Sessions Interface

Create and manage online game sessions.

Learn to use this **Beta** feature, but use caution when shipping with it.

The **Online Services Session Interface** handles the creation, destruction, and management of online game sessions. A **session** is the representation of an online match in a game, running on either a player's machine or a dedicated server. Sessions can have the following join policies:

- **Invite Only**: Players with an invitation can join the session.
- **Friends Only**: Friends of any session member can join the session.
- **Public**: Anyone can find and join the session.

You can define a public session using a set of properties that act as filters, allowing players to search for specific game modes or maps.

API Overview

The following table provides a high-level description of the functions included in the Sessions Interface.

Function	Description
Get Sessions	
`GetAllSessions`	Retrieve an array of references to all sessions the user is part of.
`GetSessionByName`	Retrieve a reference to the session with the supplied name.
`GetSessionById`	Retrieve a reference to the session with the supplied ID handle.
Presence	
`GetPresenceSession`	Retrieve a reference to the session currently set as the user's presence session.
`IsPresenceSession`	Determine whether the session with the given ID is set as the user's presence session.
`SetPresenceSession`	Set the session with the given ID as the user's presence session.
`ClearPresenceSession`	Clear the user's presence session.
Session Management	
`CreateSession`	Create a new session using the supplied parameters.
`UpdateSessionSettings`	Update settings for the session identified by the supplied name.
`LeaveSession`	Leave and optionally destroy the session identified by the supplied name.

Function	Description
`FindSessions`	Query the session service for sessions matching the supplied parameters.
`JoinSession`	Join the session with the supplied session ID.
`StartMatchmaking`	Start the matchmaking process. This searches for and joins a session that matches the given search filters, or, if no such session is found, creates a session using the supplied parameters.
`AddSessionMember`	Add the user as a new session member to the session identified by the supplied name.
`RemoveSessionMember`	Remove the user from the session identified by the supplied name.
Invites	
`SendSessionInvite`	Send an invite to the session identified by the supplied name to all supplied users.
`GetSessionInviteById`	Get a reference to the session invite identified by the supplied invite ID.
`GetAllSessionInvites`	Get an array of references to all the session invites the user has received.
`RejectSessionInvite`	Reject the session invite identified by the supplied invite ID.
Event Listening	Event will trigger as a result of:
`OnSessionJoined`	Joining a session.

Function	Description
`OnSessionLeft`	Leaving or destroying a session.
`OnSessionUpdated`	Updating a session's settings or whenever a session update event is received.
`OnSessionInviteReceived`	Receiving a session invite.
`OnUISessionJoinRequested`	Accepting a session invite or joining a session through the platform UI.

Process Flow

Session Lifecycle

- Create a new session using the desired settings.
- At any point during the lifetime of a session, you can update the session to reflect changes in the properties of the online match it represents. These changes can include:
  - Altering parameters regarding how the session appears in searches, or whether it shows up at all.
  - Restricting new players from joining the session once the game is in progress.
- Players that discover the session can join it.
- Using information obtained by joining the session, new players can connect to the session host or dedicated server.
  - After connecting, the players need to get registered with the session. This process will be automatically handled by the engine in future releases.
- Play the game.
- When the game is over, the player can leave the session or destroy it (if the player is the owner or host).
- Disconnecting from the host or server needs to be followed by unregistering the player or players from the session. This process will be automatically handled by the engine in future releases.

Create

The first step in the session lifecycle is creating a session using the desired parameters. These parameters include some that remain constant throughout the lifetime of the session (like `bIsLANSession` and `bAllowSanctionedPlayers` from the `CreateSession` function) and some you can update at any time (like the options provided by the function `SessionSettings`).

At most one session per user can be set as the **Presence Session**. This means it will appear in the user's Presence information and be visible to friends and followers as exposed through the Presence Interface. If a user is a member of many sessions, which session appears as the presence session can be changed by `SetPresenceSession` (this functionality might not be available in all platform implementations).

Discover

A user can discover new sessions in a few different ways:

Search

`FindSessions` allows users to define search parameters such as tags that match a desired session's custom settings or a specific user ID to find the session that their friend is in. This returns a list of session IDs each representing cached session information, which the user can search and access with `GetSessionById`.

Invitation

Users can receive session invitations from other users. After receiving an invitation, a user can view the information about the session by accessing the invitation with `GetSessionInviteById`. Afterwards, the user decides whether to join the session with the session ID provided by the invite information.

Presence

Specific platform UIs may show users information about sessions their friends have joined.

Join

Once a user has information about a session obtained either through search, invitation, or presence, they can attempt to join it by calling `JoinSession`. They can also choose if they want to set this new session as their presence session with `SetPresenceSession`.

Matchmaking

Another way you can join a session is by calling `StartMatchmaking`. This function acts as a combination of `CreateSession` and `FindSessions`. `StartMatchmaking` looks for sessions matching a predefined set of search filters and it will create a session using the given information if it cannot find any.

Once you have joined a session, you can call `IOnlineServices::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. If the travel succeeds, the player will be added to the session and `AddSessionMember` is called to register the player with the session.

