

Developing Multiplayer Add-Ons for Flight Simulator 2000

Flight Simulator 2000 provides support for peer-to-peer multiplayer gaming. Leveraging Microsoft's DirectPlay technology, we enable multiple users to share the same airspace over a modem, Local Area Network (LAN), and via a serial connection. It's relatively simple for end users to connect with others using this technology.

In addition, websites can take advantage of these multiplayer features by setting up Lobbies (using DirectPlay) where different users can meet on the Internet to join different Flight Simulator sessions. We make such a lobby available from within Flight Simulator 2000 itself—users can connect directly to an area on the MSN Gaming Zone where multiple users interact.

This document is aimed at developers who want to build add-on products for Flight Simulator specifically designed to take advantage of its multiplayer capabilities. For example, you could use this information to create an add-on application that provides Air Traffic Control (ATC) functionality to interact with multiple players on the Internet or on a local area network (LAN).

Audience

This document assumes you have the following expertise:

- Experience in developing applications using Visual C++ or some equivalent high-level language.
- Experience in, and understanding of, the features and capabilities of the Microsoft DirectPlay technology (from the DirectX 7 SDK). You'll find the DirectX 7 SDK and other relevant technical information on the Microsoft web site at <http://msdn.microsoft.com/directx>.

Note: The information contained in this document is **not** supported by Microsoft Product Support.

How Multiplayer Works

The architecture of the multiplayer system is based on information packets sent between the players in a session. Each information packet has a unique name and ID associated with it. Some packets contain extra information and have a data structure associated with them. Since this is a peer-to-peer system, all packets are sent to all players in the session. Flight Simulator 2000 supports up to 255 players in a session.

Players (not planes) are the core of the multiplayer system. In the system, players fall into three categories: unknown, players, and observers.

- The "unknown" player designation is a temporary designation used when a player first joins a session. A player is classified as unknown until the session host responds as to whether they can join the multiplayer session as a player or an observer.
- The "player" classification designates a player that has a visual presence on remote machines and can be collided with.
- The "observer" classification designates a player that watches other players, but has no visual presence on remote machines and can't collide with other players.

The only way to provide multiplayer functionality with Flight Simulator is through the multiplayer technology described in this document. Flight Simulator 2000 doesn't provide any other hooks for access to data relating to players in a session. Once you join a session through the multiplayer technology, your application can hook into the message stream to gain access to the remote player data.

See the discussion of Packet IDs and Packet Structures for complete lists of the packet IDs and packet structures.

Naming Convention

For the sake of brevity within this document, we've abbreviated references to Packet IDs and Packet structures as follows:

- Packet ID references don't include the **MULTIPLAYER_PACKET_ID_** prefix.
- Packet structure references don't include the **MULTIPLAYER_PACKET_** prefix.

Connecting an Add-On

Connecting an add-on to a multiplayer session accomplishes the following:

- Locates a multiplayer session to connect to.
- Connects the add-on to a multiplayer session and creates a DirectPlay player.
- Communicates with the session host to validate the player created by the add-on (as a player or an observer).
- Determines the characteristics of the other players in the multiplayer session.

To connect an add-on to a Flight Simulator multiplayer session

1. A Flight Simulator 2000 multiplayer session must exist. Start up Flight Simulator 2000 and initiate multiplayer. You can join either as a player (or as a host)—but make sure that there is room in the session for another player or observer.
2. Scan for available sessions using the DirectPlay **EnumSessions** function with the following application **GUID** specified in the **SESSIONDESC2** structure.

```
GUID_FS = { 0x1f0cb318, 0xf159, 0x432f,  
            { 0x8c, 0x38, 0x8c, 0xe7, 0x53, 0xa0,  
              0x3c, 0xda } };
```

