPlay client/server application that uses security.
- The server controls all the slot machines and tracks how much money each player has.
- The client securely logs in to the server.

SDK/SAMPLES/DUEL

- Multiplayer game that uses the IDirectPlay interface

- Uses IDirectPlayLobby to make it lobby-able

SDK/SAMPLES/DXVIEW

- An application that shows all the available service providers and their capabilities.
- Shows all the registered lobby-aware applications installed on the system.
- Uses asynchronous **EnumSessions** to monitor the active sessions on the service provider.

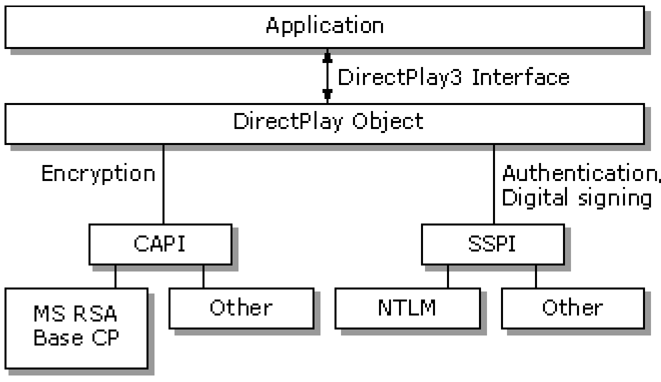
SDK/SAMPLES/OVERRIDE

- A simple demonstration of how to override the DirectPlay service provider dialogs.

Security and Authentication

DirectPlay security allows an application running on a server to authenticate users against a database of known users before allowing them to join a session. Once a user has been authenticated, all communications between the client and server can be done securely either by digitally signing messages (to verify the identity of the sender) or by encrypting the messages.

The following diagram shows DirectPlay security architecture. (SSPI = Security Support Provider Interface, CAPI = CryptoAPI, MS RSA Base CP = Microsoft RSA Base Cryptographic Provider, NTLM = NT LAN Manager.)



User and Message Authentication

DirectPlay provides user and message authentication (digital signing) support through the Windows Security Support Provider Interface (SSPI). This is a standard interface that gives software access to various security packages under the Windows Operating System. A security package is an implementation of a protocol between a client and server that establishes the identity of the client and provides other security services, such as digital signing. The default security package that DirectPlay uses is called NTLM (Windows NT LAN Manager) Security Support Provider.

This security package is based on the NTLM authentication protocol. NTLM is a shared-secret, user challenge-response authentication protocol that supports pass-through authentication to a domain controller in the server's domain or in a domain trusted by the current domain's domain controller. This protocol provides a high level of security, as passwords are never sent out on the network. NTLMSSP ships with the Windows 95 and Windows NT operating systems.

A DirectPlay application can choose to use a different SSPI security package when it creates a session by calling the **SecureOpen** method. For example, DPA (Distributed Password Authentication) Security Support Provider is another security package that organizations can use to provide membership services to a large customer base (hundreds of thousands). This security package is available through the Microsoft Commercial Internet Services (MCIS) Membership Server.

Message Privacy (encryption/decryption)

DirectPlay provides message encryption support through the Windows Cryptography Application Programming Interface (CAPI). This is a standard interface similar to SSPI that gives software access to various cryptographic packages under the Windows Operating System. This architecture allows DirectPlay applications to plug in cryptographic packages that provide the desired level of encryption (40 bit, 128 bit, and so on) legally allowed in the locale of use.

The default CryptoAPI (CAPI) provider for DirectPlay cryptography services is the Microsoft RSA Base Cryptographic Provider v. 1.0. The default CAPI provider type is PROV_RSA_FULL. The default encryption algorithm is the CALG_RC4 stream cipher. This provider is supplied by Microsoft and is included with Internet Explorer for Windows 95, Windows 95 OSR-2 Update, and Windows NT 4.0 operating system.

A DirectPlay application can choose to use a different Cryptographic provider when it creates a session by using the SecureOpen method. Please note that DirectPlay only supports stream ciphers.

For more information about SSPI, NTLM, DPA, MCIS, CAPI, and the RSA Base Cryptographic Provider, see http://www.microsoft.com.

Secure Sessions

User authentication should not be confused with password protection of a session. Authentication is used to verify that the user is allowed access to the server by virtue of having been added to the membership database by the administrator of the server. Only users that are part of the membership database are permitted to join the session. A password can be added to a session, so that only those members who know the password can join a particular session. Additionally, authentication requires a server that supports authentication, while any computer can put a password on a session.

For example, a server on the Internet might have a membership of a thousand users. Anybody can enumerate the sessions that are available on the server but only members will be able to join sessions on the server. Users who want only their friends (who are members) to be able to join can put a password on their session.

Once a secure server has been set up and an initial membership list has been established, a secure DirectPlay session can be started on it. Creating a secure DirectPlay session simply requires the server to create a session using **IDirectPlay3::Open** or **IDirectPlay3::SecureOpen** and specify the **DPSESSION_SECURESERVER** flag in the **DPSESSIONDESC2** structure.

A DirectPlay application can choose to use alternate providers when it creates a session by calling the **SecureOpen** method and specifying the providers to use in the **DPSECURITYDESC** structure. For a different SSPI provider (for user and message authentication), an application needs to specify the provider name in the *lpszSSPIProvider* member of the **DPSECURITYDESC** structure. For a different CryptoAPI provider (for message privacy), an application needs to specify the provider name, provider type, and encryption algorithm in the *lpszCAPIProvider*, *dwCAPIProviderType*, and *dwEncryptionAlgorithm* members respectively.

When a client enumerates this session, the **DPSESSIONDESC2** structure returned by **IDirectPlay3::EnumSessions** will have the **DPSESSION_SECURESERVER** flag set. This tells the client that authentication credentials will be required to join the session. If the client attempts to join the session using **IDirectPlay3::Open**, the security package may allow the user in if the user's credentials were already established through a system logon (for example, through NT LAN Manager). Otherwise, a **DPERR_LOGONDENIED** error is returned. The application must then collect credentials from the user, put them in a **DPCREDENTIALS** structure, and try to join the session again by calling **SecureOpen**, passing in the **DPCREDENTIALS** structure. **SecureOpen** is recommended for security.

Within a secure session, all DirectPlay system messages are digitally signed to verify the identity of the sender. Certain system messages that carry sensitive information are encrypted. System messages that originate from one player and need to be broadcast to all the other players in the session are first sent to the server. The server then puts its signature on the message and forwards the message to all the other computers in the session. Player to player messages are not signed by default and are not routed through the server.

An application can choose to sign or encrypt specific player-to-player messages using the **DPSEND_SIGNED** and **DPSEND_ENCRYPTED** flags in the **IDirectPlay3::Send** method. Signed and encrypted player messages are routed through the server and delivered as secure system messages. When a secure message is received by a player, the message will contain flags indicating whether the message came in signed or encrypted. Messages that don't pass the verification are dropped.

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.

New methods and functionality are exposed in COM through the use of new interfaces that are sequentially numbered. The current DirectPlay interface is called **IDirectPlay3**. The old interfaces, **IDirectPlay** and **IDirectPlay2**, still exist for backward compatibility with applications written to those interfaces.

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

- *Unicode vs. ANSI Interfaces*
- *IDirectPlay Interface*
- *IDirectPlay2 Interface*
- *IDirectPlay3 Interface*
- *IDirectPlayLobby Interface*
- *IDirectPlayLobby2 Interface*

Unicode vs. ANSI Interfaces

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 member ends with the letter "A". Depending on which **IDirectPlay** interface is chosen (*IDirectPlay3* for Unicode or **IDirectPlay3A** for ANSI), or which **IDirectPlayLobby2** interface is chosen (*IDirectPlayLobby2* for Unicode or **IDirectPlayLobby2A** for ANSI), the application should read and write the appropriate strings from the structure and ignore the other one.

IDirectPlay Interface

The **IDirectPlay** COM interface remains part of DirectPlay version 5. 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, *IDirectPlay3* and **IDirectPlay3A**, to take advantage of their increased functionality.

IDirectPlay2 Interface

The **IDirectPlay2** COM interface remains part of DirectPlay version 5. It contains the methods required to run applications that were written for the DirectX SDK versions 3. Although you could use this interface to create new applications, it is recommended that you use the newer DirectPlay interfaces, *IDirectPlay3* and *IDirectPlay3A*, to take advantage of their increased functionality.

IDirectPlay3 Interface

This is the latest interface. **IDirectPlay3** directly inherits from **IDirectPlay2** so any code written for **IDirectPlay2** will work with **IDirectPlay3** with no modification.

The new methods in **IDirectPlay3** are:

- *AddGroupToGroup*
- *CreateGroupInGroup*
- *DeleteGroupFromGroup*
- *EnumConnections*
- *EnumGroupsInGroup*
- *GetGroupConnectionSettings*
- *GetGroupFlags*
- *GetGroupParent*
- *GetPlayerAccount*
- *GetPlayerFlags*
- *InitializeConnection*
- *SecureOpen*
- *SendChatMessage*
- *SetGroupConnectionSettings*
- *StartSession*

IDirectPlayLobby Interface

The **IDirectPlayLobby** COM interface remains part of DirectPlay version 5. 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 interface, *IDirectPlayLobby2*, to take advantage of its increased functionality.

IDirectPlayLobby2 Interface

The *IDirectPlayLobby2* interface lets game developers launch external applications, enable communication between an application and a lobby client, and manipulate DirectPlay Addresses.

IDirectPlayLobby2 supports all the methods of **IDirectPlayLobby**. See *Building Lobby-Aware Applications* for information about detecting a launch by an external lobby and registering lobby-aware applications. See *DirectPlay Lobby Overview* for general information about lobby sessions.

The new method in **IDirectPlayLobby2** is:

- *CreateCompoundAddress*

DirectPlay Tutorials

This section contains four tutorials that provide step-by-step instructions about how to connect an application with or without a lobby, how to override service provider dialog boxes, and how to create a self-refreshing session list.

The first tutorial demonstrates how to connect an application by using a DirectPlay lobby. The second tutorial 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. The code is available in the DPCHAT sample in the LOBBY.CPP and DIALOG.CPP files.

The third tutorial demonstrates the calls you need to supply the service provider with all the information it needs so that it doesn't display dialog boxes to the user requesting information.

The fourth tutorial demonstrates how to create a self-refreshing session list.

- *Tutorial 1: Connecting by Using the Lobby* (DPCHAT)
- *Tutorial 2: Connecting by Using a Dialog Box* (DPCHAT)
- *Tutorial 3: Overriding the Service Provider Dialogs*
- *Tutorial 4: Creating Self-Refreshing Session Lists*

Note 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 vtables and **this** pointers to the interface methods. For more information, see .

Tutorial 1: Connecting by Using the Lobby

An application written to use the *IDirectPlayLobby2* interface can be connected to a session without requiring the user to manually enter connection information in a dialog box. To demonstrate how to create a lobbied application, the DPCHAT 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 **IDirectPlayLobby2::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 settings
hr = lpDirectPlayLobbyA->GetConnectionSettings(0, NULL, &dwSize);
if (DPERR_BUFFERTOOSMALL != hr)
    goto FAILURE;

// Allocate memory for the connection settings.
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 **IDirectPlayLobby2::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 **IDirectPlayLobby2::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 IDirectPlay3A interface.
hr = lpDirectPlayLobbyA->Connect(0, &lpDirectPlay2A, NULL);
if FAILED(hr)
    goto FAILURE;
// Obtain an IDriectPlay3A interface
hr= lpDirectPlay2A->QueryInterface(IID_DirectPlay3A,
(LPVOID*)&lpDirectPlay3A);
if FAILED(hr)
    goto FAILURE;
```

Step 5: Creating a Player

If the application was successfully started by using the **IDirectPlayLobby2::Connect** method, it can now create one or more players. It can use the **IDirectPlay3::CreatePlayer** method to create a player with the name specified in the **DPNAME** structure (which was filled in by the **IDirectPlayLobby2::GetConnectionSettings** method).

The following example shows how to create a player:

```
// create a player with the name returned in the connection settings
hr = lpDirectPlay3A->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 DPCHAT sample performs the following steps:

- *Step 1: Creating the DirectPlay Object*
- *Step 2: Enumerating and Initializing the Service Providers*
- *Step 3: Joining a Session*

- *Step 4: Creating a Session*
- *Step 5: Creating a Player*

Step 1: Creating the DirectPlay Object

Before any methods can be called, the application must create an interface to a DirectPlay object.

The following example shows how the create the *IDirectPlay3A* interface:

```
HRESULT CreateDirectPlayInterface( LPDIRECTPLAY3A *lplpDirectPlay3A )
{
   HRESULT         hr;
   LPDIRECTPLAY3A  lpDirectPlay3A = NULL;

   // Create an IDirectPlay3 interface
   hr = CoCreateInstance( CLSID_DirectPlay, NULL, CLSCTX_INPROC_SERVER,
                         IID_IDirectPlay3A, (LPVOID*)&lpDirectPlay3A);

   // Return interface created
   *lplpDirectPlay3A = lpDirectPlay3A;

   return (hr);
}
```

Step 2: Enumerating and Initializing the Service Providers

The next 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 **EnumConnections** method.

The following example shows how to enumerate the service providers:

```
lpDirectPlay3A->EnumConnections(&DPCHAT_GUID,
DirectPlayEnumConnectionsCallback, hWnd, 0);
```

The second parameter in the **EnumConnections** method 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 DirectPlayEnumConnectionsCallback(
     LPCGUID lpguidSP, LPVOID lpConnection, DWORD dwConnectionSize,
     LPCDPNAME lpName, DWORD dwFlags, LPVOID lpContext)
{
   HWND     hWnd = (HWND) lpContext;
   LRESULT  iIndex;
   LPVOID   lpConnectionBuffer;
   // Store service provider name in combo box
   iIndex = SendDlgItemMessage(hWnd, IDC_SPCOMBO, CB_ADDSTRING, 0,
       (LPARAM) lpName->lpszShortNameA);
   if (iIndex == CB_ERR)
     goto FAILURE;

   // make space for connection
   lpConnectionBuffer = GlobalAllocPtr(GHND, dwConnectionSize);
   if (lpConnectionBuffer == NULL)
     goto FAILURE;

   // Store pointer to connection  in combo box
   memcpy(lpConnectionBuffer, lpConnection, dwConnectionSize);
   SendDlgItemMessage(hWnd, IDC_SPCOMBO, CB_SETITEMDATA, (WPARAM) iIndex,
       (LPARAM) lpConnectionBuffer);

FAILURE:
    return (TRUE);
}
```

Once the user selects which connection to use, the DirectPlay object must be initialized with the connection buffer associated with it.

```
hr = lpDirectPlay3A->InitializeConnection(lpConnection, 0);
```

Step 3: Joining a Session

If the user wants to join an existing session, enumerate the available sessions by using the **IDirectPlay3::EnumSessions** method, present the choices to the user, and then connect to that session by using the **IDirectPlay3::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.

See *Tutorial 4* for details on the asynchronous **EnumSessions** functionality.

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 = lpDirectPlay3A->EnumSessions(&sessionDesc, 0, EnumSessionsCallback,
    hWnd, DPENUMSESSIONS_AVAILABLE);
if FAILED(hr)
    goto FAILURE;
```

In the previous example, the third parameter in the **IDirectPlay3A::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 = lpDirectPlay3A->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 **IDirectPlay3::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 = lpDirectPlay3A->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 **IDirectPlay3::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 = lpDirectPlay3A->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 **IDirectPlay3::GetCaps** and **IDirectPlay3::GetPlayerCaps** methods. Your application can find other players by using the **IDirectPlay3::EnumPlayers** method.

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

Tutorial 3: Overriding the Service Provider Dialogs

DirectPlay now gives applications the ability to suppress the standard service provider dialogs. Below is a brief outline of how this is to be done. A code example can be found in the OVERRIDE sample application.

It is generally not possible to suppress all service provider dialogs. The standard TCP/IP, modem, and serial service provider dialogs can be suppressed (IPX has no dialog box). However, there is the possibility that third party service providers might require fairly complex information from the user which cannot be overridden in any general way. The solution is to simply allow these dialog boxes to appear over your application user interface. If the application is a DirectDraw full-screen application, be sure to turn off page flipping before calling **IDirectPlay3::EnumSessions** or **IDirectPlay3::Open** to create a session.

Another way to suppress service provider dialog boxes is to make your application lobby-aware. Most third party service providers will also have a lobby from which to launch games, and games launched from a lobby do not display a connection dialog box.

An application first calls **IDirectPlay3::EnumConnections** to see what connections are available, presents the list to the user, and allows the user to select one. Once the user has selected one, the application can attempt to override the dialog box before calling **IDirectPlay3::EnumSessions** or **IDirectPlay3::Open**.

These are the steps you should follow to suppress service provider dialog boxes:

1. Examine the service provider GUID of the selected service provider to see if it matches one of the known service providers that the application knows how to override. If the service provider GUID is unknown then skip the remaining steps and be prepared to allow a dialog box to appear.
2. Display the appropriate user interface to collect the information needed from the user for that specific service provider. **IPX** requires no information. **TCP/IP** requires an IP address for **EnumSessions** (DPAID_Inet or DPAID_InetW) but requires nothing to create a session using **Open**. **Modem-to-modem** requires the user to select a modem (DPAID_Modem or DPAID_ModemW), and also needs a phone number (DPAID_Phone or DPAID_PhoneW) when calling **EnumSessions**. **Serial link** needs the **DPCOMPORTADDRESS** structure (DPAID_ComPort) filled in to configure the COM port for both **EnumSessions** and **Open**.
3. Build a DirectPlay Address using the **IDirectPlayLobby2::CreateCompoundAddress** method. The address elements that must be passed in are the service provider GUID (DPAID_ServiceProvider) and the individual address components for the selected service provider.
4. Initialize the DirectPlay object by calling **InitializeConnection** with the DirectPlay Address.

5 Call **EnumSessions** with the DPENUMSESSIONS_RETURNSTATUS flag. This will prevent any status dialog boxes from appearing and, if the connection cannot be made immediately, **EnumSessions** will return with a DPERR_CONNECTING error. Your application must periodically call **EnumSessions** until DP_OK is returned (meaning the enumeration was successful) or some other error is returned (meaning it failed).

6 If a session is to be created using the **Open** method with DPOPEN_CREATE, specify the DPOPEN_RETURNSTATUS flag as well. Like DPENUMSESSIONS_RETURNSTATUS, this will suppress status dialog boxes and return DPERR_CONNECTING until the function is complete.

