Session Interface

System for setting up advertisement of matches and handling matchmaking of players

Matchmaking is the process of matching players with sessions. A session is basically an instance of the game running on the server with a given set of properties which is either advertised so that it can be found and joined by players wanting to play the game or private so only players who are invited or notified of it in some way can join.

Picture an online game lobby that lists all of the games currently being played. Each game in the list is a session, or individual online match. Players are matched with sessions either by searching or some other means and then join the session to play the match.

The basic lifetime of a session is:

- Create a new session with desired settings
- Wait for players to request to join the match
- Register players who want to join
- Start the Session
- Play the match
- End Session
- Unregister the players
- Either:

- Update the Session if you want to change the type of match and go back to waiting for players to join
- Destroy the Session

Session Interface

The **Session Interface**, `IOnlineSession`, provides platform-specific functionality for setting up the pieces behind the scenes that are necessary in order to perform matchmaking as well as other methods of allowing players to find and join online games. This includes session management, finding sessions though search or other means, as well as joining and leaving those sessions. The session interface is created and owned by the OnlineSubsystem. This means it only exists on the server.

There is one session interface class per platform. When adding support for a new platform, a new type of session interface must be created. Platforms, in this sense, refer to hardware platforms. As such, only one session interface will ever exist at a time - the session interface for the platform the engine is currently running on.

While the session interface performs all of the session handling, the game doesn't generally interact directly with it. Instead, the **Game Session**, `AGameSession`, acts as a game-specific wrapper around the session interface and the game code makes calls to it when it needs to interact with the session. The game session is created and owned by the game mode, `AGameModeBase`, and also only exists on the server when running an online game.

Each game can potentially have multiple game session types, but only one will ever be used at a time. The most common case for a game having more than one type of game session is to add a type of game session for when the game is using a dedicated server.

Session Settings

The **Session Settings**, defined by the `FOnlineSessionSettings` class, are a set of properties that determine the characteristics of the session. In the base implementation, these are things like:

- Number of players allowed
- Is the session advertised or private

- Is the session a LAN match
- Is the server dedicated or player-hosted
- Are invites allowed
- Etc.

Using the online game lobby example, each of those games is a session and has its own session settings. For example, some sessions may be player versus player (PvP) while others are cooperative multiplayer (Co-Op). Different sessions could be playing different maps or playlists, or require different numbers of players, etc.

Session Management

One of the main duties of the session interface is session management, including session setup, updating, and destruction.

Creating Sessions

In order for players to find a session and potentially join it, you need to create a session and set up its properties, as well as decide which of those properties will be visible so they can be searched.

Sessions are created using `IOnlineSession::CreateSession()` which takes in a set of session settings that are used to configure the new session. Once the session is created, the `OnCreateSessionComplete` delegate is fired.

Updating Sessions

Updating a session is done when you want to change the settings of an existing session and is performed using the `IOnlineSession::UpdateSession()` function. For example, the session may be set up currently to only allow 8 players, while it needs to allow 12 players for the next match. To update the session, `UpdateSession()` would be called, passing it new session settings that specify a 12 player maximum.

When the request to update a session has completed, the `OnUpdateSessionComplete` delegate is fired. This provides an opportunity to perform any configuration or initialization

necessary to handle the session settings changing.

Updating a session is generally performed between matches on the server, but it is also performed on the client in order to keep the session information in sync.

Destroying Sessions

When a session ends and is no longer needed, the session is destroyed using the `IOnlineSession::DestroySession()` function. When the destruction operation has completed, the `OnDestroySessionComplete` delegate is fired enabling you to perform cleanup operations.

Matchmaking - Finding sessions

The Online Subsystem provides the basic building blocks necessary to match players up to active sessions. It does not provide any specific type of matchmaking built-in with the base implementation. However, implementations on platforms that provide matchmaking services do expose access to those services. This is essentially the process of finding a session for the player to join. Once that session is found, the player can join the session.

Searching Sessions

The simplest way to find sessions is to search for sessions that match some desired subset of settings. This could be in response to the player choosing a collection of filters in a user interface, or it could be done automatically behind the scenes based on the player's skill and other factors, or it could be a combination of both methods.

The most basic form of searching for sessions is the classic server browser that shows all available games and allows the player to filter those based on the type of game they want to play and then choose one to join one of the sessions that match those criteria.

To search for sessions, use `IOnlineSession::FindSessions()` and pass it a reference to the session settings to search for as an `FOnlineSessionSearch` object. The `SearchResults` member of the session settings reference is populated with the matching sessions. When the

search is completed, the `OnFindSessionsComplete` delegate is fired. From this delegate, you can iterate through the search results.

Cloud-Based Matchmaking

Cloud-based matchmaking refers to built-in matchmaking services that are available, and are generally platform-specific. An example of this type of service is the TrueSkill system available through Microsoft's Xbox Live service.

To initiate matchmaking on platforms that support it, you call `IOnlineSession::Startmatchmaking()` and pass it the controller number of the player to matchmake for, the session name, session settings to use if creating a new session, and settings to search against. The `OnMatchmakingComplete` delegate is fired when matchmaking is complete. This provides a bool specifying whether the process was successful and the name of the session to join in that case.

A matchmaking action in process can be canceled by calling `IOnlineSession::CancelMatchmaking()` and passing it the controller number of the player and the session name that was passed to the call to start matchmaking in the first place. The `OnCancelMatchmakingComplete` delegate is fired when the cancel operation is completed.

Following and Inviting Friends

On platforms that support the concept of friends, players can follow friends into a session or invite friends to join them in a session.

Following a friend into a session is done by calling `IOnlineSession:FindFriendSession()` and passing it the number of the local player that wants to join the session and the ID of the friend already in the session. The `OnFindFriendSessionComplete` delegate is fired when the session is found and it contains a search result that can be used to join the session.

A player can also invite one or more friends to join them in their current session using `IOnlineSession:SendSessionInviteToFriend()` or `IOnlineSession::SendSessionInviteToFriends()` and passing in the local player number, session name, and ID(s) of the player(s) to be invited. When a friend accepts an invitation, the `OnSessionUserInviteAccepted` delegate containing the search result of the session to join is fired.

Joining Sessions

Once you have determined a session for the player to join, the process of joining is initiated by calling `IOnlineSession::JoinSession()` and passing it the number of the player and the name and search result of the session to join. When the joining process completes, the `OnJoinSessionComplete` delegate is fired. This is where the logic to get the player into the match is performed. First, you need to call `IOnlineSession::GetResolvedConnectString()` which returns the platform specific connection information needed to join the match. The string obtained from this function can then be passed to `APlayerController::ClientTravel()` or `UWorld::Servertravel()` to send the player into the match.