3. Using the DirectPlay **Open** function, connect to the session you deem appropriate. You can either do this visually from Flight Simulator by examining the sessions listed, or programmatically by scanning the session information received via **EnumSessions**.
4. Broadcast a packet to all players in the chosen session using **ADD_PLAYER_REQUEST** or **ADD_OBSERVER_REQUEST**, depending on what type of add-on ("player" or "observer") you want to add to the session.
5. Wait for a response from the host player of the chosen session, such as **CHANGE_TO_PLAYER**, **CHANGE_TO_OBSERVER**, **ADD_PLAYER_REFUSED**, or **ADD_OBSERVER_REFUSED**.
6. If the host hasn't responded to a player's join request in a specified amount of time (30 seconds), send a **RETRANSMIT_JOIN_PLAYER** or **RETRANSMIT_JOIN_OBSERVER** packet and return to step 4.
7. If the response from the host is **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER**, check the player ID in the message data. If the player ID matches the player ID of the local player, the add-on is successfully connected to the session.
8. -Or-
9. If the response from the host is **ADD_PLAYER_REFUSED** or **ADD_OBSERVER_REFUSED**, the add-on wasn't able to connect to the session and needs to take appropriate action. For example, if the response from the host was **ADD_PLAYER_REFUSED**, the add-on could display a message informing

the user that they couldn't join as a player and ask them if they want to join as an observer instead.

Note: If the host doesn't allow the add-on to join the session, the add-on should end its connection to the session.

Determining Player Profiles

Once you've connected an add-on to the session, you need to determine the characteristics of each of the other players in the session. The add-on must determine whether remote machines represent a "player" or an "observer." The add-on must also determine whether remote players will be sending or receiving detailed plane information.

All multiplayer sessions have two DirectPlay groups, **FS61_REQUEST_PARAMS** and **FS61_REFUSE_PARAMS**.

- Remote machines that belong to the **FS61_REQUEST_PARAMS** group are interested in receiving detailed aircraft information;
- Remote machines that belong to the **FS61_REFUSE_PARAMS** group aren't interested in receiving detailed aircraft information.

Note: The grouping occurs when the user joins a multiplayer session. Flight Simulator provides the option to choose to receive (or not receive) detailed aircraft information via a checkbox in the multiplayer dialog.

To determine player profiles

1. Use the DirectPlay **EnumGroups** function to create a list of groups in the session. The two groups used by the multiplayer system are **FS61_REQUEST_PARAMS** and **FS61_REFUSE_PARAMS**. The add-on will need to make its player a member of one of the **FS61_REFUSE_PARAMS** group.
2. Using the group ID associated with the **FS61_REFUSE_PARAMS** group and the player ID of the add-on's player, call the DirectPlay function **AddPlayerToGroup**.
3. Use the DirectPlay **EnumPlayers** function to create a list of the other players in the session and mark each player as having an "unknown" player type.
4. Step through the list of players and send each player a **QUERY_PLAYER_TYPE** packet. Wait for a **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** message that identifies a player's type. (The **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** message can come from the session host as well as

from the unknown player.) Use the **player_id** field in the packet to determine which player's information should be updated.

5. Every 30 seconds or so, scan the known player list and send any "unknown" players a **QUERY_PLAYER_TYPE** packet.

It's possible for a player to join a session after the add-on creates a player. In that case, DirectPlay sends a message, **DPMSG_CREATEPLAYERORGROUP**.

When a new player notification is received

1. Add the new player to the current player list and tag that player as unknown.
2. Send the new player a **QUERY_PLAYER_TYPE** packet.
Wait for a **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** message that identifies the player's type. The **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** message can originate from the unknown player or the session host.
3. Use the **player_id** field in the packet to determine which player's information to update.

Constructing a Packet

The behavior of an add-on after it's connected to a session reflects the player type of the add-on.

- An "observer" type add-on monitors the packet stream and doesn't send packets unless prompted by a query, such as **QUERY_PLAYER_TYPE**.
- A "player" type add-on responds to packets and sends location updates (**POSITION_LLAPBH** or **POSITION_VELOCITY**) to all players in the session at the rate of four packets per second.