Note: In some cases, the application will need to query the service provider at runtime to obtain a list of valid choices for a particular DirectPlay Address element. For example, to obtain a list of the modems installed in the system. The application must create a separate DirectPlay object, initialize the modem service provider and then call **IDirectPlay3::GetPlayerAddress** with a DPID of zero to obtain a DirectPlay Address that will contain the list of modems. After releasing the DirectPlay object, the application must parse the address using **IDirectPlayLobby2::EnumAddress** and extract the modem list to present to the user.

Tutorial 4: Creating Self-Refreshing Session Lists

IDirectPlay3::EnumSessions can now be called asynchronously. This gives an application the ability to maintain a self-refreshing session list. A code example can be found in the DUEL and DXVIEW sample applications.

The steps you need to follow to create a self-refreshing session list are:

1 Call **IDirectPlay3::EnumSessions** with the DPENUMSESSIONS_ASYNC flag and a time-out of zero (which will use the service provider default). The method will not enumerate any sessions and will return immediately. However, DirectPlay is enumerating sessions in the background.

2 Display the user interface in which all the sessions will appear. Set a timer to go off at whatever interval you want to refresh your session list. The application can find out what the default time-out interval of the enumeration is by calling **IDirectPlay3::GetCaps**.

3 Each time the timer goes off, call **EnumSessions** to obtain the current session list. This is a complete active session list with stale sessions deleted, new sessions added, and existing sessions updated. Delete all the items from the list before

calling **EnumSessions** and add the sessions back to the list in the **EnumSessionsCallback2** function.

DirectPlay Reference

Functions

DirectPlayCreate

This function is obsolete and remains for compatibility with applications written using DirectX 3. It is recommended that applications create the desired DirectPlay interface directly using **CoCreateInstance**. By using **CoCreateInstance** you can obtain an **IDirectPlay3** interface directly rather than getting an **IDirectPlay** interface, having to use **QueryInterface** to access an **IDirectPlay3** interface, and releasing the **IDirectPlay** interface.

Creates a new DirectPlay object and obtains an **IDirectPlay** interface pointer. If the application supplies GUID_NULL for the *lpGUIDSP* parameter, this function creates a DirectPlay object but does not initialize a service provider. The application can then call the **IDirectPlay3::InitializeConnection** method to initialize the service provider or lobby provider.

The application can supply a service provider GUID (see **DirectPlayEnumerate**) to indicate which service provider to bind.

In order to use the latest DirectPlay functionality, the application must obtain an **IDirectPlay3** or *IDirectPlay3A* interface pointer using the **QueryInterface** method.

```
HRESULT WINAPI DirectPlayCreate(
  LPGUID lpGUIDSP,
  LPDIRECTPLAY FAR *lplpDP,
  IUnknown *lpUnk
  );
```

Parameters

lpGUIDSP
Pointer to the GUID of the service provider that the DirectPlay object should be initialized with. Pass in a pointer to GUID_NULL to create an uninitialized DirectPlay object.

lplpDP
Pointer to an interface pointer to be initialized with a valid IDirectPlay interface. The application will need to use the **QueryInterface** method to obtain an **IDirectPlay2**, *IDirectPlay3* (UNICODE strings) or **IDirectPlay2A**, *IDirectPlay3A* (ANSI strings) interface.

lpUnk
Pointer to the containing *IUnknown*. 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.

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

CLASS_E_NOAGGREGATION

DPERR_EXCEPTION

DPERR_INVALIDPARAMS

DPERR_UNAVAILABLE

DPERR_UNAVAILABLE is returned if a DirectPlay object could not be created. DPERR_INVALIDPARAMS is returned if the GUID provided is invalid.

Remarks 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 *lpGUIDSP* parameter of **DirectPlayCreate**).

This function returns a pointer to an **IDirectPlay** interface. The current interfaces for DirectX 5 are **IDirectPlay3** and *IDirectPlay3A*, which need to be obtained through a call to the **QueryInterface** method on the **IDirectPlay** interface returned by **DirectPlayCreate**.

See Also **IDirectPlay3::InitializeConnection**, **DirectPlayEnumerate**

DirectPlayEnumerate

Enumerates the DirectPlay service providers installed on the system.

This function is obsolete and remains for compatibility with applications written using DirectX 3. It will only enumerate service providers, not DirectPlay Addresses (connections). It is recommended that applications use the **IDirectPlay3::EnumConnections** method to enumerate all the connections available to the application after creating an **IDirectPlay3** interface.

This function will not enumerate service providers that have the **Enumerate** value in the registry set to zero. For backward compatibility, this function will only return simple service providers registered under the "Service Providers" registry key.

HRESULT WINAPI DirectPlayEnumerate(
    LPDPENUMDPCALLBACK *lpEnumCallback*,
    LPVOID *lpContext*
    );

Parameters

lpEnumCallback
  Pointer to a callback function that will be called with a description of each DirectPlay service provider installed in the system. Depending on whether UNICODE is defined or not, the prototype for the callback function will have the service provider name *lpSPName* defined as a LPWSTR (for Unicode) or LPSTR (for ANSI).

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

Return Values

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

DPERR_EXCEPTION

DPERR_GENERIC

DPERR_INVALIDPARAMS

DPERR_INVALIDPARAMS is returned if an invalid enumeration callback was supplied.

Remarks

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.

See Also

IDirectPlay3::EnumConnections, **DirectPlayCreate**

DirectPlayLobbyCreate

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

HRESULT WINAPI DirectPlayLobbyCreate(
    LPGUID *lpGUIDSP*,
    LPDIRECTPLAYLOBBY **lplpDPL*,

IUnknown **lpUnk*,
LPVOID *lpData*,
DWORD *dwDataSize*
);

Parameters

lpGUIDSP
Reserved for future use; must be set to NULL.

lplpDPL
Pointer to a pointer to be initialized with a valid **IDirectPlayLobby** interface. To get an **IDirectPlayLobby2** interface, use this function to get an **IDirectPlayLobby** interface, then call:

```
  IDirectPlayLobby->QueryInterface( IID_IDirectPlayLobby2, (LPVOID*)
&lpDP2 );
```

lpUnk
Pointer to the containing *IUnknown* interface. This parameter is provided for future compatibility 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.

Return Values

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

CLASS_E_NOAGGREGATION

DPERR_INVALIDPARAMS

DPERR_OUTOFMEMORY

Callback Functions

EnumAddressCallback

Application-defined callback function for the **IDirectPlayLobby2::EnumAddress** method.

BOOL FAR PASCAL EnumAddressCallback(
REFGUID *guidDataType*,
DWORD *dwDataSize*,
LPCVOID *lpData*,
LPVOID *lpContext*

);

Parameters

guidDataType
: Pointer to a globally unique identifier (GUID) indicating the type of this data chunk. For example, this parameter might be &DPAID_Phone or &DPAID_INet. (In C++, it is a reference to the GUID.)

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

lpData
: Pointer to the constant data.

lpContext
: Context passed to the callback function.

Return Values

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

Remarks

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

Application-defined callback function for the **IDirectPlayLobby2::EnumAddressTypes** method.

BOOL FAR PASCAL EnumAddressTypeCallback(
  REFGUID *guidDataType***,**
  LPVOID *lpContext***,**
  DWORD *dwFlags*
  );

Parameters

guidDataType
: Pointer to a globally unique identifier (GUID) indicating the address type. (In C+ +, it is a reference to the GUID.) Predefined address types are DPAID_Phone, DPAID_INet, and DPAID_ComPort. For more information about these address types, see *DirectPlay Address (Optional)*.

lpContext
: Context passed to the callback function.

dwFlags
Reserved; do not use.

Return Values
Returns TRUE to continue the enumeration or FALSE to stop it.

EnumConnectionsCallback

Application-defined callback function for the **IDirectPlay3::EnumConnections** method.

BOOL FAR PASCAL EnumConnectionsCallback(
LPCGUID *lpguidSP***,**
LPVOID *lpConnection***,**
DWORD *dwConnectionSize***,**
LPCDPNAME *lpName***,**
DWORD *dwFlags***,**
LPVOID *lpContext*
);

Parameters

lpguidSP
The GUID of the DirectPlay service provider or lobby provider associated with the connection. Use this GUID to uniquely identify the service or lobby provider, rather than using the order in the enumeration or the name.

lpConnection
A read-only pointer to a buffer that contains the connection. This parameter is passed to the **IDirectPlay3::InitializeConnection** method to initialize the DirectPlay object. This buffer contains a *DirectPlay Address*.

dwConnectionSize
The size, in bytes, of the *lpConnection* buffer.

lpName
A read-only pointer to a **DPNAME** structure. The structure contains the short name of the connection that should appear to the user.

If **IDirectPlay3::EnumConnections** was called on an ANSI interface, reference the strings as ANSI. If **EnumConnections** was called on a Unicode interface, reference the strings as Unicode.

dwFlags
Flags to indicate the type of connection. Not used at this time.

lpContext
Pointer to an application-defined context.

Return Values
Returns TRUE to continue the enumeration or FALSE to stop it.

Remarks

The application must implement this function in order to use the **IDirectPlay3::EnumConnections** method. It is called once for each connection that is enumerated.

The application should allocate memory and copy each of the connections for presentation to the user and for use in the **IDirectPlay3::InitializeConnection** method.

See Also

IDirectPlay3::EnumConnections, **IDirectPlay3::InitializeConnection**, *Using DirectPlay Addresses*

EnumDPCallback

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).

```
BOOL FAR PASCAL EnumDPCallback(
    LPGUID lpguidSP,
    LPSTR/LPWSTR lpSPName,
    DWORD dwMajorVersion,
    DWORD dwMinorVersion,
    LPVOID lpContext
    );
```

Parameters

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

lpSPName
Pointer to 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
Pointer to an application-defined context.

Return Values

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

Remarks

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

Application-defined callback function for the **IDirectPlayLobby2::EnumLocalApplications** method.

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

Parameters

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

lpContext
Context passed from the **IDirectPlayLobby2::EnumLocalApplications** call.

dwFlags
Reserved; do not use.

Return Values

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

Remarks

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

Application-defined callback function. The application must implement this function and pass a pointer to it in the **IDirectPlay3::EnumGroups**, **IDirectPlay3::EnumGroupPlayers**, **IDirectPlay3::EnumPlayers**, and **IDirectPlay3::EnumGroupsInGroup** methods. The callback is called once for each player/group that is enumerated.

BOOL FAR PASCAL EnumPlayersCallback2(
DPID *dpId*,
DWORD *dwPlayerType*,
LPCDPNAME *lpName*,
DWORD *dwFlags*,
LPVOID *lpContext*
);

Parameters

dpId
The DPID of the player or group being enumerated.

dwPlayerType
Type of entity, either DPPLAYERTYPE_GROUP or DPPLAYERTYPE_PLAYER.

lpName
Read only pointer to a **DPNAME** structure containing the name of the player or group. This pointer is only valid for the duration of the callback function. Any data that is to be saved for future reference must be copied to some application owned memory.

dwFlags
Flags describing the group or player being enumerated.

DPENUMGROUPS_SHORTCUT — the group is a shortcut.

DPENUMGROUPS_STAGINGAREA — the group is a staging area.

DPENUMPLAYERS_GROUP — both players and groups are being enumerated. This flag is returned only if it was specified in the enumeration method calling this callback.

DPENUMPLAYERS_LOCAL — the player or group exists on a local computer. This flag is returned only if it was specified in the enumeration method calling this callback.

DPENUMPLAYERS_REMOTE — the player or group exists on a remote computer. This flag is returned only if it was specified in the enumeration method calling this callback.

DPENUMPLAYERS_SESSION — the player or group exists in the session identified by *lpguidInstance* in the enumeration method. This flag is returned only if it was specified in the enumeration method calling this callback.

DPENUMPLAYERS_SERVERPLAYER — the player is the server player in an application/server session. Only one server player exists in each session.

DPENUMPLAYERS_SPECTATOR — the player is a spectator (applies to players only).

lpContext
Pointer to an application-defined context.

Return Values

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

Remarks

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.

See Also

DPNAME, **IDirectPlay3::EnumGroups**, **IDirectPlay3::EnumPlayers**, **IDirectPlay3::EnumGroupPlayers**, **IDirectPlay3::EnumGroupsInGroup**

EnumSessionsCallback2

Application-defined callback function for the **IDirectPlay3::EnumSessions** method.

BOOL FAR PASCAL EnumSessionsCallback2(
  LPCDPSESSIONDESC2 *lpThisSD***,**
  LPDWORD *lpdwTimeOut***,**
  DWORD *dwFlags***,**
  LPVOID *lpContext*
  );

Parameters

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

lpdwTimeOut
Pointer to 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
Pointer to an application-defined context.

Return Values

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

Remarks

The application must implement this function in order to use the **IDirectPlay3::EnumSessions** method. This callback function will be called once for each session that is enumerated. Once all the session are enumerated, the callback function will be called one additional time with the DPESC_TIMEDOUT flag.

Applications should look at the flags of the **DPSESSIONDESC2** to determine the nature of the session.

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.

IDirectPlay3

Applications use the methods of the **IDirectPlay3** interface to create DirectPlay objects and work with system-level variables. (The **IDirectPlay3A** interface is the same as the **IDirectPlay3** interface, except that **IDirectPlay3A** uses ANSI characters, and **IDirectPlay3** uses Unicode.) This section is a reference to the methods of this interface.

The methods of the **IDirectPlay3** interface can be organized into the following groups:

Session Management	**Close**
	EnumConnections
	EnumSessions
	GetCaps
	GetGroupConnectionSettings
	GetSessionDesc
	InitializeConnection
	Open
	SetGroupConnectionSettings
	SetSessionDesc
	StartSession
Player management	**CreatePlayer**
	DestroyPlayer
	EnumPlayers
	GetPlayerAddress
	GetPlayerCaps
	GetPlayerData
	GetPlayerName
	SetPlayerData

	SetPlayerName
Message Management	**GetMessageCount**
	Receive
	Send
	SendChatMessage
Group management	**AddGroupToGroup**
	AddPlayerToGroup
	CreateGroup
	CreateGroupInGroup
	DeleteGroupFromGroup
	DeletePlayerFromGroup
	DestroyGroup
	EnumGroupPlayers
	EnumGroups
	EnumGroupsInGroup
	GetGroupData
	GetGroupName
	SetGroupData
	SetGroupName
Initialization	**Initialize**
Security and Authentication	**SecureOpen**

The **IDirectPlay3** interface, like all COM interfaces, inherits the *IUnknown* interface methods. The *IUnknown* interface supports the following three methods:

AddRef
QueryInterface
Release

IDirectPlay3::AddGroupToGroup

Adds a shortcut to a group to an already existing group. This allows the linked group to be enumerated by the **IDirectPlay3::EnumGroupsInGroup** method. To remove a group's shortcut from another group, call **DeleteGroupFromGroup**.

HRESULT AddGroupToGroup(
  DPID *idParentGroup***,**
  DPID *idGroup*
  );

Parameters

idParentGroup
  ID of the group to which the shortcut will be added. Can be any valid group ID.

idGroup
  The group ID of the group whose shortcut will be added. Can be any valid group ID.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDGROUP

Remarks

This method can be used to place a group "inside" more than one group.

A **DPMSG_ADDGROUPTOGROUP** system message is generated to inform players of this change.

See Also

IDirectPlay3::CreateGroupInGroup, **IDirectPlay3::DeleteGroupFromGroup**, **IDirectPlay3::EnumGroupsInGroup**, **DPMSG_ADDGROUPTOGROUP**

IDirectPlay3::AddPlayerToGroup

Add a player to an existing group. A player can be a member of multiple groups. Groups cannot be added to other groups using this API (see **IDirectPlay3::AddGroupToGroup**). An application can add any player to any group (including players and groups that weren't created locally).

HRESULT AddPlayerToGroup(
  DPID *idGroup***,**
  DPID *idPlayer*

);

Parameters

idGroup
Group ID of the group to be augmented.

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

Return Values

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

DPERR_CANTADDPLAYER
DPERR_INVALIDGROUP
DPERR_INVALIDPLAYER
DPERR_NOSESSIONS

This method returns **DPERR_INVALIDPLAYER** if the player DPID is not a player id or if the id is not for a local player.

Remarks

A **DPMSG_ADDPLAYERTOGROUP** system message will be generated and sent to all the other players.

See Also

IDirectPlay3::CreateGroup, **IDirectPlay3::DeletePlayerFromGroup**, **DPMSG_ADDPLAYERTOGROUP**

IDirectPlay3::Close

Closes a previously opened session. Any locally created groups will migrate to be owned by the host of the session.

HRESULT Close( );

Return Values

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

DPERR_INVALIDOBJECT
DPERR_INVALIDPARAMS
DPERR_NOSESSIONS

Remarks

All locally created players will be destroyed and appropriate **DPMSG_DELETEPLAYERFROMGROUP** and **DPMSG_DESTROYPLAYERORGROUP** system messages will be sent to other session participants.

See Also

IDirectPlay3::DestroyPlayer, **DPMSG_DESTROYPLAYERORGROUP**, **IDirectPlay3::Open**

IDirectPlay3::CreateGroup

Creates a group in the current session. A group is a logical collection of players or other groups.

HRESULT CreateGroup(
  LPDPID *lpidGroup***,**
  LPDPNAME *lpGroupName***,**
  LPVOID *lpData***,**
  DWORD *dwDataSize***,**
  DWORD *dwFlags*
  );

Parameters

lpidGroup
Pointer to a variable that will be filled with the DirectPlay group ID. This value is defined by DirectPlay.

lpGroupName
Pointer to a **DPNAME** structure that holds the name of the group. NULL indicates that the group has no initial name. The name in *lpGroupName* is provided for human use only; it is not used internally and need not be unique.

lpData
Pointer to a block of application-defined remote data to associate initially 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 **IDirectPlay3::SetGroupData** were called.

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

dwFlags
Flag indicating what type of group to create. By default (*dwFlags* = 0), ownership of the group will migrate to the host when the owner leaves the session, and the group persists until it is explicitly destroyed.

DPGROUP_STAGINGAREA – the group is created as a staging area. A **staging area** is used to marshal players together in order to launch a new session.

Return Values

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

DPERR_CANTADDPLAYER

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

Remarks

This method will generate a **DPMSG_CREATEPLAYERORGROUP** system message that will be sent to all the other players. The application can use **IDirectPlay3::Send** to send a message to all the players in a group by sending one message to the group ID. DirectPlay will either use multicast to send the message (if the service provider supports it) or send individual messages to each 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.

Groups created with **CreateGroup** are top-level groups in the session. They are enumerated with **IDirectPlay3::EnumGroups**. In contrast, the **IDirectPlay3::CreateGroupInGroup** method creates a group that is a sub-group of a parent group.

Groups can also be used by the application for general organization. In a lobby session, a staging area is used as the mechanism for collecting players for the purpose of starting a new application session using **IDirectPlay3::StartSession**.