Inviting

After joining a session, either by creating it or joining it, you can send the session information to other players with `SendSessionInvite`. This is a good way to get friends together in the same online match. Once a player receives an invite, they can access its information using `GetAllSessionInvites` to access all invites for a given user, or `GetSessionInviteById` to get the information about a particular invite. They can also reject session invites using a call to `RejectSessionInvite` by passing the corresponding invite ID as a parameter.

Update

You can update session settings at any point during the session's lifetime by calling `UpdateSessionSettings`. These settings include, but are not limited to:

- Maximum number of players in session
- Join policy for the session:
  - Invite Only

    - Friends Only
    - Public
- Restrict access to new players
- Add, modify, or remove custom settings and user-defined parameters

Leave and Destroy

You can leave a session by calling `LeaveSession`. The owner of the session can set the additional parameter `bDestroySession` to `true` if they want to remove the session from the backend services as they leave. This forces all other members of the session to leave as well.

Examples

You can access the Sessions Interface through a reference to an OnlineServices instance. From here, the functionality of the Sessions Interface is exposed. We provide a few examples of accessing the Sessions Interface and performing synchronous and asynchronous operations.

Get Session By Name

```
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::ISessionsPtr SessionsInterface = OnlineServices->GetSessionsInterface();
UE::Online::FGetSessionByName::Params Params;
Params.SessionName = FName(TEXT("MySession"));

UE::Online::TOnlineResult<UE::Online::FGetSessionByName> Result = SessionsInterface->GetSessionByName(MoveTemp(Params));
if(Result.IsOk())
{
TSharedRef<const UE::Online::ISession> Session = Result.GetOkValue().Session;
// now we can read information from the session
}
```

Copy full snippet

Walkthrough

1. Use the default online services by calling `GetServices` with no parameters specified:

```
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices
```

Copy full snippet

2. Access the Sessions interface for the default online services:

```
UE::Online::ISessionsPtr SessionsInterface = OnlineServices->GetSession
```

Copy full snippet

3. Initialize a `FGetSessionByName` struct using the necessary parameters to call `GetSessionByName`:

```
UE::Online::FGetSessionByName::Params Params;
Params.SessionName = FName(TEXT("MySession"));
```

Copy full snippet

4. Call `GetSessionByName` passing in the parameters from the previous step and save the result:

```
UE::Online::TOnlineResult<UE::Online::FGetSessionByName> Result = Sessi
```

Copy full snippet

5. Process the result of the call to `GetSessionByName` after ensuring the function call did not throw an error and the result can be accessed:

```cpp
if(Result.IsOk())
{
TSharedRef<const UE::Online::ISession> Session = Result.GetOkValue().Session;
// now we can read information from the session
}
```

Copy full snippet

Update Session Settings

```cpp
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::ISessionsPtr SessionsInterface = OnlineServices->GetSessionsInterface();

UE::Online::FUpdateSessionSettings::Params Params;
Params.LocalAccountId = AccountId;
Params.SessionName = FName(TEXT("MySession"));
Params.Mutations.bAllowNewMembers = false;

SessionsInterface->UpdateSessionSettings(MoveTemp(Params))
.OnComplete([this](const UE::Online::TOnlineResult<UE::Online::FUpdateSessionSettings>& Result)
{
if(Result.IsError())
{
const UE::Online::FOnlineError OnlineError = Result.GetErrorValue();
// update was not successful, process OnlineError
return;
}
// update was successful
});
```

Copy full snippet

Walkthrough

1. Use the default online services by calling `GetServices` with no parameters specified and access the Sessions interface for the default online services:

```
UE::Online::IOnlineServicesPtr OnlineServices =
UE::Online::GetServices();
UE::Online::ISessionsPtr SessionsInterface = OnlineServices-
>GetSessionsInterface();
```

Copy full snippet

2. Initialize a struct using the necessary parameters to call `UpdateSessionSettings`:

```
UE::Online::FUpdateSessionSettings::Params Params;
Params.LocalAccountId = AccountId;
Params.SessionName = FName(TEXT("MySession"));
Params.Mutations.bAllowNewMembers = false;
```

Copy full snippet

3. Handle the `UpdateSessionSettings.OnComplete` callback by processing the error or the queried stats if it resulted in an error, or the result if it returns okay with a lambda function:

```
SessionsInterface->UpdateSessionSettings(MoveTemp(Params))
.OnComplete([this](const
UE::Online::TOnlineResult<UE::Online::FUpdateSessionSettings>& Result)
{
if(Result.IsError())
{
const UE::Online::FOnlineError OnlineError = Result.GetErrorValue();
// update was not successful, process OnlineError
return;
}
// update was successful
});
```

Copy full snippet

Converting Code from Online Subsystem

The Online Services Sessions Interface is responsible for all code owned by the Online Subsystem Sessions Interface.

More Information

Header File

Consult the `Sessions.h` header file directly for more information as needed. The Sessions Interface header file `Sessions.h` is located in the directory:

```
Engine\Plugins\Online\OnlineServices\Source\OnlineServicesInterface\Public\Onlin
```

Copy full snippet

For instructions on how to obtain the UE source code, refer to our documentation on Downloading Unreal Engine Source Code.

Function Parameters and Return Types

Refer to the Functions section of the Online Services Overview page for an explanation of function parameters and return types, including how to pass parameters and processing the results when functions return.