Both player and observer add-ons must monitor **POSITION_LLAPBH** and **POSITION_VELOCITY** packets to track other players.

All of the packets sent by the multiplayer system have a standard header; for more information, see the **MULTIPLAYER_PACKET_HEADER** structure in the discussion of Packet Structures.

- The first element in the structure (**packet_id**) is a 32-bit number that designates the ID of the packet;
- The second element in the structure (**data_size**) specifies the size of the data block that follows the header.

- The third element in the structure (**data[]**) is a placeholder that indicates where extra packet data will begin. It's possible that a packet will contain only a packet ID and size fields; the data size can be 0.

The following pages describe the packets encountered during the course of a normal session and identify what action (if any) the add-on should take when the packet is received.

Send-Only Packets

The following packets are those your add-on might typically send during a session. If your add-on receives any of these packets, it should be ignored.

ADD_OBSERVER_REQUEST

An add-on sends this packet to all players when joining the session as an observer. Send this packet with guaranteed messaging. Contains no extra data.

ADD_PLAYER_REQUEST

An add-on sends this packet to all players when joining the session as a player. Send this packet with guaranteed messaging. Contains no extra data.

CHAT_TEXT_SEND

Represents a chat message. The **chat_data** field is a null-terminated string containing the chat message. An add-on must include the null termination on the message when sending this packet.

Receive-Only Packets

The following packets could be received by your add-on during a session. Appropriate actions are noted where necessary. An add-on should never send any of these packets.

ADD_OBSERVER_REFUSED

Indicates that the add-on hasn't been successfully added to the session. The add-on should notify the user and end the DirectPlay connection.

ADD_PLAYER_REFUSED

Indicates that the add-on hasn't been successfully added to the session. The add-on should notify the user and ends the DirectPlay connection.

LEAVE_SESSION

Indicates that the host wants to remove the add-on from the session. The add-on responds by closing its connection to the session.

OBSERVER_CHANGE_OK

Indicates that the add-on has been changed from a "player" to an "observer" and will be recognized as such for the remainder of the session.

OBSERVER_CHANGE_REFUSED

Indicates that the add-on hasn't been allowed to change from a "player" to an "observer."

Packets Sent and Received

You may send or receive the following packets, depending on the nature of your add-on.

CHANGE_PLAYER_PLANE

Indicates that a player has changed to a new plane. The **engine_type** field contains one of the following values:

- ENGINE_TYPE_PISTON = 0
- ENGINE_TYPE_JET = 1
- ENGINE_TYPE_NONE = 2
- ENGINE_TYPE_HELO_TURBINE = 3

The **aircraft_name** field contains a null-terminated string that indicates the name of the aircraft.

An add-on shouldn't need to send this packet, but can do so to change the visual model of the plane that represents the add-on on other machines. When constructing this packet, ensure that the packet properly accounts for the null termination required at the end of the aircraft name.

CHANGE_TO_OBSERVER

When received by an add-on, the **player_id** field in the data structure contains a DirectPlay player ID. The player associated with this ID is identified as an "observer" for the remainder of the session.

If the player ID matches the ID of the player created by the add-on, the add-on has been successfully added to the session as an observer. The add-on doesn't send location update information.

An add-on must send this packet as a response to a **QUERY_PLAYER_TYPE** packet. This response is sent only if the add-on has successfully joined a session as an observer (see **QUERY_PLAYER_TYPE**).

CHANGE_TO_PLAYER

When received by an add-on, the **player_id** field in the data structure contains a DirectPlay player ID. The player associated with this ID is identified as a "player" for the remainder of the session.

If the player ID matches the ID of the player created by the add-on, the add-on has been successfully added to the session as a player. At this point, the add-on must send location update information with **POSITION__LLABPH** and/or **POSITION_VELOCITY**.