The player that creates a group is the default owner of it. Only the owner can change group properties such as the name and remote data. If the owner leaves the session, ownership is transferred to the host of the session.

Any player in the session can change the membership of the group or delete the group.

Groups will persist in the session until they are explicitly destroyed.

See Also

DPNAME, **DPMSG_CREATEPLAYERORGROUP**, **IDirectPlay3::DestroyGroup**, **IDirectPlay3::EnumGroups**, **IDirectPlay3::EnumGroupPlayers**, **IDirectPlay3::Send**, **IDirectPlay3::SetGroupData**, **IDirectPlay3::SetGroupName**, **IDirectPlay3::CreateGroupInGroup**, **IDirectPlay3::GetGroupFlags**

IDirectPlay3::CreateGroupInGroup

Creates a group within an existing group. A group created within another group can only be enumerated using the **IDirectPlay3::EnumGroupsInGroup** method. A group created this way can be destroyed by calling the **IDirectPlay3::DestroyGroup** method.

```
HRESULT CreateGroupInGroup(
    DPID idParentGroup,
    LPDPID lpidGroup,
    LPDPNAME lpGroupName,
    LPVOID lpData,
    DWORD dwDataSize,
```

DWORD *dwFlags*
);

Parameters

idParentGroup
The DPID of the group within which a group will be created. Must be an already existing group.

lpidGroup
Pointer to the DPID that will be filled in with the DirectPlay group ID of the created group.

lpGroupName
Pointer to a **DPNAME** structure that holds the name of the group to be created. NULL indicates that the group has no initial name.

lpData
Pointer to a block of application-defined remote data to associate initially 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 the **IDirectPlay3::SetGroupData** method had been called.

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

dwFlags
Flag indicating what type of group to create. Default (*dwFlags* = 0) is a normal group.

DPGROUP_STAGINGAREA – the group can be used to launch DirectPlay sessions using the **IDirectPlay3::StartSession** method.

Return Values

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

DPERR_CANTADDPLAYER
DPERR_INVALIDFLAGS
DPERR_INVALIDGROUP
DPERR_INVALIDPARAMS

This method returns **DPERR_CANTADDPLAYER** if the group could not be created. It returns **DPERR_INVALIDGROUP** if the parent group ID is invalid.

Remarks

A **DPMSG_CREATEPLAYERORGROUP** system message is generated to inform players of this change.

See Also

IDirectPlay3::DestroyGroup, **IDirectPlay3::EnumGroupsInGroup**, **DPMSG_CREATEPLAYERORGROUP**

IDirectPlay3::CreatePlayer

Creates a local player for the current session.

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

Parameters

lpidPlayer
Pointer to a variable that will be filled with the DirectPlay player ID. This value is defined by DirectPlay.

lpPlayerName
Pointer to a **DPNAME** structure that holds the name of the player. NULL indicates that the player has no initial name information. The name in *lpPlayerName* is provided for human use only. It is not used internally and need not be unique.

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

lpData
Pointer to 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 **IDirectPlay3::SetPlayerData** were called.

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

dwFlags
Flags indicating what type of player this is. Default (*dwFlags* = 0) is a nonspectator, nonserver player.

DPPLAYER_SERVERPLAYER – the player is a server player for client/server communications. Only the host can create a server player. There can only be one server player in a session. **CreatePlayer** will always return a player ID of DPID_SERVERPLAYER if this flag is specified.

DPPLAYER_SPECTATOR – the player is created as a spectator. A spectator player behaves exactly as a normal player except the player is flagged as a spectator. The application can then limit what a spectator player can do. The behavior of a spectator player is completely defined by the application. DirectPlay simply propagates this flag.

Return Values

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

DPERR_CANTADDPLAYER
DPERR_CANTCREATEPLAYER
DPERR_INVALIDFLAGS
DPERR_INVALIDPARAMS
DPERR_NOCONNECTION

Remarks

A single process can have multiple local players that communicate through a DirectPlay object with other players on the same computer or players on remote computers.

Upon successful completion, this method sends a **DPMSG_CREATEPLAYERORGROUP** system message to all the other players in the session announcing that a new player has joined the session. By default, all local players receive copies of all the system messages.

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.

If the application closes the session, any local players created will be automatically destroyed. Only the application that created the player can:

- destroy the player.
- change the player's name or remote data.
- send messages from the player.

If the application uses a separate thread to retrieve DirectPlay messages, it is highly recommended that a non-NULL *hEvent* be supplied and used for synchronization. This event will be set when this player receives a message. Within the message receive thread, use the Win32 API **WaitForSingleObject** (or use **WaitForMultipleObjects** if more than one event is used) within the thread to determine if a player has messages. It is inefficient to loop on **IDirectPlay3::Receive** inside a separate thread waiting for a message. The same event can be used for all the local players, or the application can supply different events for each player. The application is responsible for creating and destroying the event. See *Synchronization* for more information.

See Also

DPNAME, **DPMSG_CREATEPLAYERORGROUP**, **IDirectPlay3::DestroyPlayer**, **IDirectPlay3::EnumPlayers**, **IDirectPlay3::Receive**, **IDirectPlay3::Send**, **IDirectPlay3::SetPlayerData**, **IDirectPlay3::SetPlayerName**, **IDirectPlay3::GetPlayerFlags**

IDirectPlay3::DeleteGroupFromGroup

Removes a shortcut to a group previously added with the **IDirectPlay3::AddGroupToGroup** method from a group. Deleting the shortcut does not destroy the group.

HRESULT DeleteGroupFromGroup(
    DPID *idParentGroup***,**
    DPID *idGroup*
    );

Parameters

idParentGroup
    The group DPID of the group containing the shortcut to be deleted. Can be any valid group DPID.

idGroup
    The group DPID of the group to delete. Can be any valid group DPID.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDGROUP

Remarks

A **DPMSG_DELETEGROUPFROMGROUP** system message is generated to inform players of this change.

See Also

IDirectPlay3::AddGroupToGroup, **DPMSG_DELETEGROUPFROMGROUP**

IDirectPlay3::DeletePlayerFromGroup

Removes a player from a group.

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

Parameters

idGroup
    Group ID of the group to be adjusted.

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

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

DPERR_INVALIDGROUP

DPERR_INVALIDOBJECT

DPERR_INVALIDPLAYER

Remarks 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*.

A player can delete any player from a group, even if that group was created by some other computer.

See Also **IDirectPlay3::AddPlayerToGroup**, **DPMSG_DELETEPLAYERFROMGROUP**

IDirectPlay3::DestroyGroup

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

HRESULT DestroyGroup(
  DPID *idGroup*
  );

Parameters

idGroup
  The ID of the group being removed from the game.

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

DPERR_INVALIDGROUP

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks 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 **DPMSG_DELETEPLAYERFROMGROUP** system message for each player in the group, and then a **DPMSG_DESTROYPLAYERORGROUP** system message. For a list of system messages, see *Using System Messages*.

Any application can destroy any group even if the group was not created locally.

See Also **IDirectPlay3::CreateGroup**, **DPMSG_DESTROYPLAYERORGROUP**

IDirectPlay3::DestroyPlayer

Deletes a local 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.

HRESULT DestroyPlayer(
    DPID *idPlayer*
    );

Parameters

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

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDOBJECT

DPERR_INVALIDPLAYER

Remarks

This method will generate a **DPMSG_DELETEPLAYERFROMGROUP** system message for each group that the player belongs to, and then a **DPMSG_DESTROYPLAYERORGROUP** system message.

Only the application that created the player can destroy it.

See Also

IDirectPlay3::CreatePlayer, **DPMSG_DESTROYPLAYERORGROUP**

IDirectPlay3::EnumConnections

Enumerates all the registered service providers and lobby providers that are available to the application. These should be presented to the user to make a selection. The connection that the user selects should be passed to the **IDirectPlay3::InitializeConnection** method.

HRESULT EnumConnections(
    LPCGUID *lpguidApplication*,
    LPDPENUMCONNECTIONSCALLBACK *lpEnumCallback*,
    LPVOID *lpContext*,
    DWORD *dwFlags*
    );

Parameters

lpguidApplication
Pointer to an application GUID. Only service providers and lobby providers that are usable by this application will be returned. If set to a NULL pointer, a list of all the connections is enumerated regardless of the application GUID.

lpEnumCallback
Pointer to a user-supplied **EnumConnectionsCallback** function that will be called for each available connection.

lpContext
Pointer to a user-defined context that is passed to the callback function.

dwFlags
Flags that specify the type of connections to be enumerated. The default (*dwFlags* = 0) will enumerate DirectPlay service providers only. Possible values are:

DPCONNECTION_DIRECTPLAY – enumerate DirectPlay service providers to communicate in an application session.

DPCONNECTION_DIRECTPLAYLOBBY – enumerate DirectPlay lobby providers to communicate with a lobby server.

Return Values

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

DPERR_INVALIDFLAGS
DPERR_INVALIDPARAMS

Remarks

This method replaces the **DirectPlayEnumerate** function. **DirectPlayEnumerate** still works, but only returns registered service providers.

The order in which the service and lobby providers are returned is not guaranteed to be the same in subsequent calls to **EnumConnections**

Not all the enumerated connections are available for use. For example, this method will return the Modem service provider even if the user has no modem installed. The application can call the **IDirectPlay3::InitializeConnection** method on each connection and check for an error code to determine if the service provider can be used.

See Also

IDirectPlay3::InitializeConnection, **EnumConnectionsCallback**, *DirectPlay Address (Optional)*

IDirectPlay3::EnumGroupPlayers

Enumerates the players belonging to a specific group in the currently open session. If there is no open session, players in a remote session can be enumerated by specifying the DPENUMPLAYERS_SESSION flag and the *guidInstance* of the session. Password protected remote sessions cannot be enumerated.

A pointer to an application-implemented callback function must be supplied, and DirectPlay calls it once for each player in the group that matches the criteria specified in *dwFlags*.

You can't use **EnumGroupPlayers** in a lobby session you're not connected to.

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

Parameters

idGroup
DPID of the group whose players are to be enumerated.

lpguidInstance
Pointer to a GUID identifying the session to be enumerated. This parameter is ignored unless the DPENUMPLAYERS_SESSION flag is specified. The GUID must be equal to one of the sessions enumerated by **IDirectPlay3::EnumSessions**.

lpEnumPlayersCallback2
Pointer to the **EnumPlayersCallback2** function that will be called for every player in the group that matches the criteria specified in *dwFlags*.

lpContext
Pointer to an application-defined context that is passed to each enumeration callback.

dwFlags
Flags specifying how the enumeration is to be done. The default (*dwFlags* = 0) enumerates players in a group in the current active session. You should OR together the flags that you want to specify. Only players that meet all the criteria of the combined flags will be enumerated. For example, if you specify (DPENUMPLAYERS_LOCAL | DPENUMPLAYERS_SPECTATOR) only players in the group that are local and are spectator players will be enumerated. If you specify (DPENUMPLAYERS_LOCAL | DPENUMPLAYERS_REMOTE), no players will be enumerated because no player can be both local and remote.

Can be one or more of the following values:

DPENUMPLAYERS_LOCAL
Enumerates players created locally by this DirectPlay object.

DPENUMPLAYERS_REMOTE
Enumerates players created by remote DirectPlay objects.

DPENUMPLAYERS_SERVERPLAYER
Enumerates the server player.

DPENUMPLAYERS_SESSION

Enumerates the players for the session identified by *lpguidInstance*. This flag can only be used if there is no current open session. This flag can't be used in a lobby session.

DPENUMPLAYERS_SPECTATOR

Enumerates players who are spectator players.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDPARAMS

DPERR_INVALIDGROUP

DPERR_NOSESSIONS

DPERR_UNAVAILABLE

This method returns **DPERR_ACCESSDENIED** if the session is a lobby session you're not connected to. It returns **DPERR_INVALIDPARAMS** if an invalid callback, an invalid *guidInstance*, or invalid flags were supplied. It returns **DPERR_NOSESSIONS** if there is no active session. It returns **DPERR_UNAVAILABLE** if the session could not be enumerated.

Remarks

By default, this method will enumerate all players in 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. Use of the DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay3::EnumSessions** method has been called, and before any calls to the **IDirectPlay3::Close** or **IDirectPlay3::Open** methods.

See Also

IDirectPlay3::CreatePlayer, **IDirectPlay3::DestroyPlayer**, **IDirectPlay3::AddPlayerToGroup**, **IDirectPlay3::DeletePlayerFromGroup**

IDirectPlay3::EnumGroups

Enumerates all the top-level groups in the current session. Top-level groups are groups that were created with the **IDirectPlay3::CreateGroup** method. If there is no open session, groups in a remote session can be enumerated by specifying the DPENUMGROUPS_SESSION flag and the *guidInstance* of the session. Password protected remote sessions cannot be enumerated.

A pointer to an application-implemented callback function must be supplied, and DirectPlay calls it once for each group in the session that matches the criteria specified in *dwFlags*.

You can't use **EnumGroups** in a lobby session you're not connected to.

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

Parameters

lpguidInstance
This parameter should be NULL to enumerate groups in the currently open session. If there is no open session, this parameter can be a pointer to a GUID identifying the session to be enumerated. The GUID must be equal to one of the sessions enumerated by **IDirectPlay3::EnumSessions**. This parameter is ignored unless the DPENUMGROUPS_SESSION flag is specified.

lpEnumPlayersCallback2
Pointer to the **EnumPlayersCallback2** function that will be called for every group in the session that matches the criteria specified in *dwFlags*.

lpContext
Pointer to an application-defined context that is passed to each enumeration callback.

dwFlags
Flags specifying how the enumeration is to be done. Default (dwFlags = 0) enumerates all groups in the current active session. You should OR together the flags that you want to specify. Only groups that meet all the criteria of the combined flags will be enumerated. For example, if you specify (DPENUMGROUPS_LOCAL | DPENUMGROUPS_STAGINGAREA) only groups that are local and are staging areas will be enumerated. If you specify (DPENUMGROUPS_LOCAL | DPENUMGROUPS_REMOTE), no groups will be enumerated because no group can be both local and remote.

Can be one or more of the following values:

DPENUMGROUPS_ALL
Enumerates all groups in this session.

DPENUMGROUPS_LOCAL
Enumerates groups created locally by this DirectPlay object.

DPENUMGROUPS_REMOTE
Enumerates groups created by remote DirectPlay objects.

DPENUMGROUPS_SESSION
Performs enumeration in the session identified by the *lpguidInstance* parameter. This flag can only be used if there is no current active session. You can't use this flag in a lobby session.

DPENUMGROUPS_STAGINGAREA

Enumerates groups that are staging areas. (Staging areas are used to marshal players together in order to launch a new session.)

Return Values

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

DPERR_INVALIDPARAMS

DPERR_NOSESSIONS

DPERR_UNAVAILABLE

This method returns **DPERR_INVALIDPARAMS** if an invalid callback, an invalid *guidInstance*, or invalid flags were supplied. It returns **DPERR_NOSESSIONS** if there is no open session. It returns **DPERR_UNAVAILABLE** if the remote session could not be enumerated.

Remarks

To enumerate the subgroups in a group, use **IDirectPlay3::EnumGroupsInGroup**.

See Also

EnumPlayersCallback2, **IDirectPlay3::CreateGroup**, **IDirectPlay3::DestroyGroup**, **IDirectPlay3::EnumSessions**

IDirectPlay3::EnumGroupsInGroup

Enumerates all groups and shortcuts to groups that are contained within another group. Groups are placed inside other groups by creating them with the **IDirectPlay3::CreateGroupInGroup** method or by adding them to a group with the **IDirectPlay3::AddGroupToGroup** method. This method is not recursive.

You can't use **EnumGroupsInGroup** in a lobby session you're not connected to.

```
HRESULT EnumGroupsInGroup(
    DPID idGroup,
    LPGUID lpguidInstance,
    LPDPENUMPLAYERSCALLBACK2 lpEnumCallback,
    LPVOID lpContext,
    DWORD dwFlags
    );
```

Parameters

idGroup
DPID of the group whose subgroups are to be enumerated.

lpguidInstance
Pointer to a GUID identifying the session to be enumerated. This parameter is ignored unless the **DPENUMPLAYERS_SESSION** flag is specified. The GUID

must equal one of the sessions enumerated by the **IDirectPlay3::EnumSessions** method.

lpEnumCallback
Pointer to the **EnumPlayersCallback2** function that will be called for every group in the group that matches the criteria specified in *dwFlags*.

lpContext
Pointer to an application-defined context that is passed to each enumeration callback.

dwFlags
Flags specifying how the enumeration will be done. The default (*dwFlags* = 0) enumerates all groups in the current active session. You should OR together the flags that you want to specify. Only groups that meet all the criteria of the combined flags will be enumerated. For example, if you specify (DPENUMGROUPS_LOCAL | DPENUMGROUPS_STAGINGAREA) only groups that are local and are staging areas will be enumerated. If you specify (DPENUMGROUPS_LOCAL | DPENUMGROUPS_REMOTE), no groups will be enumerated because no group can be both local and remote.

Can be one or more of the following values:

DPENUMGROUPS_ALL
Enumerates all groups in the group.

DPENUMGROUPS_LOCAL
Enumerates groups in the group created locally by this DirectPlay object.

DPENUMGROUPS_REMOTE
Enumerates groups in the group created by remote DirectPlay objects.

DPENUMGROUPS_SESSION
Performs enumeration in the session identified by the *lpguidInstance* parameter. This flag can only be used if there is no current active session. This flag can't be used in lobby sessions.

DPENUMGROUPS_SHORTCUT
Enumerates groups that are shortcuts added to the group using **IDirectPlay3::AddGroupToGroup**. (A shortcut is a link to another group.)

DPENUMGROUPS_STAGINGAREA
Enumerates groups in the group that are staging areas. (Staging areas are used to marshal players together in order to launch a new session.)

Return Values

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

DPERR_INVALIDFLAGS

DPERR_INVALIDGROUP

DPERR_INVALIDPARAMS

DPERR_NOSESSIONS

DPERR_UNSUPPORTED

This method returns **DPERR_INVALIDPARAMS** if an invalid callback or an invalid *guidInstance* is supplied. It returns **DPERR_NOSESSIONS** if there is no active session. It returns **DPERR_UNSUPPORTED** if the session could not be enumerated.

See Also **IDirectPlay3::CreateGroupInGroup**, **IDirectPlay3::DestroyGroup**, **IDirectPlay3::AddGroupToGroup**, **IDirectPlay3::DeleteGroupFromGroup**

IDirectPlay3::EnumPlayers

Enumerates the players in the current open session. If there is no open session, players in a remote session can be enumerated by specifying the DPENUMPLAYERS_SESSION flag and the *guidInstance* of the session. Password protected remote sessions cannot be enumerated.

A pointer to an application-implemented callback function must be supplied and DirectPlay calls it once for each player in the session that matches the criteria specified in *dwFlags*.

Within a lobby session, this method will always return DPERR_ACCESSDENIED.

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

Parameters

lpguidInstance
: This parameter should be NULL to enumerate players in the currently open session. If there is no open session, this parameter can be a pointer to a GUID identifying the session to be enumerated. The GUID must be equal to one of the sessions enumerated by **IDirectPlay3::EnumSessions**. This parameter is ignored unless the DPENUMPLAYERS_SESSION flag is specified.

lpEnumPlayersCallback2
: Pointer to the **EnumPlayersCallback2** function that will be called for every player in the session that matches the criteria specified in *dwFlags*.

lpContext
: Pointer to an application-defined context that is passed to each enumeration callback.

dwFlags

Flags specifying how the enumeration is to be done. Default (*dwFlags* = 0) enumerates all players in the current active session. You should OR together the flags that you want to specify. Only players that meet all the criteria of the combined flags will be enumerated. For example, if you specify (DPENUMPLAYERS_LOCAL | DPENUMPLAYERS_SPECTATOR) only players that are local and are spectator players will be enumerated. If you specify (DPENUMPLAYERS_LOCAL | DPENUMPLAYERS_REMOTE), no players will be enumerated because no player can be both local and remote.

Can be one or more of the following values:

DPENUMPLAYERS_ALL

Enumerates all players in this session.

DPENUMPLAYERS_GROUP

Includes groups in the enumeration of players.

DPENUMPLAYERS_LOCAL

Enumerates players created locally by this DirectPlay object.

DPENUMPLAYERS_REMOTE

Enumerates players created by remote DirectPlay objects.

DPENUMPLAYERS_SERVERPLAYER

Enumerates the server player.

DPENUMPLAYERS_SESSION

Enumerates the players for the session identified by *lpguidInstance*. This flag can only be used if there is no current open session.

DPENUMPLAYERS_SPECTATOR

Enumerates players who are spectator players.

Return Values

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

DPERR_INVALIDPARAMS

DPERR_NOSESSIONS

DPERR_UNAVAILABLE

This method returns **DPERR_INVALIDPARAMS** if an invalid callback, an invalid *guidInstance*, or invalid flags were supplied. It returns **DPERR_NOSESSIONS** if there is no open session. It returns **DPERR_UNAVAILABLE** if the remote session could not be enumerated.

Remarks

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 **IDirectPlay3::EnumSessions** enumeration. Furthermore, use of the

DPENUMPLAYERS_SESSION flag with this method must occur after the **IDirectPlay3::EnumSessions** method has been called, and before any calls to the **IDirectPlay3::Close** or **IDirectPlay3::Open** methods.

See Also **IDirectPlay3::CreatePlayer**, **IDirectPlay3::DestroyPlayer**, **IDirectPlay3::EnumSessions**

IDirectPlay3::EnumSessions

Enumerates all the active sessions for a particular application and/or all the active lobby sessions that serve a particular application. This method can be called after a DirectPlay object has been created and initialized either by **DirectPlayCreate** or **IDirectPlay3::InitializeConnection**.

This method returns an error if called while a session is already open.

HRESULT EnumSessions(
  LPDPSESSIONDESC2 *lpsd***,**
  DWORD *dwTimeout***,**
  LPDPENUMSESSIONSCALLBACK2 *lpEnumSessionsCallback2***,**
  LPVOID *lpContext***,**
  DWORD *dwFlags*
  );

Parameters

lpsd
: Pointer to the **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 can be set to the globally unique identifier (GUID) of an application of interest if it is known, or to GUID_NULL to obtain all sessions. The *lpszPassword* member is only needed if you want private sessions. All data members besides *guidApplication* and *lpszPassword* are ignored.

dwTimeout
: In the synchronous case, the total amount of time, in milliseconds, that DirectPlay will wait for replies to the enumeration request (not the time between each enumeration). Any replies received after this time-out will be ignored. The application will be blocked until the time-out expires.

  In the asynchronous case, this is the interval, in milliseconds, that enumeration requests will be broadcast by DirectPlay in order to update the internal sessions list.

  If the time-out is set to zero, a default time-out appropriate for the service provider and connection type will be used. It is recommended that you set this value to zero. The application can determine this time-out by calling

IDirectPlay3::GetCaps and examining the *dwTimeout* data member of the **DPCAPS** structure.

lpEnumSessionsCallback2
Pointer to the application-supplied **EnumSessionsCallback2** function to be called for each DirectPlay session responding.

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

dwFlags
The default is 0, which is equivalent to DPENUMSESSIONS_AVAILABLE. When enumerating sessions with DPENUMSESSIONS_ALL or DPENUMSESSIONS_PASSWORDREQUIRED it is important for the application to know which sessions cannot be joined and which sessions are password-protected so the user can be warned or prompted for a password.

DPENUMSESSIONS_ALL

Enumerate all active sessions, whether they are accepting new players or not. Sessions in which the player limit has been reached, new players have been disabled, or joining has been disabled will be enumerated.

Password protected sessions will not be enumerated unless the DPENUMSESSIONS_PASSWORDREQUIRED flag is also specified.

If DPENUMSESSIONS_ALL is not specified, DPENUMSESSIONS_AVAILABLE is assumed.

DPENUMSESSIONS_ASYNC

Enumerates all the current sessions in the session cache and returns immediately. Starts the asynchronous enumeration process if not already started. Updates to the session list continue until canceled by calling **EnumSessions** with the **DPENUMSESSIONS_STOPASYNC** flag, or by calling **Open**, or by calling **Release**.

If this flag is not specified, the enumeration is done synchronously.

DPENUMSESSIONS_AVAILABLE

Enumerate all sessions that are accepting new players to join. Sessions which have reached their maximum number of players, or have disabled new players, or have disabled joining will not be enumerated.

Password protected sessions will not be enumerated unless the **DPENUMSESSIONS_PASSWORDREQUIRED** flag is also specified.

DPENUMSESSIONS_PASSWORDREQUIRED

When used in combination with one of the two flags **DPENUMSESSIONS_AVAILABLE** or **DPENUMSESSIONS_ALL**, enables password-protected sessions to be enumerated in addition to sessions without password protection.

If this flag is not specified, no password protected sessions will be returned.

DPENUMSESSIONS_RETURNSTATUS

If this flag is specified, the enumeration will not display any dialog boxes showing the connection progress status. If the connection cannot be made immediately, the method will return with the **DPERR_CONNECTING** error. The application must keep calling **EnumSessions** until either DP_OK is returned, indicating successful completion, or some other error code is returned, indicating an error.

DPENUMSESSIONS_STOPASYNC

Enumerates all the current sessions in the session cache and cancels the asynchronous enumeration process.

Return Values

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

DPERR_CONNECTING

DPERR_EXCEPTION

DPERR_GENERIC

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_UNINITIALIZED

DPERR_USERCANCEL

This method returns **DPERR_GENERIC** if the session is already open. It returns **DPERR_UNINITIALIZED** if the DirectPlay object has not been initialized. It returns **DPERR_USERCANCEL** if the user canceled the enumeration process (usually by canceling a service provider dialog box). It returns **DPERR_CONNECTING** if the method is in the process of connecting to the network.

Remarks

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.

EnumSessions can be called synchronously (default) or asynchronously. When called synchronously, DirectPlay will clear the internal session cache, send out an enumeration request, and collect replies until the specified time-out expires. Each session will then be returned to the application through the callback function. The application will be blocked until all the sessions have been returned through the callback function.

When called asynchronously (**DPENUMSESSIONS_ASYNC**), all the current sessions (if any) in the session cache will be returned to the application through the callback function and then the method will return. DirectPlay will then automatically send out enumeration requests with the period of the time-out parameter and listen for replies. Each reply will be used to update the session cache:

- Sessions already in the cache will be updated

- Sessions that haven't been updated for a set period of time (and thus have expired) will be deleted
- New sessions will be added

The application must call **EnumSessions** (with the **DPENUMSESSIONS_ASYNC** flag) again to obtain the most up-to-date session list with all the expired sessions deleted, new sessions added, and updated sessions. Subsequent calls to **EnumSessions** will not generate an enumeration request. Enumeration requests will be generated periodically by DirectPlay until the process is either canceled by calling **EnumSessions** with the **DPENUMSESSIONS_STOPASYNC** flag, a session is opened using the **IDirectPlay3::Open** method, or the DirectPlay object is released.

Once enumeration of the available sessions is complete, the application can join one of the sessions using the **IDirectPlay3::Open** method. Only sessions in the session cache can be opened. It is possible for the application to attempt to open a session that has expired since the last time **EnumSessions** was called in which case an error will be returned.

No authentication is performed when enumerating sessions on a secure server. All sessions that meet the enumeration criteria will be returned. Authentication will be done when the application attempts to open one of these sessions with **IDirectPlay3::Open** or **IDirectPlay3::SecureOpen**.

If the application was not started by a lobby, the service provider can display a dialog box asking the user for information (such as a phone number or IP address) in order to perform the enumeration on the network, if that information was not provided in **IDirectPlay3::InitializeConnection**. If the service provider can do a broadcast enumeration and requires no extra information from the user, no dialog box will appear. If the user cancels a service provider dialog box, **EnumSessions** returns **DPERR_USERCANCEL**.

See Also **DPSESSIONDESC2**, **IDirectPlay3::Open**, **IDirectPlay3::SecureOpen**

IDirectPlay3::GetCaps

Obtains the capabilities of this DirectPlay object.

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

Parameters

lpDPCaps
Pointer to 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 **IDirectPlay3::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.

Return Values

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

DPERR_INVALIDOBJECT
DPERR_INVALIDPARAMS

Remarks

This method returns the capabilities of the current session, while the **IDirectPlay3::GetPlayerCaps** method returns the capabilities of the requested player.

See Also

DPCAPS, **IDirectPlay3::GetPlayerCaps**, **IDirectPlay3::Send**

IDirectPlay3::GetGroupConnectionSettings

Retrieves the connection settings for a group from the **DPLCONNECTION** structure. Any sessions launched from this group will use these settings. This method can only be used in a lobby session.

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

Parameters

dwFlags
Not used. Must be zero.

idGroup
The DPID of the group to get the connection settings for.

lpData
Pointer to a buffer into which the **DPLCONNECTION** structure and all its data will be copied. 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
: Pointer to 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.

Return Values

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

DPERR_ACCESSDENIED

DPERR_BUFFERTOOSMALL

DPERR_INVALIDGROUP

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

DPERR_UNSUPPORTED

Remarks

Group connection settings are only relevant for **staging area** groups.

You can see if a game has been launched by looking at the *guidInstance* data member of the **DPSESSIONDESC2** structure, whose pointer is contained in the *lpSessionDesc* data member of the **DPLCONNECTION** structure returned by this method. The *guidInstance* will be GUID_NULL until a game has been launched.

See Also

DPLCONNECTION, **IDirectPlayLobby2::RunApplication**, **IDirectPlay3::SetGroupConnectionSettings**, **DPGROUP_STAGINGAREA**

IDirectPlay3::GetGroupData

Retrieves an application-specific data block that was associated with a group ID by using **IDirectPlay3::SetGroupData**.

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

Parameters

idGroup
: Group ID for which data is being requested.

lpData
Pointer to 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
Pointer to 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_LOCAL
Retrieves the local data set by this application

DPGET_REMOTE
Retrieves the current data from the remote shared data space.

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

DPERR_BUFFERTOOSMALL
DPERR_INVALIDGROUP
DPERR_INVALIDOBJECT
DPERR_INVALIDPARAMS
DPERR_INVALIDPLAYER

Remarks
DirectPlay can maintain two types of group 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
IDirectPlay3::SetGroupData

IDirectPlay3::GetGroupFlags

Returns the flags describing the group.

HRESULT GetGroupFlags(
DPID *idGroup***,**
LPDWORD *lpdwFlags*
);

Parameters

idGroup
The DPID of the group whose flag settings are to be retrieved.

lpdwFlags
Pointer to a DWORD to be set to the group flag settings. Can be one or more of the following:

DPGROUP_LOCAL – the group was created by this application. If this flag is not specified, the group is a remote group.

DPGROUP_STAGINGAREA – the group is a staging area.

Return Values

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

DPERR_INVALIDGROUP

DPERR_INVALIDPARAMS

See Also

IDirectPlay3::CreateGroup

IDirectPlay3::GetGroupName

Returns the name associated with a group.

HRESULT GetGroupName(
  DPID *idGroup***,**
  LPVOID *lpData***,**
  LPDWORD *lpdwDataSize*
  );

Parameters

idGroup
ID of the group whose name is being requested.

lpData
Pointer to 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
Pointer to 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.

Return Values

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDGROUP

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

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

See Also **DPNAME**, **IDirectPlay3::SetGroupName**

IDirectPlay3::GetGroupParent

Returns the DPID of the parent of the group.

HRESULT GetGroupParent(
  DPID *idGroup***,**
  LPDPID *lpidParent*
  );

Parameters

idGroup
  ID of the group whose parent is being requested.

lpidParent
  Pointer to a DPID to be set to the ID of the parent group. If this method returns a parent ID of zero, the group is a root group.

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

DPERR_INVALIDGROUP

DPERR_INVALIDPARAMS

See Also **IDirectPlay3::CreateGroupInGroup**

IDirectPlay3::GetMessageCount

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

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

Parameters

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

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

Return Values

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

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

See Also

IDirectPlay3::Receive

IDirectPlay3::GetPlayerAccount

In a secure session, can be called by the session host to obtain account information about the specified player.

HRESULT GetPlayerAccount(
  DPID *idPlayer***,**
  DWORD *dwFlags***,**
  LPVOID *lpData***,**
  LPDWORD *lpdwDataSize*
  );

Parameters

idPlayer
The DPID of the player whose account information is to be retrieved.

dwFlags
Not used. Must be zero.

lpData
A pointer to a buffer (allocated by the application) where the account data is to be written. A **DPACCOUNTDESC** structure will be copied as well as any data referenced by the members of the structure; therefore, the number of bytes copied is variable. Cast the *lpData* parameter to LPDPACCOUNTDESC in order to read the name. By passing NULL, the application can request the number of bytes required be put in the *lpdwDataSize* parameter. The application can then allocate the space and call this method again.

lpdwDataSize
A pointer to a DWORD that is initialized with the size of the buffer. After the method returns, *lpdwDataSize* will be set to the number of bytes that were actually copied into the buffer. If the buffer was too small the method returns

DPERR_BUFFERTOOSMALL, and *lpdwDataSize* is set to the minimum required buffer size.

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

DPERR_ACCESSDENIED

DPERR_BUFFERTOOSMALL

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks The information returned by **GetPlayerAccount** uniquely identifies an account. This information can be used to record the transactions or other activities of a logged-in player.

See Also **IDirectPlay3::SecureOpen**

IDirectPlay3::GetPlayerAddress

Retrieves the DirectPlay Address for a player.

The DirectPlay Address is a network address for a player using a specific service provider. You should call the **IDirectPlayLobby2::EnumAddress** method to parse the DirectPlay Address buffer retrieved by **IDirectPlay3::GetPlayerAddress**.

HRESULT GetPlayerAddress(
  DPID *idPlayer***,**
  LPVOID *lpData***,**
  LPDWORD *lpdwDataSize*
  );

Parameters

idPlayer
  Player ID that the address is being requested for. Pass in zero to obtain a list of valid address options for a service provider (for example, a list of valid modem choices for the modem-to-modem service provider).

lpData
  Pointer to a buffer where the DirectPlay Address 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
  Pointer to 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, then this parameter will be set to the buffer size that is required and the method will return **DPERR_BUFFERTOOSMALL**.

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks To get a list of valid modem choices, pass in zero for the *idPlayer* parameter. A list of modem choices will be returned as a list of ANSI or Unicode strings with a zero-length string at the end.

For more information about the DirectPlay Address, see *DirectPlay Address (Optional)*.

See Also **IDirectPlayLobby2::EnumAddress**

IDirectPlay3::GetPlayerCaps

Retrieves the current capabilities of a specified player.

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

Parameters

idPlayer
Player ID for which the capabilities should be computed.

lpPlayerCaps
Pointer to a **DPCAPS** structure that will be filled with the capabilities. The *dwSize* member of the **DPCAPS** structure must be filled in before using **IDirectPlay3::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.

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

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks This method returns the capabilities of the requested player, while the **IDirectPlay3::GetCaps** method returns the capabilities of the current session.

See Also **DPCAPS**, **IDirectPlay3::GetCaps**, **IDirectPlay3::Send**

IDirectPlay3::GetPlayerData

Retrieves an application-specific data block that was associated with a player ID by using **IDirectPlay3::SetPlayerData**.

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

Parameters

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

lpData
Pointer to 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
Pointer to 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_LOCAL
Retrieves the local data set by this application.

DPGET_REMOTE
Retrieves the current data from the remote shared data space.

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDFLAGS

DPERR_INVALIDOBJECT

DPERR_INVALIDPLAYER

Remarks

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

IDirectPlay3::SetPlayerData

IDirectPlay3::GetPlayerFlags

Returns the flags describing the player.

HRESULT GetPlayerFlags(
    DPID *idPlayer*,
    LPDWORD *lpdwFlags*
    );

Parameters

idPlayer
The DPID of the player whose flag settings are to be retrieved.

lpdwFlags
Pointer to a DWORD to be set to the player's flag settings. Can be one or more of the following:

DPPLAYER_LOCAL – the player was created by this application. If this flag is not specified, the player is a remote player.

DPPLAYER_SERVERPLAYER – the player is a server player for client/server communications.

DPPLAYER_SPECTATOR – the player was created as a spectator.

Return Values

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

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

See Also

IDirectPlay3::CreatePlayer

IDirectPlay3::GetPlayerName

Retrieves the name associated with a player.

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

Parameters

idPlayer
 ID of the player whose name is requested.

lpData
 Pointer to 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
 Pointer to 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.

Return Values

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDOBJECT

DPERR_INVALIDPLAYER

Remarks

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

See Also

DPNAME, **IDirectPlay3::SetPlayerName**

IDirectPlay3::GetSessionDesc

Retrieves the properties of the current open session.

HRESULT GetSessionDesc(
 LPVOID *lpData***,**
 LPDWORD *lpdwDataSize*

);

Parameters

lpData
Pointer to 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
Pointer to 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. If this parameter is NULL, the method returns DPERR_INVALIDPARAM.

Return Values

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_NOCONNECTION

Remarks

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

See Also

DPSESSIONDESC2, **IDirectPlay3::EnumSessions**, **IDirectPlay3::Open**

IDirectPlay3::Initialize

This method is provided for compliance with the COM protocol.