An add-on sends this packet in response to a **QUERY_PLAYER_TYPE** packet.

PLAYER_CRASH

Indicates that a remote player has registered a collision with the add-on. The add-on then processes the collision. An add-on sends this packet to another player (but not to an observer) only if it has registered a collision with that player.

POSITION_LLAPBH

Indicates the location of a remote player. Receiving this packet also implies that the remote player is in a paused state and their plane shouldn't be collided with for at least 10 seconds. Each time this packet is received from a player, the collision avoidance time should be reset to 10 seconds. The reduced (uses only the 10 most significant bits of pitch, bank, and heading) **LLAPBH** data contents of this packet are as follows:

- **application_time** Field is ignored.
- **packet_index** Contains an indexing ID that indicates the ordering of location packets. The index of the last received location packet is stored on a player-by-player basis. A new packet's contents is considered valid only if it has a higher packet index than the last packet index received from a player.
- **pbh** Contains a fractional representation of the pitch, bank, and heading of the remote plane. The pitch is found in the most significant 10 bits, bank is found in the second most significant 10 bits, and heading is found in the third most significant 10 bits. The least significant 2 bits aren't used.
- **lat_I** Contains the high-order 32 bits of latitude.
- **lon_hi** Contains the high-order 32 bits of longitude.
- **alt_I** Contains the high-order 32 bits of altitude.
- **lat_f** Contains the low-order 16 bits of latitude.
- **lon_lo** Contains the low-order 16 bits of longitude.
- **alt_f** Contains the low-order 16 bits of altitude.

An add-on could also send this packet to change its location in the world without colliding with other players.

POSITION_VELOCITY

Indicates the location of a remote player. The data contents of this packet are as follows:

- **packet_index** See POSITION_LLAPBH packet.
- **application_time** Field is ignored.
- **lat_velocity** Contains the latitude component of the plane's current velocity.
- **lon_velocity** Contains the longitude component of the plane's current velocity.
- **alt_velocity** Contains the altitude component of the plane's current velocity.
- **ground_velocity** Contains the net ground velocity of the remote plane.
- **reduced_llapbh** See POSITION_LLAPBH packet.

All velocities are 16-bit integer, 16-bit fractional values with units of feet/second.

An add-on could send this packet to change its location in the world in a manner that enables positional extrapolation—the packet contains location information and velocity vectors on which any change in position can be based.

QUERY_PLAYER_PLANE

Indicates that a remote player wants to find out the type of aircraft that the add-on is flying. The add-on can ignore this packet or send a **CHANGE_PLAYER_PLANE** response indicating the aircraft that will represent the add-on when it has successfully joined as a player.

An add-on sends this packet only to find out the type of aircraft that a remote player is flying.

QUERY_PLAYER_TYPE

Indicates that the remote machine doesn't know the player type ("player" vs. "observer") of an add-on. An add-on should only respond to this packet after it has received a confirmation **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** packet. At this point, the add-on responds to the **QUERY_PLAYER_TYPE** packet with either **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER**, as appropriate.

An add-on sends this packet to a new player after receiving notification from DirectPlay that a new player has been created. If a player has joined a game, but a **CHANGE_TO_PLAYER** or **CHANGE_TO_OBSERVER** message hasn't arrived, a **QUERY_PLAYER_TYPE** message is sent to that player every 30 seconds until a confirmation arrives.

RETRANSMIT_JOIN_PLAYER

An add-on sends this packet to all players if it has sent an **ADD_PLAYER_REQUEST**, but has not received a **CHANGE_TO_PLAYER** or an **ADD_PLAYER_REFUSED** response within 30 seconds. The add-on continues to broadcast this packet every 30 seconds until it receives either a **CHANGE_TO_PLAYER** or an **ADD_PLAYER_REFUSED** packet.

If an add-on receives this packet, it should be ignored.