HRESULT Initialize(
LPGUID *lpGUID*
);

Parameters

lpGUID
Pointer to the globally unique identifier (GUID) used as the interface identifier.

Return Values

Returns **DPERR_ALREADYINITIALIZED**.

Remarks

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**

IDirectPlay3::InitializeConnection

Initializes a DirectPlay connection. All the information about the connection, including the service provider to use, the network address of the server, and the GUID of the session, is passed in through the *lpConnection* parameter.

HRESULT InitializeConnection(
  LPVOID *lpConnection*,
  DWORD *dwFlags*
  );

Parameters

lpConnection
  Pointer to a buffer that contains all the information about the connection to be initialized as a *DirectPlay Address*.

dwFlags
  Not used. Must be zero.

Return Values

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

DPERR_ALREADYINITIALIZED

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

DPERR_UNAVAILABLE

This method returns **DPERR_ALREADYINITIALIZED** if **InitializeConnection** has previously been called on this object. It returns **DPERR_UNAVAILABLE** if the service provider could not be initialized. This can be because the resources necessary to operate this service provider are not present (for example, TCP/IP stack not present).

Remarks

The *lpConnection* parameter supplied to this method can be obtained from the **IDirectPlay3::EnumConnections** method, or the application can create one directly using the **IDirectPlayLobby2::CreateCompoundAddress** method.

This method loads and initializes the appropriate service provider. The *lpConnection* parameter is passed on to the service provider, which extracts and saves any relevant information. This information is used when the **IDirectPlay3::EnumSessions** or **IDirectPlay3::Open** method is called so that the service provider does not pop up a dialog box asking for that information.

This method replaces **DirectPlayCreate** as the means of binding a service provider to a DirectPlay object. The primary benefits of **InitializeConnection** are that you can override the service provider dialogs, and you can initialize a lobby provider.

See Also **IDirectPlay3::EnumConnections**, **IDirectPlayLobby2::CreateCompoundAddress**, *DirectPlay Address (Optional)*

IDirectPlay3::Open

Used to join a session that has been enumerated by a previous call to **IDirectPlay3::EnumSessions** or to create a new session that other users can enumerate and join.

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

Parameters

lpsd
: Pointer to the **DPSESSIONDESC2** structure describing the session to be created or joined. If a session is being joined, then only the *dwSize*, *guidInstance*, and *lpszPassword* data members need to be specified. A password need only be supplied if the enumerated session had the DPSESSION_PASSWORD flag set.

  If a session is being created, then the application must completely fill out the **DPSESSIONDESC2** structure with the properties of the sessions to be created. The *guidInstance* will be generated by DirectPlay.

dwFlags
: One and only one of the following flags:

  DPOPEN_CREATE
  Create a new instance of an application session. The local computer will be the name server and host of the session.

  DPOPEN_JOIN
  Join an existing instance of an application session for the purpose of participating. The application will be able to create players and send and receive messages.

  DPOPEN_RETURNSTATUS
  If this flag is specified, the method will not display any dialog boxes showing the connection progress status. If the connection cannot be made immediately, the method will return with the **DPERR_CONNECTING** error. The application must keep calling **Open** until either DP_OK is returned, indicating successful completion, or some other error code is returned, indicating an error.

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

DPERR_ACCESSDENIED

DPERR_ALREADYINITIALIZED
DPERR_AUTHENTICATIONFAILED
DPERR_CANTLOADCAPI
DPERR_CANTLOADSECURITYPACKAGE
DPERR_CANTLOADSSPI
DPERR_CONNECTING
DPERR_ENCRYPTIONFAILED
DPERR_INVALIDCREDENTIALS
DPERR_INVALIDFLAGS
DPERR_INVALIDPARAMS
DPERR_INVALIDPASSWORD
DPERR_LOGONDENIED
DPERR_NOCONNECTION
DPERR_NONEWPLAYERS
DPERR_SIGNFAILED
DPERR_TIMEOUT
DPERR_UNINITIALIZED
DPERR_USERCANCEL

This method returns **DPERR_ALREADYINITIALIZED** if there is already an open session on this DirectPlay object. It returns **DPERR_TIMEOUT** if the session did not respond to the **Open** request. It returns **DPERR_USERCANCEL** if the user canceled the enumeration process(usually by canceling a service provider dialog box).

Remarks

Once an application has joined a session, it can create a player and start communicating with other players in the session. The application will not receive any messages nor can it send any messages on this DirectPlay object until it creates a local player using **IDirectPlay3::CreatePlayer**.

In order to have two sessions open simultaneously, the application must create two DirectPlay objects and open a session on each one.

To join a session, it is only necessary to fill in the *dwSize* and *guidInstance* members of the **DPSESSIONDESC2** structure. The *lpszPassword* member must also be filled in if the session was marked as password protected. An enumerated session will have the **DPSESSION_PASSWORDREQUIRED** flag set if it requires a password.

If joining a secure session, you must use **SecureOpen** to provide login credentials. The enumerated session will have the **DPSESSION_SECURESERVER** flag set if it requires credentials.

If you specify the DPSESSION_SECURESERVER flag in the **DPSESSIONDESC2** structure, the session will be opened with the default security package, NTLM (NT LAN Manager). To specify an alternate security package, use **SecureOpen**.

When an application attempts to join a session, the server can reject the **Open** request or ignore it (in which case **Open** will time-out). Attempting to join a session where new players are disabled, joining is disabled, the player limit has been reached, or an incorrect password is supplied will result in a **DPERR_NONEWPLAYERS** or **DPERR_INVALIDPASSWORD** error.

See Also

DPSESSIONDESC2, **IDirectPlay3::Close**, **IDirectPlay3::SecureOpen**, **IDirectPlay3::EnumSessions**

IDirectPlay3::Receive

Retrieves a message from the message queue.

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

Parameters

lpidFrom
  Pointer to a DPID 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
  Pointer to a DPID 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. By default (*dwFlags* = 0), the first available message will be retrieved.

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

  DPRECEIVE_PEEK
    Returns a message as specified by the other flags, but does not remove it

from the message queue. This flag must be specified if *lpData* is NULL.

DPRECEIVE_TOPLAYER and **DPRECEIVE_FROMPLAYER**

If both DPRECEIVE_TOPLAYER and DPRECEIVE_FROMPLAYER are specified, **Receive** will only return messages that are 1) sent to the player specified by *lpidTo* and 2) sent from the player specified by *lpidFrom*. Note that both conditions must be met.

If only DPRECEIVE_TOPLAYER is specified, **Receive** will only return messages sent to the player specified by *lpidTo*.

If only DPRECEIVE_FROMPLAYER is specified, **Receive** will only return messages sent from the player specified by *lpidFrom*.

If neither DPRECEIVE_TOPLAYER nor DPRECEIVE_FROMPLAYER is set, **Receive** will return the first available message.

lpData

Pointer to 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

Pointer to a DWORD that is initialized to the size of the buffer before calling this method. After the method returns, this parameter will be set to the number of bytes copied into the buffer. 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 **IDirectPlay3::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 **IDirectPlay3::Receive**. It is best to keep reallocating memory until a DPERR_BUFFERTOOSMALL error is not received.

Return Values

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

Remarks

Any message received from a player ID defined as DPID_SYSMSG is a system message used to notify the application of a change in the session. 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 a player ID defined as DPID_ALLPLAYERS or to a Group ID still appear to come from the sending player ID. An application will only receive 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* will always be 0xFFFFFFFF and the *lpidTo* value is arbitrary.

All the service providers shipped with DirectPlay perform integrity checks on the data to protect against corruption. Any message retrieved using **Receive** is guaranteed to be free from corruption.

See Also **DPMSG_GENERIC**, **IDirectPlay3::Send**

IDirectPlay3::SecureOpen

Creates or joins a secure session. When joining a secure session, use this method to supply login credentials.

When creating a new session, the host computer can specify an alternate security package to use. When joining a session, a computer can specify a user name and password.

HRESULT IDirectPlay3::SecureOpen(
  LPCDPSESSIONDESC2 *lpsd*,
  DWORD *dwFlags*,
  LPCDPSECURITYDESC *lpSecurity*,
  LPCDPCREDENTIALS *lpCredentials*
)

Parameters

lpsd
Pointer to the **DPSESSIONDESC2** structure describing the session to be created or joined. If a session is to be joined, then only the **guidInstance** and **lpszPassword** members need to be specified. The password need only be supplied if the enumerated session had the DPSESSION_PASSWORDREQUIRED flag set.

If a session is to be created, then the application must completely fill out the **DPSESSIONDESC2** structure with the properties of the sessions to be created. The *guidInstance* will be generated by DirectPlay. NOTE: The DPSESSION_SECURESERVER flag must be set to indicate that all computers attempting to open the session must be authenticated.

If you don't specify the DPSESSION_SECURESERVER flag when filling out the **DPSESSIONDESC2** structure, **SecureOpen** will open an unsecure session, just as if you had called **Open**.

dwFlags

One of the following flags:

DPOPEN_CREATE

Creates a new instance of a secured session.

DPOPEN_JOIN

Joins an existing instance of a secured session.

DPOPEN_RETURNSTATUS

If this flag is specified, the method will not display any dialog boxes showing the connection progress status. If the connection cannot be made immediately, the method will return with the **DPERR_CONNECTING** error. The application must keep calling **SecureOpen** until either DP_OK is returned, indicating successful completion, or some other error code is returned, indicating an error.

lpSecurity

Pointer to a **DPSECURITYDESC** structure containing the security package to use. Set this parameter to NULL to use the default security package (NT LAN Manager) and CryptoAPI package (Microsoft RSA Base Cryptographic Provider). Relevant only when creating a session. Must be set to NULL when joining a session.

lpCredentials

Pointer to a **DPCREDENTIALS** structure containing the logon name, password, and domain to be authenticated on the server. NULL if there are no credentials. Credentials are ignored when creating a session.

Return Values

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

DPERR_ACCESSDENIED

DPERR_ALREADYINITIALIZED

DPERR_AUTHENTICATIONFAILED

DPERR_CANTLOADCAPI

DPERR_CANTLOADSECURITYPACKAGE

DPERR_CANTLOADSSPI

DPERR_CONNECTING

DPERR_ENCRYPTIONFAILED

DPERR_INVALIDCREDENTIALS

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

DPERR_INVALIDPASSWORD

DPERR_LOGONDENIED
DPERR_NOCONNECTION
DPERR_NONEWPLAYERS
DPERR_SIGNFAILED
DPERR_TIMEOUT
DPERR_UNINITIALIZED
DPERR_USERCANCEL

This method **DPERR_LOGONDENIED** after being called with invalid creadentials or without credentials when credentials are required. The application must collect the user's credentials and call **SecureOpen** again.

Remarks

When joining a session, first call **SecureOpen** with no credentials. If the method returns **DPERR_LOGONDENIED**, then the application must collect the user name and password from the user and call **SecureOpen** again, passing in the user's credentials through the *lpCredentials* parameter. If the method returns DP_OK, then the player was able to login with the credentials he or she had specified earlier during system logon (network logon in NTLM).

See Also

IDirectPlay3::Open

IDirectPlay3::Send

Sends a message to another player, to a group of players, or to all players in the session. To send a message to another player, specify the target player's player ID. To send a message to a group of players, send the message to the group ID assigned to the group. To send a message to the entire session, send the message to the player ID DPID_ALLPLAYERS. Messages can be sent using either a guaranteed or nonguaranteed protocol on a per message basis. If the session is being hosted on a secure server, messages can be sent encrypted (to ensure privacy) or digitally signed (to ensure authenticity) on a per message basis.

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

Parameters

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

idTo
The destination ID of the message. To send a message to another player, specify the ID of the player. To send a message to all the players in a group, specify the ID of the group. To send a message to all the players in the session, use the constant symbol DPID_ALLPLAYERS. To send a message to the server player, specify the constant symbol DPID_SERVERPLAYER. A player cannot send a message to itself.

dwFlags
Indicates how the message should be sent. By default (*dwFlags* = 0), the message is sent nonguaranteed.

DPSEND_ENCRYPTED
Sends the messages encrypted. This can only be done in a secure session. This flag can only be used if the **DPSEND_GUARANTEED** flag is also set. The message will be sent as a **DPMSG_SECUREMESSAGE** system message.

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

DPSEND_SIGNED
Sends the message with a digital signature. This can only be done in a secure session. This flag can only be used if the **DPSEND_GUARANTEED** flag is also set. The message will be sent as a **DPMSG_SECUREMESSAGE** system message.

lpData
Pointer to the data being sent.

dwDataSize
Length of the data being sent.

Return Values

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

DPERR_BUSY

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

DPERR_NOTLOGGEDIN

DPERR_SENDTOOBIG

This method returns DPERR_INVALIDPARAMS if the encrypted or signed flag is specified for a message that is not also specified as guaranteed. It returns DPERR_NOTLOGGEDIN when the client application tries to send a secure message without first logging in.

Remarks

Messages can be sent guaranteed or nonguaranteed. By default, messages are sent nonguaranteed which means that DirectPlay does no verification that the message reached the intended recipient. Sending a guaranteed message takes much longer; a minimum of 2 to 3 times longer than nonguaranteed messages. Applications should try to minimize sending guaranteed messages as much as possible and design the application to tolerate lost messages. All the service providers shipped with DirectPlay perform integrity checks on the data to protect against corruption. Any message retrieved using this method is guaranteed to be free from corruption.

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. The exception to this rule is if the DPSESSION_NOMESSAGEID was specified in the session description (**DPSESSIONDESC2**). Then 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 based on the message ID.

When DPSESSION_NOMESSAGEID is used, the *idFrom* parameter has no meaning and the *idTo* parameter is used simply to direct the message to the correct target computer. If the target computer has more than one player on it, it cannot be determined whose receive queue the message will appear in. When the message is received, it will appear to have come from player DPID_UNKNOWN.

There is no limit to the size of messages that can be transmitted using the **Send** method. DirectPlay will automatically break up large messages into packets (packetize) and reassemble them on the receiving end. Beware of sending large messages nonguaranteed — if even one of the packets fails to reach the receiver then the entire message will be ignored. The application can determine the maximum size of a message before it starts packetizing by calling *GetCaps* and examining the **dwMaxBufferSize** member of the **DPCAPS** structure.

When you send an encryted or signed message, it is not delivered as an application message, but as a system message, **DPMSG_SECUREMESSAGE**.

See Also

IDirectPlay3::Receive, **IDirectPlay3::SendChatMessage**, **DPMSG_SECUREMESSAGE**

IDirectPlay3::SendChatMessage

Sends a text message to another player, a group of players, or all players. This method supports both Unicode (the **IDirectPlay3** interface) and ANSI strings (the *IDirectPlay3A* interface). The player receiving the chat message is informed through a **DPMSG_CHAT** system message in the player's receive queue. This method must be used in a lobby session.

HRESULT SendChatMessage(
    DPID *idFrom***,**
    DPID *idTo***,**
    DWORD *dwFlags***,**
    LPDPCHAT *lpChatMessage*
    );

Parameters

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.

dwFlags
Indicates how the message should be sent. If this parameter is set to 0, the message is sent nonguaranteed.

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

lpChatMessage
Pointer to a **DPCHAT** structure containing the message to be sent.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

This method returns **DPERR_INVALIDPARAMS** if the *idTo* ID is not a valid player or group. It returns **DPERR_INVALIDPLAYER** if the *idFrom* ID is not a valid player. It returns **DPERR_ACCESSDENIED** if the *idFrom* ID is not a local player.

Remarks

This method facilitates player to player chatting within a lobby session where it is possible for different client applications to be connected. You must use this method in a lobby session. Use is optional in an application seession.

The receiving player will receive a system message (*idFrom* = DPID_SYSTEM). The **DPCHAT** structure will specify which player the chat message came from.

See Also **DPCHAT**, **DPMSG_CHAT**, **IDirectPlay3::Send**

IDirectPlay3::SetGroupConnectionSettings

Sets the connection settings for a session that will be launched from this group. This method can only be used in a lobby session.

HRESULT SetGroupConnectionSettings(
    DWORD *dwFlags*,
    DPID *idGroup*,
    LPDPLCONNECTION *lpConnection*
    );

Parameters

dwFlags
    Not used. Must be zero.

idGroup
    The DPID of the group to set the connection settings on.

lpConnection
    Pointer to a **DPLCONNECTION** structure describing the application to be launched, the service provider to use, and the session description of the session to be created.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDGROUP

DPERR_INVALIDPARAMS

DPERR_UNSUPPORTED

Remarks

Call **IDirectPlay3::GetGroupConnectionSettings** before calling **SetGroupConnectionSettings** to see if any of the **DPLCONNECTION** structure members already have default values (non-NULL or non-zero). If so, you may get an error if you try to change these default values.

You do not have to set the *lpAddress* and *dwAddressSize* data members of the **DPLCONNECTION** structure with **SetGroupConnectionSettings**. In the **DPSESSIONDESC2** structure within the **DPLCONNECTION** structure, you do not have to fill in the *guidInstance* member.

See Also **DPLCONNECTION**, **IDirectPlayLobby2::RunApplication**, **IDirectPlay3::GetGroupConnectionSettings**

IDirectPlay3::SetGroupData

Associates an application-specific data block with a group ID. Only the computer that created the group can change the remote data associated with it.

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

Parameters

idGroup
  Group ID for which data is being set.

lpData
  Pointer to the data to be set. Set to NULL to clear any existing group data.

dwDataSize
  Size of the data buffer. If *lpData* is NULL and this parameter does not equal zero, the method returns DPERR_INVALIDPARAMS.

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

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

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

  DPSET_REMOTE
    This data is for use by all the applications, and will be propagated to all the other applications in the session. This flag can only be used on groups owned by the local session.

Return Values

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

DPERR_INVALIDGROUP

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks

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*.

This method should not be used for updating real-time information (such as position updates) due to the overhead introduced. It is much more efficient to use **IDirectPlay3::Send** for this. **IDirectPlay3::SetGroupData** is more appropriate for shared state information that doesn't change very often and is not time critical (such as a team color).

See Also

DPMSG_SETPLAYERORGROUPDATA, **IDirectPlay3::GetGroupData**, **IDirectPlay3::Send**

IDirectPlay3::SetGroupName

Sets the name of a group after it has been created. Only the computer that created the group can set the name of the group. A **DPMSG_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.

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

Parameters

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

lpGroupName
Pointer to 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. This value can only be used on groups owned by the local session.

DPSET_GUARANTEED

Propagates the data using guaranteed messaging (if available).

DPSET_LOCAL

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

DPSET_REMOTE

This data is for use by all the applications, and will be propagated to all the other applications in the session. This flag can only be used on groups owned by the local session.

Return Values

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

DPERR_INVALIDGROUP

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

See Also

DPNAME, **DPMSG_SETPLAYERORGROUPNAME**, **IDirectPlay3::GetGroupName**, **IDirectPlay3::Send**

IDirectPlay3::SetPlayerData

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

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

Parameters

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

lpData
  Pointer to the data to be set. Set this parameter to NULL and *dwDataSize* to zero to clear out any existing player data.

dwDataSize
  Size of the data buffer. If *lpData* is NULL and this parameter does not equal zero, the method returns DPERR_INVALIDPARAMS.

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

DPSET_GUARANTEED

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

DPSET_LOCAL

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

DPSET_REMOTE

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

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDFLAGS

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_INVALIDPLAYER

Remarks

SetPlayerData should not be used for updating real-time information (such as position updates) due to the overhead introduced. It is much more efficient to use **Send** for this. **SetPlayerData** is more appropriate for shared state information that doesn't change very often and is not time critical (such as a player's name).

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. This method returns **DPERR_ACCESSDENIED** if you try to set player data for a remote player. A **DPMSG_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, **IDirectPlay3::GetPlayerData**, **IDirectPlay3::Send**

IDirectPlay3::SetPlayerName

Sets the name of a local player after it has been changed. Only the computer that created the player can change the name. A **DPMSG_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.

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

Parameters

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

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

dwFlags
Flags indicating how the name will be propagated. It can be one of the following values:

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

DPSET_LOCAL
Data is not propagated to other players.

DPSET_REMOTE
Propagates the data to all players in the session using nonguaranteed message passing. This is the default value.

Return Values

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

DPERR_INVALIDOBJECT

DPERR_INVALIDPLAYER

See Also

DPNAME, **DPMSG_SETPLAYERORGROUPNAME**, **IDirectPlay3::GetPlayerName**, **IDirectPlay3::Send**

IDirectPlay3::SetSessionDesc

Changes the properties of the current session. Only the host of the session can change the session properties. If a nonhost attempts to call it, the method returns DPERR_ACCESSDENIED.

The updated session description will be propagated to all the other computers in the session. Each player will receive a **DPMSG_SETSESSIONDESC** system message.

You can't use **SetSessionDesc** in a lobby session.

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

Parameters

lpSessDesc
    Pointer to the **DPSESSIONDESC2** structure containing the new settings.

dwFlags
    No flags are currently used by this method.

Return Values

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

DPERR_ACCESSDENIED

DPERR_INVALIDPARAMS

DPERR_NOSESSIONS

This method returns **DPERR_ACCESSDENIED** if this computer does not have permission to change the session. Only the host can change the session properties. It returns **DPERR_INVALIDPARAMS** if the application attempts to change a property in the session description that cannot be changed or tries to give a property an invalid value.

Remarks

Changing the session description will cause a *DPSYS_SETSESSIONDESC* system message to be generated on all the other computers in the session.

The following members and flags of the **DPSESSIONDESC2** structure can be changed using **IDirectPlay3::SetSessionDesc**:

dwFlags

DPSESSION_JOINDISABLED

DPSESSION_MIGRATEHOST

DPSESSION_NEWPLAYERSDISABLED

DPSESSION_NODATAMESSAGES

DPSESSION_PRIVATE

dwMaxPlayers

If you set the maximum players to a value less than the current number of players, the method returns **DPERR_INVALIDPARAMS**.

lpszSessionName / *lpszSessionNameA*

lpszPassword / *lpszPasswordA*

dwUser1

dwUser2

dwUser3

dwUser4

If the following members and flags of the **DPSESSIONDESC2** structure are changed, an error will occur (for example, if the **DPSESSION_KEEPALIVE** flag is currently set and you try to set this bit, you will not get an error, but if you try to clear this bit, you will get an error and **IDirectPlay3::SetSessionDesc** will fail):

dwSize

dwFlags

DPSESSION_NOMESSAGEID

DPSESSION_KEEPALIVE

The following members of the **DPSESSIONDESC2** structure are ignored (that is, it does not matter what you pass in these members — DirectPlay will always use the values passed when **IDirectPlay3::Open** was called):

guidInstance

guidApplication

dwCurrentPlayers

dwReserved1

dwReserved2

See Also **DPSESSIONDESC2**, **IDirectPlay3::GetSessionDesc**

IDirectPlay3::StartSession

Use this method to initiate the launch of a DirectPlay session. Call **SetGroupConnectionSettings** first to specify what application to launch, which service provider to use, and what session description to use.

When **StartSession** is called, each player in the group receives a **DPMSG_STARTSESSION** instructing the player to launch the session.

```
HRESULT StartSession(
    DWORD dwFlags,
    DPID idGroup
```

);

Parameters

dwFlags
Not used. Must be zero.

idGroup
The DPID of the group to send start session commands to. The group must be a staging area.

Return Values

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

DPERR_ACCESSDENIED
DPERR_INVALIDFLAGS
DPERR_INVALIDGROUP
DPERR_INVALIDPARAMS

This method returns **DPERR_INVALIDGROUP** if the given group ID is not a valid staging area.

Remarks

A player joining a staging area group for which the session has already started can call this method to join the session. The player will receive a **DPMSG_STARTSESSION** message instructing it how to join the session.

You can determine if a session is already in progress by calling **IDirectPlay3::GetGroupConnectionSettings** and examining the *guidInstance* parameter it returns through a pointer in the **DPLCONNECTION** structure to the **DPSESSIONDESC2** structure. If *guidInstance* is GUID_NULL, no session is in progress. If it is any other value, a session is already in progress.

See Also

DPMSG_STARTSESSION

IDirectPlayLobby2

Applications use the methods of the **IDirectPlayLobby2** interface to manage applications and their associated data. This section is a reference to the methods of this interface. For a conceptual overview, see *IDirectPlayLobby2 Interface*.

Address management	**CreateAddress**
	CreateCompoundAddress
	EnumAddress
	EnumAddressTypes
Application management	**Connect**
	EnumLocalApplications
	RunApplication

Data management	**GetConnectionSettings**
	ReceiveLobbyMessage
	SendLobbyMessage
	SetConnectionSettings
	SetLobbyMessageEvent

IDirectPlayLobby2::Connect

Connects an application to the session specified by the **DPLCONNECTION** structure currently stored with the DirectPlayLobby object.

Note This method will return an **IDirectPlay2** or **IDirectPlay2A** interface. Use the standard COM **QueryInterface** method to obtain an **IDirectPlay3** or *IDirectPlay3A* method.

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

Parameters

dwFlags
  Reserved; must be zero.

lplpDP
  Pointer to a pointer to be initialized with a valid interface — either **IDirectPlay2** (if called on **IDirectPlayLobby2**) or **IDirectPlay2A** (if called on **IDirectPlayLobby2A**).

pUnk
  Pointer to the containing *IUnknown* interface. This parameter is provided for future compatibility with COM aggregation features. Presently, however, **IDirectPlayLobby2::Connect** returns an error if this parameter is anything but NULL.

Return Values

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

Remarks

After this method is successfully completed, the application can skip the process of calling **IDirectPlay3::InitializeConnection**, **IDirectPlay3::EnumSessions**, and **IDirectPlay3::Open**. The application should not ask the user a name but instead create a player using the player name information in the **DPLCONNECTION** structure.

Before calling this method, the application can examine the connection settings that will be used to start the application by using the **IDirectPlayLobby2::GetConnectionSettings** method. The application then can modify these settings and set them by using the **IDirectPlayLobby2::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.

IDirectPlayLobby2::CreateAddress

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.

For more information about the DirectPlay Address, see *DirectPlay Address*. For a list of predefined Microsoft data types, see *DirectPlay Address Data Types*.

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

Parameters

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

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

lpData
Pointer to a buffer containing the specific network address.

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

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

lpdwAddressSize
Pointer to 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.

Return Values

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDPARAMS

Remarks

The **IDirectPlayLobby2::CreateCompoundAddress** method can be used to create longer DirectPlay Addresses than **CreateAddress** allows.

See Also

IDirectPlayLobby2::EnumAddress, **IDirectPlayLobby2::CreateCompoundAddress**

IDirectPlayLobby2::CreateCompoundAddress

Creates a *DirectPlay Address* from a list of individual data chunks. This method can be used to create longer DirectPlay Addresses than **IDirectPlayLobby2::CreateAddress** allows. For a list of predefined Microsoft data types, see *DirectPlay Address Data Types*.

```
HRESULT CreateCompoundAddress(
    LPDPCOMPOUNDADDRESSELEMENT lpElements,
    DWORD dwElementCount,
    LPVOID lpAddress,
    LPDWORD lpdwAddressSize
    );
```

Parameters

lpElements
: Pointer to the first element in an array of **DPCOMPOUNDADDRESSELEMENT** structures that will be used to generate the *DirectPlay Address*.

dwElementCount
: The number of address elements in the array pointed to by the *lpElements* parameter.

lpData
: Pointer to a buffer that the complete DirectPlay Address is to be written to. Pass NULL if only the required size of the buffer is desired.

lpdwDataSize
: Pointer to a DWORD with the size of the *lpData* buffer. The DWORD will be modified to reflect the actual number of bytes copied into the buffer. If the buffer was too small, it will contain the number of bytes required.

Return Values

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

DPERR_BUFFERTOOSMALL

DPERR_INVALIDFLAGS

DPERR_INVALIDPARAMS

See Also

IDirectPlayLobby2::CreateAddress, **IDirectPlayLobby2::EnumAddress**, **DPCOMPOUNDADDRESSELEMENT**

IDirectPlayLobby2::EnumAddress

Parses out chunks from the DirectPlay Address buffer.

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

Parameters

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

lpAddress
: Pointer to the start of the DirectPlay Address buffer.

dwAddressSize
: Size of the DirectPlay Address.

lpContext
Context that will be passed to the callback function.

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

DPERR_EXCEPTION
DPERR_INVALIDOBJECT
DPERR_INVALIDPARAMS

Remarks
For more information about the DirectPlay Address, see *DirectPlay Address*.

See Also
IDirectPlayLobby2::CreateAddress

IDirectPlayLobby2::EnumAddressTypes

Enumerates all the address types that a given service provider needs to build the DirectPlay Address. The application or lobby can use this information to obtain the correct information from the user and create a DirectPlay Address.

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

Parameters

lpEnumAddressTypeCallback
Pointer to 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
Pointer to 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.

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

DPERR_EXCEPTION
DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

Remarks

For more information about the DirectPlay Address, see *DirectPlay Address*. You can use **EnumAddressTypes** to determine if a service provider displays dialog boxes prompting the user for information; for example, a dialog box that asks for an IP address. If the service provider takes no address types, then it needs no information and will not display the dialog boxes.

An application can call **IDirectPlay3::GetPlayerAddress** to obtain a list of valid choices for an address type. This is only available for the modem-to-modem service providers. DirectPlay Address data types that are null-terminated Unicode strings end in W (for example, DPAID_INetW), while DirectPlay Address data types that are null-terminated ANSI strings do not (for example, DPAID_INet). For a list of predefined Microsoft data types, see *DirectPlay Address Data Types*.

See Also

IDirectPlayLobby2::CreateAddress

IDirectPlayLobby2::EnumLocalApplications

Enumerates what applications are registered with DirectPlay.

HRESULT EnumLocalApplications(
    LPDPENUMLOCALAPPLICATIONSCALLBACK
*lpEnumLocalAppCallback***,**
    LPVOID *lpContext***,**
    DWORD *dwFlags*
    );

Parameters

lpEnumLocalAppCallback
Pointer to the **EnumLocalApplicationsCallback** function that will be called for each enumerated application.

lpContext
Context passed to the callback function.

dwFlags
Reserved; must be zero.

Return Values

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

DPERR_GENERIC

DPERR_INVALIDINTERFACE

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_OUTOFMEMORY

See Also **DPLAPPINFO**

IDirectPlayLobby2::GetConnectionSettings

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 **IDirectPlayLobby2::RunApplication** method by the lobby client, or set by calling the **IDirectPlayLobby2::SetConnectionSettings** method.

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

Parameters

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 **IDirectPlayLobby2::RunApplication**.

lpData
  Pointer to 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
  Pointer to 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.

Return Values 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

Remarks

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

See Also

DPLCONNECTION, **IDirectPlayLobby2::RunApplication**, **IDirectPlayLobby2::SetConnectionSettings**

IDirectPlayLobby2::ReceiveLobbyMessage

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.

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

Parameters

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 **IDirectPlayLobby2::RunApplication** method.

lpdwMessageFlags
Flags indicating what type of message is being returned. The default (*lpdwMessageFlags* = 0) indicates that the message is custom-defined by the sender. Processing of this type of message is optional. The receiver must interpret this message based on the identity of the sending application. A lobby can identify the sending application based on the GUID of the application that was launched. An application will need to identify the lobby by sending the lobby a standard message requesting an identifying GUID.

DPLMSG_STANDARD
Indicates that this is a DirectPlay-defined message. Processing of this type of message is optional.

DPLMSG_SYSTEM
Indicates that this is a DirectPlay generated system message used to

inform the lobby of changes in the status of the application it launched.

lpData
Pointer to 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
Pointer to 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.

Return Values

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

See Also

IDirectPlayLobby2::RunApplication, **IDirectPlayLobby2::SendLobbyMessage**

IDirectPlayLobby2::RunApplication

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.

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

Parameters

dwFlags
Reserved; must be zero.

lpdwAppId
Pointer to 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 **IDirectPlayLobby2::SendLobbyMessage** and **IDirectPlayLobby2::ReceiveLobbyMessage** methods.

lpConn
Pointer to 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 **IDirectPlayLobby2::SetLobbyMessageEvent** method.

Return Values

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

Remarks

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_DPLAYCONNECTSUCCEEDED 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 *IDirectPlayLobby2* interface before it receives a DPLSYS_CONNECTIONSETTINGSREAD system message. The lobby client can either check **IDirectPlayLobby2::ReceiveLobbyMessage** in a loop until it is received, or supply a synchronization event.

See Also

IDirectPlayLobby2::ReceiveLobbyMessage, **IDirectPlayLobby2::GetConnectionSettings**, **IDirectPlayLobby2::SetLobbyMessageEvent**

IDirectPlayLobby2::SendLobbyMessage

Sends a message between the application and the lobby client.

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

Parameters

dwFlags
Flags indicating the type of message being sent. The default (*dwFlags* = 0) is a custom message defined by the application sending it. Other possible values are:

DPLMSG_STANDARD – this is a standard message defined by DirectPlay.

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 **IDirectPlayLobby2::RunApplication** method.

lpData
Pointer to the buffer containing the message to send.

dwDataSize
Size, in bytes, of the buffer.

Return Values

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

See Also

IDirectPlayLobby2::RunApplication, **IDirectPlayLobby2::ReceiveLobbyMessage**, **DPLMSG_SETPROPERTY**, **DPLMSG_SETPROPERTYRESPONSE**, **DPLMSG_GETPROPERTY**, **DPLMSG_GETPROPERTYRESPONSE**,

IDirectPlayLobby2::SetConnectionSettings

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

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

Parameters

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 **IDirectPlayLobby2::RunApplication** method.

lpConn
  Pointer to 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.

Return Values

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

DPERR_GENERIC

DPERR_INVALIDINTERFACE

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_OUTOFMEMORY

See Also

IDirectPlayLobby2::GetConnectionSettings

IDirectPlayLobby2::SetLobbyMessageEvent

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 **IDirectPlayLobby2::RunApplication** method.

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

Parameters

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 **IDirectPlayLobby2::RunApplication**.

hReceiveEvent
Event handle to be set when a message is received.

Return Values

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

DPERR_GENERIC

DPERR_INVALIDINTERFACE

DPERR_INVALIDOBJECT

DPERR_INVALIDPARAMS

DPERR_OUTOFMEMORY

See Also

IDirectPlayLobby2::ReceiveLobbyMessage, **IDirectPlayLobby2::SendLobbyMessage**

Structures

Some structures have changed for DirectPlay 5, with new data members added to the end. Applications that use the **IDirectPlay2** interface but are compiled using the DirectX 5 header files can have problems. An application might crash if it is run on a computer that has the DirectX 3 run time installed and the application references data members that were added for DirectX 5.

Applications that need to be backward compatible with older run times should either:

- use the structures that existed in the DirectX 3 header files
- check the *dwSize* member in each structure before attempting to access to members of the structure

The structures that have new members for DirectPlay 5 are:

Structure	New Members
DPCREDENTIALS	union of
	LPWSTR lpszDomain
	LPSTR lpszDomainA

The DirectPlay structures are:

- **DPACCOUNTDESC**
- **DPCAPS**
- **DPCHAT**
- **DPCOMPORTADDRESS**
- **DPCOMPOUNDADDRESSELEMENT**
- **DPCREDENTIALS**
- **DPLAPPINFO**
- **DPLCONNECTION**
- **DPNAME**
- **DPSECURITYDESC**
- **DPSESSIONDESC2**

DPACCOUNTDESC

Describes the account information for a specific player.

```
typedef struct {
  DWORD  dwSize;
  DWORD  dwFlags;
  union {
    LPWSTR  lpszAccountID;
    LPSTR   lpszAccountIDA;
  };
} DPACCOUNTDESC, FAR *LPDPACCOUNTDESC;
```

Members

dwSize
The size of the DPACCOUNTDESC structure, *dwsize* = sizeof(DPACCOUNTDESC).

dwFlags
Not used. Must be zero.

lpszAccountID
Pointer to a Unicode string containing the account identifier. This is a unique identifier that describes a player who is securely logged in. The format of the

identifier depends on the Security Support Provider Interface (SSPI) package being used.

lpszAccountIDA
Pointer to an ANSI string containing the account identifier. This is a unique identifier that describes a player who is securely logged in. The format of the identifier depends on the SSPI package being used.

See Also **IDirectPlay3::GetPlayerAccount**

DPCAPS

Contains the capabilities of a DirectPlay object after a call to the **IDirectPlay3::GetCaps** or **IDirectPlay3::GetPlayerCaps** methods. Any of these capabilities can differ depending on whether guaranteed or nonguaranteed capabilities are requested. This structure is read-only.

```
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;
```

Members

dwSize
The size of the DPCAPS structure, *dwsize* = sizeof(DPCAPS). 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_ENCRYPTIONSUPPORTED
Indicates that message encryption is supported by this DirectPlay object, either because it is a secure session or because the service provider can encrypt messages.

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.

DPCAPS_SIGNINGSUPPORTED

Indicates that message authentication is supported by this DirectPlay object , either because it is a secure session or because the service provider can sign messages.

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 bits per second. .

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 time-out value. By default, DirectPlay will use this time-out value when waiting for replies to messages.