RETRANSMIT_JOIN_OBSERVER

An add-on sends this packet to all players if it has sent an **ADD_OBSERVER_REQUEST**, but hasn't received a **CHANGE_TO_OBSERVER** or an **ADD_OBSERVER_REFUSED** response within 30 seconds. The add-on continues to broadcast this packet every 30 seconds until it receives either a **CHANGE_TO_OBSERVER** or an **ADD_OBSERVER_REFUSED** packet.

If an add-on receives this packet, it should be ignored.

REQUEST_OBSERVER

An add-on sends this packet to all players to change its player type from "player" to "observer" during a session. If the add-on doesn't receive either an **OBSERVER_CHANGE_OK** or an **OBSERVER_CHANGE_REFUSED** packet within 30 seconds, it sends a **RETRANSMIT_GO_OBSERVER** packet. An add-on can change from a player to an observer, but not from an observer to a player.

If an add-on receives this packet, it should be ignored.

RETRANSMIT_GO_OBSERVER

An add-on sends this packet to all players at 30 second intervals if it has sent a **REQUEST_OBSERVER** packet, but hasn't received an **OBSERVER_CHANGE_OK** or an **OBSERVER_CHANGE_REFUSED** response.

If an add-on receives this packet, it should be ignored.

Non-Relevant Packets

Any add-on you build should never send any of the following packets. If your add-on ever receives one of these packets, just ignore it.

- AIRCRAFT_CANCEL
- AIRCRAFT_SEGMENT
- HOST_QUIT
- INITIALIZE_AIRCRAFT_SEND
- INITIALIZE_TEXTURE_SEND
- PARAMS
- REMOTE_PLANE_UNKNOWN
- REQUEST_SITUATION
- SITUATION_DATA
- SYNC_INFORMATION
- TEXTURE_CANCEL
- TEXTURE_REQUEST_LIST
- TEXTURE_SEGMENT

Packet IDs

The following code includes the packet IDs sent by the multiplayer system. You must use the packet IDs shown in this example to determine packet contents.

```
// The following defines are the IDs used as tags on the packets sent by  
the multiplayer system.
```

```
Typedef enum  
{  
  
MULTIPLAYER_PACKET_ID_BASE = 0x1000,  
  
MULTIPLAYER_PACKET_ID_PARAMS = MULTIPLAYER_PACKET_ID_BASE,  
  
MULTIPLAYER_PACKET_ID_ADD_PLAYER_REQUEST,  
  
MULTIPLAYER_PACKET_ID_ADD_OBSERVER_REQUEST,  
  
MULTIPLAYER_PACKET_ID_CHANGE_TO_PLAYER,  
  
MULTIPLAYER_PACKET_ID_CHANGE_TO_OBSERVER,  
  
MULTIPLAYER_PACKET_ID_ADD_PLAYER_REFUSED,  
  
MULTIPLAYER_PACKET_ID_ADD_OBSERVER_REFUSED,  
  
MULTIPLAYER_PACKET_ID_QUERY_PLAYER_TYPE,  
  
MULTIPLAYER_PACKET_ID_RETRANSMIT_JOIN_PLAYER,  
  
MULTIPLAYER_PACKET_ID_RETRANSMIT_JOIN_OBSERVER,  
  
MULTIPLAYER_PACKET_ID_REQUEST_OBSERVER,  
  
MULTIPLAYER_PACKET_ID_RETRANSMIT_GO_OBSERVER,  
  
MULTIPLAYER_PACKET_ID_OBSERVER_CHANGE_OK,  
  
MULTIPLAYER_PACKET_ID_OBSERVER_CHANGE_REFUSED,  
  
MULTIPLAYER_PACKET_ID_CHANGE_PLAYER_PLANE,  
  
MULTIPLAYER_PACKET_ID_QUERY_PLAYER_PLANE,  
  
MULTIPLAYER_PACKET_ID_REMOTE_PLANE_UNKNOWN,
```

```

MULTIPLAYER_PACKET_ID_PLAYER_CRASH,

MULTIPLAYER_PACKET_ID_HOST_QUIT,

MULTIPLAYER_PACKET_ID_REQUEST_SITUATION,

MULTIPLAYER_PACKET_ID_SITUATION_DATA,

MULTIPLAYER_PACKET_ID_TEXTURE_REQUEST_LIST,

MULTIPLAYER_PACKET_ID_LEAVE_SESSION,

MULTIPLAYER_PACKET_ID_SYNC_INFORMATION,

MULTIPLAYER_PACKET_ID_POSITION_LLAPBH,

MULTIPLAYER_PACKET_ID_POSITION_VELOCITY,

MULTIPLAYER_PACKET_ID_INITIALIZE_AIRCRAFT_SEND,

MULTIPLAYER_PACKET_ID_AIRCRAFT_SEGMENT,

MULTIPLAYER_PACKET_ID_AIRCRAFT_CANCEL,

MULTIPLAYER_PACKET_ID_INITIALIZE_TEXTURE_SEND,

MULTIPLAYER_PACKET_ID_TEXTURE_SEGMENT,

MULTIPLAYER_PACKET_ID_TEXTURE_CANCEL,

MULTIPLAYER_PACKET_CHAT_TEXT_SEND,

} MULTIPLAYER_PACKET_ID;