See Also **IDirectPlay3::Send**

DPCHAT

Contains a DirectPlay chat message. Chat messages are sent with the **IDirectPlay3::SendChatMessage** method.

```
typedef struct {
    DWORD   dwSize;
    DWORD   dwFlags;
    union{
      LPWSTR lpszMessage;
      LPSTR  lpszMessageA;
    };
} DPCHAT, FAR *LPDPCHAT;
```

Members

dwSize
The size of the DPCHAT structure, *dwsize* = sizeof(DPCHAT).

dwFlags
Not used. Must be 0.

lpszMessage
Pointer to a Unicode string containing the message to be sent. Can only be used with a Unicode interface (**IDirectPlay3**).

lpszMessageA
Pointer to an ANSI string containing the message to be sent. Can only be used with an ANSI interface (*IDirectPlay3A*).

See Also **IDirectPlay3::SendChatMessage**

DPCOMPORTADDRESS

Contains information about the configuration of the COM port.

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

typedef DPCOMPORTADDRESS FAR* LPDPCOMPORTADDRESS;
```

Members

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).

Remarks

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

DPCOMPOUNDADDRESSELEMENT

Describes a *DirectPlay Address* data chunk. See *DirectPlay Address Data Types* for a list of predefined Microsoft data types.

```
typedef struct {
    GUID    guidDataType;
    DWORD   dwDataSize;
    LPVOID  lpData;
} DPCOMPOUNDADDRESSELEMENT, FAR *LPDPCOMPOUNDADDRESSELEMENT;
```

Members

guidDataType
GUID identifying the type of data contained in this structure.

dwDataSize
Size of the data in bytes.

lpData
Pointer to a buffer containing the data.

See Also

IDirectPlayLobby2::CreateCompoundAddress

DPCREDENTIALS

Holds the user name, password, and domain to connect to a secure server.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwFlags;
    union {
           LPWSTR lpszUsername;
           LPSTR  lpszUsernameA;
          };
    union {
           LPWSTR lpszPassword;
           LPSTR  lpszPasswordA;
          };
    union {
           LPWSTR lpszDomain;
           LPSTR  lpszDomainA;
          };

} DPCREDENTIALS, FAR *LPDPCREDENTIALS;
```

Members

dwSize
The size of the **DPCREDENTIALS** structure, *dwsize* = sizeof(DPCREDENTIALS).

dwFlags
Not used. Must be zero.

lpszUsername, **lpszPassword**, **lpszDomain**
Pointers to Unicode strings containing the user name, password, and domain name. Can only be used with a Unicode interface.

lpszUsernameA, **lpszPasswordA**, **lpszDomainA**
Pointers to ANSI strings containing the user name, password, and domain name. Can only be used with an ANSI interface.

See Also

IDirectPlay3::SecureOpen

DPLAPPINFO

Contains information about the application from the registry and is passed to the **IDirectPlayLobby2::EnumLocalApplications** callback function.

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

Members

dwSize
The size of the DPLAPPINFO structure, *dwsize* = sizeof(DPLAPPINFO). 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

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

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

Members

dwSize
The size of the DPLCONNECTION structure, *dwsize* = sizeof(DPLCONNECTION). Your application must set this member before it uses this structure; otherwise, an error will result. Examine this member to determine if the fields added in DirectPlay version 5 are available.

dwFlags
Indicates how to open a session.

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
Pointer to a **DPSESSIONDESC2** structure describing the session to be created or the session to join.

lpPlayerName
Pointer to a **DPNAME** structure holding the name 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
Pointer to a DirectPlay Address that contains the information that the service provider needs to connect to a session. For more information about the DirectPlay Address, see *DirectPlay Address (Optional)*. For a list of Microsoft predefined address data types, see *DirectPlay Address Data Types*.

dwAddressSize
Size, in bytes, of the address data.

See Also **IDirectPlayLobby2::RunApplication**, **IDirectPlayLobby2::GetConnectionSettings**, **IDirectPlayLobby2::SetConnectionSettings**

DPNAME

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

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

Members

dwSize
The size of the DPNAME structure, *dwsize* = sizeof(DPNAME). 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**
Pointers to Unicode strings containing the short (friendly) and long (formal) names of a player or group. Use these members only if the *IDirectPlay3* interface is in use.

lpszShortNameA and **lpszLongNameA**
Pointers to ANSI strings containing the short (friendly) and long (formal) names of a player or group. Use these members only if the *IDirectPlay3A* interface is in use.

See Also

IDirectPlay3::CreateGroup, **IDirectPlay3::CreatePlayer**, **IDirectPlay3::GetGroupName**, **IDirectPlay3::GetPlayerName**, **IDirectPlay3::SetGroupName**, **IDirectPlay3::SetPlayerName**

DPSECURITYDESC

Describes the security properties of a DirectPlay session instance.

```
typedef struct {
    DWORD  dwSize;
    DWORD  dwFlags;
    union {
           LPWSTR  lpszSSPIProvider;
           LPSTR   lpszSSPIProviderA;
    };
    union {
           LPWSTR lpszCAPIProvider;
           LPSTR  lpszCAPIProviderA;
    };
    DWORD  dwCAPIProviderType;
    DWORD  dwEncryptionAlgorithm;
} DPSECURITYDESC, FAR *LPDPSECURITYDESC;
```

Members

dwSize
The size of the DPSECURITYDESC structure, *dwsize* = sizeof(DPSECURITYDESC).

dwFlags
Not used. Must be zero.

lpszSSPIProvider, **lpszSSPIProviderA**
Pointer to a Unicode or ANSI string describing the Security Support Provider Interface (SSPI) package to use for authenticated logins. Pass NULL to use the default, the NTLM (NT LAN Manager) security provider.

lpszCAPIProvider, **lpszCAPIProviderA**
Pointer to a Unicode or ANSI string describing the CryptoAPI package to use for cryptography services. Pass NULL to use the default, the Microsoft RSA Base Cryptographic Provider v. 1.0.

dwCAPIProviderType
CryptoAPI service provider type. Pass zero to use the default type, PROV_RSA_FULL.

dwEncryptionAlgorithm
Encryption algorithm to use. DirectPlay only supports stream ciphers. Pass zero to use the default, the CALG_RC4 stream cipher.

Remarks

For more information about the CryptoAPI, see the CryptoAPI amd Cryptography topics at http://www.microsoft.com. The Microsoft RSA Base Cryptographic Provider is supplied by Microsoft and is included with the Windows 95 and Windows NT operating systems.

DirectPlay does not support block encryption.

See Also

IDirectPlay3::SecureOpen

DPSESSIONDESC2

Contains a description of an *IDirectPlay3* session's capabilities. (The **DPSESSIONDESC** structure is no longer used in the **IDirectPlay3** interface.)

```
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;
```

Members

dwSize

The size of the DPSESSIONDESC2 structure, *dwsize* = sizeof(DPSESSIONDESC2). Your application must set this member before it uses this structure; otherwise, an error will result.

dwFlags

A combination of the following flags. These flags can be changed after a session has started by using **IDirectPlay3::SetSessionDesc**.

DPSESSION_CLIENTSERVER

This is a client/server session that is being hosted by an application server process. This flag must be specified at the time the session is created. Any clients that join this session will only be able to enumerate the server player and any local players. The host of the session will see all the players. If this flag is not set, the session is a peer-to-peer session. If this flag is set, the host can only create a server player. This flag cannot be used with DPSESSION_MIGRATEHOST.

DPSESSION_JOINDISABLED

No new applications can join this session. Any call to the **IDirectPlay3::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 through the **DPMSG_SESSIONLOST** system message. This flag must be specified at the time the session is created. If this flag is not specified, DirectPlay will not automatically keep the session alive if players are abnormally terminated.

DPSESSION_MIGRATEHOST

If the current host exits, another computer in the session will become the host. The players on the new host computer will receive a **DPMSG_HOST** system message. This flag must be specified at the time the session is created. If this flag is not specified, the host will not migrate, new computers cannot join the session, and new players cannot be created if the current host leaves. Note that the

DPSESSION_MIGRATEHOST flag cannot be used with the DPSESSION_CLIENTSERVER, DPSESSION_MULTICASTSERVER, or DPSESSION_SECURESERVER flags.

DPSESSION_MULTICASTSERVER

In a peer-to-peer session, broadcast and group messages are routed through the host which acts like a multicast server. This flag must be specified at the time the session is created. The flag is only useful if the host has a high bandwidth connection to the network. If this flag is not specified, broadcast and group messages are sent directly between peers. This flag cannot be used with DPSESSION_MIGRATEHOST.

DPSESSION_NEWPLAYERSDISABLED

Indicates that new players cannot be created in the session. Any call to the **IDirectPlay3::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 data, group data, or session data is changed with the **IDirectPlay3::SetPlayerData**, **IDirectPlay3::SetGroupData**, **IDirectPlay3::SetPlayerName**, **IDirectPlay3::SetGroupName**, or **IDirectPlay3::SetSessionDesc** method. If this flag is not specified, messages will be generated indicating that the data changed.

Note that setting this flag also suppresses the **DPMSG_SETSESSIONDESC** message.

DPSESSION_NOMESSAGEID

Do not attach data to messages indicating what player the message is from and to whom it is sent. Saves message overhead if this information is not relevant. (For more information, see the **IDirectPlay3::Receive** method.) If this flag is not specified, the message ID will be added. This flag must be specified at the time the session is created.

DPSESSION_PASSWORDREQUIRED

This session is password protected. Any applications wishing to join the session must supply the password in the **IDirectPlay3::Open** call. This is a read-only flag. It will be set automatically by DirectPlay when the host of the session specifies a non-NULL password.

DPSESSION_PRIVATE

This is a private session. It can not respond to enumeration requests unless the request contains a non-NULL, matching password. If this flag is not specified, the session will respond to enumeration requests.

DPSESSION_SECURESERVER

This session is being hosted on a secure server which will require authentication credentials before it can be opened. The host must specify this flag when the session is created. It cannot be changed later through

IDirectPlay3::SetSessionDesc. This flag can only be used if the DPSESSION_CLIENTSERVER flag is also specified. This flag cannot be used with the DPSESSION_MIGRATEHOST flag.

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. A value of zero means there is no maximum.

dwCurrentPlayers
Number of players currently in the session.

lpszSessionName
Pointer to a Unicode string containing the name of the session. Use only if a Unicode DirectPlay interface (*IDirectPlay3*) is being used.

lpszSessionNameA
Pointer to an ANSI string containing the name of the session. Use only if an ANSI DirectPlay interface (*IDirectPlay3A*) is being used.

lpszPassword
Pointer to a Unicode string containing the password used to join the session for participation. Use only if a Unicode DirectPlay interface (*IDirectPlay3*) is being used.

lpszPasswordA
Pointer to an ANSI string containing the password used to join the session for participation. Use only if an ANSI DirectPlay interface (*IDirectPlay3A*) is being used.

dwReserved1 and **dwReserved2**
Must be zero.

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

See Also

IDirectPlay3::EnumSessions, **IDirectPlay3::GetSessionDesc**, **IDirectPlay3::Open**, **IDirectPlay3::SecureOpen**, **IDirectPlay3::SetSessionDesc**, **IDirectPlay3::CreatePlayer**, **DPPLAYER_SERVERPLAYER**

System Messages

Some system message structures have changed for DirectPlay 5, with new data members added to the end. Make sure you only reference these new members if you are using the **IDirectPlay3** interface.

The system message structures that have new members for DirectPlay 5 are:

Structure	**New Members**
DPMSG_CREATEPLAYERORGROUP	DPID dpIdParent
	DWORD dwFlags
DPMSG_DESTROYPLAYERORGROUP	DPNAME dpnName
	DPID dpIdParent
	DWORD dwFlags

DPMSG_ADDGROUPTOGROUP

DirectPlay generates this message and sends it to every player when a group is added to a group.

```
typedef struct{
    DWORD    dwType;
    DPID     dpIdParentGroup;
    DPID     dpIdGroup;
} DPMSG_ADDGROUPTOGROUP, FAR *LPDPMSG_ADDGROUPTOGROUP;
```

Members

dwType
Identifies the message. This member is DPSYS_ADDGROUPTOGROUP.

dpIdParentGroup
ID of the group to which the group was added.

dpIdGroup
ID of the group that was added to the group.

See Also **IDirectPlay3::AddGroupToGroup**

DPMSG_ADDPLAYERTOGROUP

Contains information for the DPSYS_ADDPLAYERTOGROUP system message. DirectPlay generates this message and sends it to every player when a player is added to a group.

The application can use **IDirectPlay3::GetPlayerCaps**, **IDirectPlay3::GetPlayerName**, and **IDirectPlay3::GetPlayerData** for information about the player involved int his message, or **IDirectPlay3::GetGroupName** and **IDirectPlay3::GetGroupData** to get more information about the group involved in this message.

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

Members

dwType
Identifies the message. This member is DPSYS_ADDPLAYERTOGROUP.

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

dpIdPlayer
ID of the player that was added to the group.

See Also **IDirectPlay3::AddPlayerToGroup**

DPMSG_CHAT

Contains information for the DPSYS_CHAT system message. The system message is generated for a local player receiving a chat message from another player.

```
typedef struct{
    DWORD dwType;
    DWORD dwFlags;
    DPID  idFromPlayer;
    DPID  idToPlayer;
    DPID  idToGroup;
    LPDPCHAT  lpChat;
} DPMSG_CHAT, FAR *LPDPMSG_CHAT;
```

Members

dwType
Identifies the message. This member has the value DPSYS_CHAT.

dwFlags
Not used.

idFromPlayer
The DPID of the player from whom the message originated. DPID_SERVERPLAYER indicates the message originated from the server.

idToPlayer
The DPID of the player to whom the message was directed. If this DPID is zero, then the message was sent to a group or broadcast to everyone.

idToGroup
The DPID of the group to whom the message was directed. If this DPID is zero and *idToPlayer* is also zero, then the message was broadcast to everyone.

lpChat
Pointer to a **DPCHAT** structure containing the content of the chat message received.

Remarks
Use the *idFromPlayer* value to determine who sent the message. The value of *lpidFrom* available through **Receive** will always be zero, so you must retrieve the value of *idFromPlayer* in the **DPMSG_CHAT** structure.

See Also
IDirectPlay3::SendChatMessage, **DPCHAT**, **IDirectPlay3::Receive**

DPMSG_CREATEPLAYERORGROUP

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

DirectPlay generates this message and sends it to each player when a new player or group is created in a session.

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

Members

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
The ID of a player or group created.

dwCurrentPlayers
Current number of players in the session before this player was created.

lpData
Pointer to 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.

dpIdParent
The ID of the parent group if this message is caused by a call to **IDirectPlay3::CreateGroupInGroup**; otherwise, the value is 0.

dwFlags
The player or group flags.

Remarks
The DirectPlay 5 version of this structure has two members added at the end, *dpIdParent* and *dwFlags*.

See Also
IDirectPlay3::CreateGroup, **IDirectPlay3::CreatePlayer**; **IDirectPlay3::CreateGroupInGroup**

DPMSG_DELETEGROUPFROMGROUP

DirectPlay generates this message and sends it to every player when a group is removed from a group. Note that this structure is identical to **DPMSG_ADDGROUPTOGROUP**.

```
typedef struct{
    DWORD    dwType;
    DPID     dpIdParentGroup;
    DPID     dpIdGroup;
} DPMSG_ADDGROUPTOGROUP, FAR *LPDPMSG_ADDGROUPTOGROUP;
typedef DPMSG_ADDGROUPTOGROUP  DPMSG_DELETEGROUPFROMGROUP;
```

Members

dwType
Identifies the message. This member is DPSYS_DELETEGROUPFROMGROUP.

dpIdParentGroup
ID of the group from which the group was removed.

dpIdGroup
ID of the group that was removed from the group.

See Also **IDirectPlay3::DeleteGroupFromGroup**

DPMSG_DELETEPLAYERFROMGROUP

Contains information for the DPSYS_DELETEPLAYERFROMGROUP system message. DirectPlay generates this message and sends it to each local player on the computer when a player is deleted from a group.

For a description of the structure members, see the **DPMSG_ADDPLAYERTOGROUP** structure.

The application can use **IDirectPlay3::GetPlayerCaps**, **IDirectPlay3::GetPlayerName**, and **IDirectPlay3::GetPlayerData** for information about the player involved int his message, or **IDirectPlay3::GetGroupName** and **IDirectPlay3::GetGroupData** to get more information about the group involved in this message.

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

Members

dwType
Identifies the message. This member is DPSYS_DELETEPLAYERFROMGROUP.

dpIdGroup
ID of the group from which the player was removed.

dpIdPlayer
ID of the player that was removed from the group.

See Also **IDirectPlay3::DeletePlayerFromGroup**

DPMSG_DESTROYPLAYERORGROUP

Contains information for the DPSYS_DDESTROYPLAYERORGROUP system message. DirectPlay generates this message and sends it to each player when a player or group is destroyed in a session.

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

Members

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
Pointer to the local data associated with this player/group.

dwLocalDataSize
Size, in bytes, of the local data.

lpRemoteData
Pointer to the remote data associated with this player/group.

dwRemoteDataSize
Size, in bytes, of the remote data.

dpnName
Structure containing the name of the player/group.

dpIdParent
The ID of the parent group if the group being destroyed is a subgroup of the parent group (the group being destroyed was created by a call to **IDirectPlay3::CreateGroupInGroup**; otherwise, the value is 0.

dwFlags
The player or group flags.

Remarks

The DirectPlay 5 version of this structure has three members added at the end, *dpnName*, *dpIdParent* and *dwFlags*.

See Also

IDirectPlay3::DestroyGroup, **IDirectPlay3::DestroyPlayer**

DPMSG_GENERIC

This structure is provided for message processing.

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

Members

dwType
Identifies the system message type.

Remarks

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

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

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

DPMSG_SECUREMESSAGE

DirectPlay generates this message when it receives a signed or encrypted message from another player.

```
typedef struct {
    DWORD   dwType;
    DWORD   dwFlags;
    DPID    dpIdFrom;
    LPVOID  lpData;
    DWORD   dwDataSize;
} DPMSG_SECUREMESSAGE, FAR *LPDPMSG_SECUREMESSAGE;
```

Members

dwType
Identifies the system message type. This is DPSYS_SECUREMESSAGE.

dwFlags
Flags indicating how the message was secured by the sender. One of the following values:

DPSEND_SIGNED - the message was signed by the sender and the signature was successfully verified.

DPSEND_ENCRYPTED - the messages was encrypted by the sender and successfully decrypted.

dpIdFrom
The DPID of the player that sent the secure message.

lpData
Pointer to a buffer containing the fully verified message.

dwDataSize
Size of the buffer containing the message.

See Also **IDirectPlay3::Send**

DPMSG_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.

```
typedef DPMSG_GENERIC    DPMSG_SESSIONLOST;
typedef DPMSG_SESSIONLOST  FAR  *LPDPMSG_SESSIONLOST;
```

DPMSG_SETPLAYERORGROUPDATA

Contains information for the DPSYS_SETPLAYERORGROUPDATA system message. DirectPlay generates this message and sends it to each player when the remote data of a player or group changes. This message will not be generated if the DPSESSION_ NODATAMESSAGES flag is specified in the session description.

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

Members

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
Pointer to an application-specific block of data.

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

Remarks

It is not necessary for the application to save the data from this message because it can be retrieved at any time using **IDirectPlay3::GetPlayerData** or **IDirectPlay3::GetGroupData**.

See Also

IDirectPlay3::GetPlayerData, **IDirectPlay3::GetGroupData**, **IDirectPlay3::SetPlayerData**, **IDirectPlay3::SetGroupData**

DPMSG_SETPLAYERORGROUPNAME

Contains information for the DPSYS_SETPLAYERORGROUPNAME system message. DirectPlay generates this message and sends it to each local player on the computer when the name of a player or group changes. This message will not be generated if the DPSESSION_ NODATAMESSAGES flag is specified in the session description.

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

Members

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.

Remarks It is not necessary for the application to save the *dpnName* from this message because it can be retrieved at any time using **IDirectPlay3::GetPlayerName** or **IDirectPlay3::GetGroupName**.

See Also **IDirectPlay3::GetPlayerName**, **IDirectPlay3::GetGroupName**, **IDirectPlay3::SetPlayerName**, **IDirectPlay3::SetGroupName**

DPMSG_SETSESSIONDESC

Contains information for the DPSYS_SETSESSIONDESC system message. Every player will receive this system message when the session description changes.

This messages will not be generated if the **DPSESSION_NODATAMESSAGES** flag is specified in the session description.

```
typedef struct
{
    DWORD           dwType;
    DPSESSIONDESC2  dpDesc;
} DPMSG_SETSESSIONDESC, FAR *LPDPMSG_SETSESSIONDESC;
```

Members

dwType
Identifies the message. This member is always DPSYS_SETSESSIONDESC.

dpDesc
Pointer to a **DPSESSIONDESC2** structure containing the updated session description.

See Also **IDirectPlay3::GetSessionDesc**, **IDirectPlay3::SetSessionDesc**

DPMSG_STARTSESSION

Contains information for the DPSYS_STARTSESSION system message. The lobby server sends this message to each player in a group when it is time for that player to join an application session.

```
typedef struct {
   DWORD            dwType;
   LPDPLCONNECTION  lpConn;
} DPMSG_STARTSESSION, FAR *LPDPMSG_STARTSESSION;
```

Members

dwType
Identifies the message. This member is always DPSYS_STARTSESSION.

lpConn
Pointer to a **DPLCONNECTION** structure that contains all the information needed to launch an application into a session. This structure can be passed into **IDirectPlayLobby2::RunApplication** (if an external application is to be launched) or passed into **IDirectPlayLobby2::SetConnectionSettings** and then followed by a call to **IDirectPlayLobby2::Connect** (to connect the current application to the session).

Remarks

Upon receipt of this message by a stand-alone lobby client, the client must launch the application by calling the **IDirectPlayLobby2::RunApplication** method with the **DPLCONNECTION** structure.

Upon receipt of this message by an application with an internal lobby, the application must set the connection settings by calling **IDirectPlayLobby2::SetConnectionSettings** and then connect to the session using the **IDirectPlayLobby2::Connect** method.

See Also

IDirectPlay3::StartSession, **IDirectPlayLobby2::RunApplication**, **IDirectPlayLobby2::SetConnectionSettings**, **IDirectPlayLobby2::Connect**

Standard Lobby Messages

To determine whether the lobby that launched your application supports standard lobby messages, it is necessary to send the lobby an initial **DPLMSG_GETPROPERTY** request and see whether it responds or not. The property that should be requested is **DPLPROPERTY_MessagesSupported**.

If the lobby responds with TRUE, then you can assume that it will respond to all further messages. If it responds with FALSE or doesn't respond at all within a specified timeout, then you can assume that messages are not supported.

If the application supports spectator players, then the application should also request **DPLPROPERTY_PlayerGuid** before creating players to determine if they should be created as spectators or not.

DPLMSG_GENERIC

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

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

Members

dwType
Identifies what type of system message has been received.

DPLSYS_APPTERMINATED
Indicates the application started by **IDirectPlayLobby2::RunApplication** has terminated.

DPLSYS_CONNECTIONSETTINGSREAD
Indicates the application started by the **IDirectPlayLobby2::RunApplication** method has read the connection settings.

DPLSYS_DPLAYCONNECTFAILED
Indicates the application started by **IDirectPlayLobby2::RunApplication** failed to connect to a session.

DPLSYS_DPLAYCONNECTSUCCEEDED
Indicates the application started by **IDirectPlayLobby2::RunApplication** has created a session and is ready for other applications to join or has successfully joined a session.

DPLMSG_GETPROPERTY

Message sent by an application to the lobby to request the current value of a property. These properties can be information such as the ranking of a player, a bitmap representing a player, or initial configuration information for the game or players that was done inside the lobby.

```
typedef struct {
    DWORD   dwType;
    DWORD   dwRequestID;
    GUID    guidPlayer;
    GUID    guidPropertyTag;
} DPLMSG_GETPROPERTY, FAR *LPDPLMSG_GETPROPERTY;
```

Members

dwType
Identifies the message. This value is DPSYS_ GETPROPERTY.

dwRequestID
An application generated ID to identify the request. When the lobby responds, it will be tagged with this request ID. The application can use the request ID to match responses to pending requests.

guidPlayer
GUID identifying the player that this property applies to (if applicable). If the property is not player-specific, this member should be set to GUID_NULL. The GUID for the player or players created by this application can be obtained from the lobby by requesting the **DPLPROPERTY_PlayerGuid** property.

guidPropertyTag
A GUID identifying the property that is being requested. The property can one of the predefined ones listed in the section *DirectPlay Defined Properties* or the application/lobby can define its own GUIDs for additional properties.

Remarks

Each property is identified by a GUID (defined by the application developer or the lobby developer). When a request for a property is made, the lobby responds with a **DPLMSG_GETPROPERTYRESPONSE** message. Even if the lobby cannot supply the information, it should respond with an error indicating the information is unavailable.

The application should not block waiting for a response and should have a way to time-out pending requests that haven't been fulfilled.

See Also

DPLMSG_GETPROPERTYRESPONSE

DPLMSG_GETPROPERTYRESPONSE

Message sent by a lobby to an application in response to a **DPLMSG_GETPROPERTY** message. The request that is being filled is identified by the *dwRequestID* parameter.

```
typedef struct {
    DWORD    dwType;
    DWORD    dwRequestID
    GUID     guidPlayer;
    GUID     guidPropertyTag;
    HRESULT  hr;
    DWORD    dwDataSize;
    DWORD    dwPropertyData[1];
} DPLMSG_GETPROPERTYRESPONSE, FAR *LPDPLMSG_GETPROPERTYRESPONSE;
```

Members

dwType
Identifies the message. This value is DPSYS_ GETPROPERTYRESPONSE.

dwRequestID
The ID that identifies the **DPLMSG_GETPROPERTY** message that this message is in response to.

guidPlayer
GUID identifying the player that this property applies to (if applicable). If the property is not player-specific, this member will be set to GUID_NULL. This will be the same as the GUID from the **DPLMSG_GETPROPERTY** message.

guidPropertyTag
A GUID identifying the property that is being requested. This will be the same as the GUID from the **DPLMSG_GETPROPERTY** message.

hr
Return code for the get property request. One of the following values:

DP_OK - successfully returned the property.

DPERR_UNKNOWN - the requested property is unknown to the lobby.

DPERR_UNAVAILABLE - the requested property is unavailable.

dwDataSize
The size, in bytes, of the property data.

dwPropertyData
A variable-size buffer that contains the property data. The property tag will define how to interpret this data.

Remarks

A lobby must either respond to all **DPLMSG_GETPROPERTY** requests or none of them.

When constructing this message, the lobby needs to allocate enough memory to hold the **DPLMSG_GETPROPERTYRESPONSE** structure and the complete property data. For example, if the property data requires 52 bytes, the lobby will allocate (sizeof(DPLMSG_GETPROPERTYRESPONSE) + 52) bytes and assign it to a **DPLMSG_GETPROPERTYRESPONSE** pointer.

See Also

DPLMSG_GETPROPERTY

DPLMSG_SETPROPERTY

Message sent by an application to the lobby to inform it that a property of a specific player or a property of the session has changed. These properties can range from the score or status of a player to the current level of a game or the current status of the session.

```
typedef struct {
    DWORD    dwType;
    DWORD    dwRequestID
    GUID     guidPlayer;
    GUID     guidPropertyTag;
    DWORD    dwDataSize;
    DWORD    dwPropertyData[1];
} DPLMSG_SETPROPERTY, FAR *LPDPLMSG_SETPROPERTY;
```

Members

dwType
Identifies the message. This value is DPSYS_ SETPROPERTY.

dwRequestID
A nonzero request ID supplied by the application if it would like confirmation that the data was recognized and set appropriately. If no confirmation is required, set this to DPL_NOCONFIRMATION.

guidPlayer
GUID identifying the player that this property applies to (if applicable). If the property is not player-specific, this member should be set to GUID_NULL. The GUID for the player or players created by this application can be obtained from the lobby by requesting the **DPLPROPERTY_PlayerGuid** property.

guidPropertyTag
A GUID identifying the property that is being set. The property can be one of the predefined GUIDs listed in the section *DirectPlay Defined Properties* or the application/lobby can define its own GUIDs for additional properties.

dwDataSize
The size, in bytes, of the property data.

dwPropertyData
A variable size buffer that contains the property data. The property tag will define how to interpret this data.

Remarks

Each property is identified by a GUID (defined by the application developer or the lobby developer), and it is the responsibility of the lobby to maintain a mapping of property GUIDs of the various applications to their descriptions and data types. The lobby server can choose to act on the information or ignore it.

When constructing this message, the application needs to allocate enough memory to hold the DPLMSG_ SETPROPERTY structure and the complete property data. For example, if the property data requires 52 bytes, the application will allocate (sizeof(DPLMSG_SETPROPERTY) + 52) bytes and assign it to a LPDPLMSG_SETPROPERTY pointer.

See Also

DPLMSG_SETPROPERTYRESPONSE

DPLMSG_SETPROPERTYRESPONSE

Message sent by a lobby to an application as confirmation of a **DPLMSG_SETPROPERTY** message. The message that is being confirmed is identified by the *dwRequestID* parameter.

```
typedef struct {
    DWORD    dwType;
    DWORD    dwRequestID
    GUID     guidPlayer;
    GUID     guidPropertyTag;
    HRESULT  hr;
} DPLMSG_SETPROPERTYRESPONSE, FAR *LPDPLMSG_SETPROPERTYRESPONSE;
```

Members

dwType
Identifies the message. This value is DPSYS_ SETPROPERTYRESPONSE.

dwRequestID
The ID that identifies the **DPLMSG_SETPROPERTY** message being confirmed.

guidPlayer
GUID identifying the player that this property was set for (if applicable). If the property is not player-specific, this member is GUID_NULL. This must be the same as the GUID from the **DPLMSG_SETPROPERTY** message.

guidPropertyTag
A GUID identifying the property that was set. This must be the same as the GUID from the **DPLMSG_SETPROPERTY** message.

hr
Result of the confirmation. One of the following values:

DP_OK - the property was set correctly.

DPERR_ACCESSDENIED - an attempt was made to set a property the lobby will not allow to be changed.

DPERR_UNKNOWN - the property is unknown to the lobby.

See Also **DPLMSG_SETPROPERTY**

DirectPlay Defined Properties

This section describes the defined properties that can be set and retrieved by applications from a lobby.

When a **DPLMSG_GETPROPERTYRESPONSE** message is received its *guidPropertyTag* field identifies the property and its *dwPropertyData* field contains the property information. To extract this information, simply cast the *dwPropertyData* field to the appropriate value. For example, the following routine retrieves the DPLDATA_PLAYERGUID struture from the given message:

```
BOOL GetPlayerGuid(LPDPLMSG_GETPROPERTYRESPONSE  lpPropertyResponseMsg,
                LPDPLDATA_PLAYERGUID lpPlayerGuid)
{
  if (IsEqualGUID(lpPropertyResponseMsg->guidPropertyTag, DPLPROPERTY_PlayerGuid))
  {
    *lpPlayerGuid = *((LPDPLDATA_PLAYERGUID) lpPropertyResponseMsg->dwPropertyData);
    return (TRUE);
  }
  else
  {
    return (FALSE);
  }
```

See the following descriptions of defined properties:

- *DPLPROPERTY_LobbyGuid*
- *DPLPROPERTY_MessagesSupported*
- *DPLPROPERTY_PlayerGuid*
- *DPLPROPERTY_PlayerScore*

DPLPROPERTY_LobbyGuid

A GUID used to identify the lobby software. If an application was designed to interoperate with a specific lobby using custom messages, this property can be used to identify the lobby.

However, an application should attempt to use standard messages whenever possible to be able to interoperate with as many lobbies as possible.

For this property, the *dwPropertyData* field of the **DPLMSG_GETPROPERTYRESPONSE** structure will contain the following:

```
GUID  guidLobby;
```

DPLPROPERTY_MessagesSupported

A boolean value indicating whether the lobby supports standard messages or not. An application can request this property from the lobby to determine if it supports standard messages. If the lobby does not respond to this message or if it responds with FALSE, then the application should assume that it does not support standard lobby messages and not send any further messages. If the lobby returns TRUE, then the application should freely send standard messages to the lobby and query for further properties of the lobby.

For this property, the *dwPropertyData* field of the **DPLMSG_GETPROPERTYRESPONSE** structure will contain the following:

```
BOOL  fMessagesSupported;
```

DPLPROPERTY_PlayerGuid

A GUID used to uniquely identify the application's local players. Any further **DPLMSG_SETPROPERTY** or **DPLMSG_GETPROPERTY** messages that require a player GUID will need to use this value to communicate with the lobby.

An application can manipulate properties for remote players on another computer. The application only needs to obtain the player GUID from the remote player (not from the lobby).

If an application supports spectator players, it should request this property before creating any players in order to determine what flags (if any) to use when creating the players; for example, DPPLAYER_SPECTATOR and DPPLAYER_SERVERPLAYER.

For this property, the *dwPropertyData* field of the **DPLMSG_GETPROPERTYRESPONSE** structure will contain the following:

```
typedef struct {
    GUID  guidPlayer;
    DWORD dwPlayerFlags;
} DPLDATA_PLAYERGUID, FAR *LPDPLDATA_PLAYERGUID;
```

DPLPROPERTY_PlayerScore

A generic structure that can be used by an application to report a composite score value to the lobby. The *guidPlayer* GUID of the player the score applies to must be provided when setting or getting this property. This structure can handle an arbitrary list of long integer values that collectively represent the score of a player. The application must allocate enough memory to hold all the scores. For example, if the score was passed as N integers, the application must allocate (sizeof(DPLDATA_PLAYERSCORE) + N*sizeof(LONG)) bytes and cast the pointer to LPDPLDATA_PLAYERSCORE.

For this property, the *dwPropertyData* field of the **DPLMSG_GETPROPERTYRESPONSE** structure will contain the following:

```
typedef struct {
    DWORD   dwScoreCount;
    LONG    Score[1];
} DPLDATA_PLAYERSCORE, FAR *LPDPLDATA_PLAYERSCORE;
```

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all *IDirectPlay3* and *IDirectPlayLobby2* 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 **IDirectPlayLobby2::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_AUTHENTICATIONFAILED

The password or credentials supplied could not be authenticated.

DPERR_BUFFERTOOLARGE

The data buffer is too large to store.

DPERR_BUSY

A message cannot be sent because the transmission medium is busy.

DPERR_BUFFERTOOSMALL

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

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_CANTLOADCAPI

No credentials were supplied and the CryptoAPI package (CAPI) to use for cryptography services cannot be loaded.

DPERR_CANTLOADSECURITYPACKAGE

The software security package cannot be loaded.

DPERR_CANTLOADSSPI

No credentials were supplied and the software security package (SSPI) that will prompt for credentials cannot be loaded.

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_CONNECTING

The method is in the process of connecting to the network. The application should keep calling the method until it returns DP_OK, indicating successful completion, or it returns a different error.

DPERR_ENCRYPTIONFAILED

The requested information could not be digitally encrypted. Encryption is used for message privacy. This error is only relevant in a secure session.

DPERR_EXCEPTION

An exception occurred when processing the request.

DPERR_GENERIC

An undefined error condition occurred.

DPERR_INVALIDCREDENTIALS

The credentials supplied (as to **IDirectPlay3::SecureOpen**) were not valid.

DPERR_INVALIDFLAGS

The flags passed to this method are invalid.

DPERR_INVALIDGROUP

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

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 method are invalid.

DPERR_INVALIDPASSWORD

An invalid password was supplied when attempting to join a session that requires a password.

DPERR_INVALIDPLAYER

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

DPERR_LOGONDENIED

The session could not be opened because credentials are required and either no credentials were supplied or the credentials were invalid.

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 in the receive queue.

DPERR_NONAMESERVERFOUND

No name server (host) could be found or created. A host must exist to create a player.

DPERR_NONEWPLAYERS

The session is not accepting any new players.

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 **IDirectPlayLobby2::Connect** method if the application was not started by using the **IDirectPlayLobby2::RunApplication** method or if there is no **DPLCONNECTION** structure currently initialized for this DirectPlayLobby object.

DPERR_NOTLOGGEDIN

An action cannot be performed because a player or client application is not logged in. Returned by the **IDirectPlay3::Send** method when the client application tries to send a secure message without being logged in.

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 being sent by the **IDirectPlay3::Send** method is too large.

DPERR_SESSIONLOST

The connection to the session has been lost.

DPERR_SIGNFAILED

The requested information could not be digitally signed. Digital signatures are used to establish the authenticity of messages.

DPERR_TIMEOUT

The operation could not be completed in the specified time.

DPERR_UNAVAILABLE

The requested function is not available at this time.

DPERR_UNINITIALIZED

The requested object has not been initialized.

DPERR_UNKNOWNAPPLICATION

An unknown application was specified.

DPERR_UNSUPPORTED

The function is not available in this implementation. Returned from **IDirectPlay3::GetGroupConnectionSettings** and **IDirectPlay3::SetGroupConnectionSettings** if they are called from a session that is not a lobby session.

DPERR_USERCANCEL

Can be returned in two ways. 1) The user canceled the connection process during a call to the **IDirectPlay3::Open** method. 2) The user clicked Cancel in one of the DirectPlay service provider dialog boxes during a call to **IDirectPlay3::EnumSessions**.