```

Packet Structures

The following code includes the packet structures you must use in the multiplayer system.

// You must use the following structure as a header for all multiplayer packets to be sent.

```

typedef struct MULTIPLAYER_PACKET_HEADER
{

MULTIPLAYER_PACKET_ID packet_id;

UINT32 data_size;

```

```

VAR8 data[];

} MULTIPLAYER_PACKET_HEADER;

// Use the following structure to hold lat, lon, alt, pbh data.

Typedef struct REDUCED_LLAPBH_DATA
{
VAR32 pbh;

SINT32 lat_i;

SINT32 lon_hi;

SINT32 alt_i;

UINT16 lat_f;

UINT16 lon_lo;

UINT16 alt_f;

} REDUCED_LLAPBH_DATA;

// Template for changing to player packet.

Typedef struct MULTIPLAYER_PACKET_CHANGE_TO_PLAYER
{
DPID player_id;

} MULTIPLAYER_PACKET_CHANGE_TO_PLAYER;

// Template for changing to observer packet.

Typedef struct MULTIPLAYER_PACKET_CHANGE_TO_OBSERVER
{
DPID player_id;

} MULTIPLAYER_PACKET_CHANGE_TO_OBSERVER;

// Template for the "player changed planes" packet.

typedef struct MULTIPLAYER_PACKET_CHANGE_PLAYER_PLANE
{

```

```

ENUM engine_type;

STRINGZ aircraft_name[];

} MULTIPLAYER_PACKET_CHANGE_PLAYER_PLANE;

// Template for the basic position packet.

Typedef struct MULTIPLAYER_PACKET_POSITION_LLAPBH
{
    UINT32 application_time;

    UINT32 packet_index;

    REDUCED_LLAPBH_DATA reduced_llapbh;

} MULTIPLAYER_PACKET_POSITION_LLAPBH;

// Template for the plane's position and velocities.

Typedef struct MULTIPLAYER_PACKET_POSITION_VELOCITY
{
    UINT32 packet_index;

    UINT32 application_time;

    SIF321 at_velocity;

    SIF321 on_velocity;

    SIF32 alt_velocity;

    UIF32 ground_velocity;

    REDUCED_LLAPBH_DATA reduced_llapbh;

} MULTIPLAYER_PACKET_POSITION_VELOCITY;

// Template for the chat message data packet.

Typedef struct MULTIPLAYER_PACKET_CHAT_TEXT
{
    STRINGZ chat_data;

} MULTIPLAYER_PACKET_CHAT_TEXT;

```