Online Services Presence Interface

Access the presence and joinability status of friends and followers.

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

When logged into an online service, you may want to look for information about your friends and other users you have met online. For example, on many online services, you can see whether other users are online, what game they are currently playing, if they are available to join matches, and so on. The **Online Services Presence Interface** encompasses all functionality related to platform-specific user states across online services, including querying and updating a user's presence as well as listening for changes.

This document provides an API overview and code examples as well as tips for converting code from the Online Subsystem Presence Interface.

API Overview

Functions

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

Function	Description
Query	
`QueryPresence`	Fetch the presence of the user with the supplied `TargetAccountId`.
`BatchQueryPresence`	Fetch the presence for every user in the supplied list of `TargetAccountIds`.
Get	
`GetCachedPresence`	Retrieve the presence of the user with the supplied `TargetAccountId` cached in the interface.
Update	
`UpdatePresence`	Update the presence of the user.
`PartialUpdatePresence`	Update the presence of the user with only the specified presence settings.
Event Listening	
`OnPresenceUpdated`	Event will trigger as a result of updates to a user's presence.

Enumeration Classes

The Presence Interface defines three enumeration classes that represent a user's status (`EUserPresenceStatus`), joinability (`EUserPresenceJoinability`), and game status (`EUserPresenceGameStatus`). These enumeration classes represent three primary members of the `FUserPresence` struct. For more information, refer to the Primary Struct section of this page.

EUserPresenceStatus

Enumerator	Description
`Offline`	User is offline.
`Online`	User is online.
`Away`	User is away.
`ExtendedAway`	User has been away for at least two hours (may be platform dependent).
`DoNotDisturb`	User does not want to be disturbed.
`Unknown`	Default user presence status.

EUserPresenceJoinability

Enumerator	Description
`Public`	Anyone can discover and join this session.
`FriendsOnly`	Anyone trying to join must be a friend of a lobby member.
`InviteOnly`	Anyone trying to join must be invited first.
`Private`	User is not currently accepting invitations.
`Unknown`	Default user presence joinability status.

EUserPresenceGameStatus

Enumerator	Description
`PlayingThisGame`	User is playing the same game as you.
`PlayingOtherGame`	User is playing a different game than you.
`Unknown`	Default user presence game status.

Primary Struct

FUserPresence

The `FUserPresence` struct is the primary object in the Presence interface and consists of all necessary information pertaining to a user's presence.

Member	Type	Description
`AccountId`	`FAccountId`	User whose presence this is.
`Status`	`EUserPresenceStatus`	User presence state. (Default value is `EUserPresenceStatus::Unknown`.)
`Joinability`	`EUserPresenceJoinability`	User session state. (Default value is `EUserPresenceJoinability::Unknown`.)
`GameStatus`	`EUserPresenceGameStatus`	User game state. (Default value is `EUserPresenceGameStatus::Unknown`.)
`StatusString`	`FString`	String representation of user presence state.

Member	Type	Description
`RichPresenceString`	`FString`	Game-defined representation of the current game state.
`Properties`	`FPresenceProperties`	Session keys.

The type `FPresenceProperties` is a typedef for `TMap<FString, FPresenceVariant>` where `FPresenceVariant` is an `FString`.

Examples

We now provide an example demonstrating `UpdatePresence`, `QueryPresence`, and `GetPresence`. `UserA` updates their presence with the default platform services, then `UserB` queries the presence of `UserA` after it has been updated. If the query successfully returns, then `UserB` retrieves the presence of `UserA`.

Code

UserAPresence.cpp

```
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::IPresencePtr PresenceInterface = OnlineServices->GetPresenceInterface();

TSharedRef<UE::Online::FUserPresence> Presence = MakeShared<UE::Online::FUserPresence>();
Presence->AccountId = UserA;
Presence->Status = UE::Online::EUserPresenceStatus::Online;
Presence->Joinability = UE::Online::EUserPresenceJoinability::Public;
Presence->RichPresenceString = TEXT("Exploring the Great Citadel");
Presence->Properties.Add(TEXT("advanced_class"), TEXT("advanced_class_assassin"));

```

```cpp
UE::Online::FUpdatePresence::Params Params;
Params.LocalAccountId = AccountId;
Params.Presence = Presence;

PresenceInterface->UpdatePresence(MoveTemp(Params))
.OnComplete([](const UE::Online::TOnlineResult<UE::Online::FUpdatePresence> Result)
{
if(Result.IsOk())
{
// we succeeded - UserB is now clear to query presence
}
else
{
// we failed - check the error state in Result.GetErrorValue();
}
});
```

Copy full snippet

UserBPresence.cpp

```cpp
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::IPresencePtr PresenceInterface = OnlineServices->GetPresenceInterface();

PresenceInterface->QueryPresence({UserA})
.OnComplete([](const UE::Online::TOnlineResult<UE::Online::FQueryPresence> Result)
{
if(Result.IsOk())
{
// we succeeded - now use GetPresence to actually view the presence object

UE::Online::TOnlineResult<UE::Online::FGetPresence> GetPresenceResult = PresenceInterface->GetPresence({UserB});
if(GetPresenceResult.IsOk())
{
TSharedRef<const UE::Online::FUserPresence> Presence = GetPresenceResult.GetOkValue().Presence;

// Presence->RichPresenceString will now be "Exploring the Great Citadel"
// Presence->Properties will now contain {advanced_class:
advanced_class_assassin}
// and so on...
}
else
{
// we failed - check error state with GetPresenceResult.GetErrorValue();
}

}
else
{
// we failed - check the error state in Result.GetErrorValue();
}
});
```

Copy full snippet

Walkthrough

1. Both users retrieve the default online services by calling `GetServices` with no parameters specified and access the Presence Interface:

   UserAPresence.cpp and UserBPresence.cpp

   ```
   UE::Online::IOnlineServicesPtr OnlineServices =
   UE::Online::GetServices();
   UE::Online::IPresencePtr PresenceInterface = OnlineServices-
   >GetPresenceInterface();
   ```

   Copy full snippet

2. `UserA` initializes an `FUserPresence` struct named `Presence`. Notice that we are using two of the aforementioned enumerations provided by the Presence Interface: `EUserPresenceStatus` and `EUserPresenceJoinability`.

   UserAPresence.cpp

```cpp
TSharedRef<UE::Online::FUserPresence> Presence =
MakeShared<UE::Online::FUserPresence>();
Presence->AccountId = UserA;
Presence->Status = UE::Online::EUserPresenceStatus::Online;
Presence->Joinability = UE::Online::EUserPresenceJoinability::Public;
Presence->RichPresenceString = TEXT("Exploring the Great Citadel");
Presence->Properties.Add(TEXT("advanced_class"),
TEXT("advanced_class_assassin"));
```

Copy full snippet

3. `UserA` initializes an `FUpdatePresence::Params` struct named `Params` with the parameters that will be passed to `UpdatePresence`:

   UserAPresence.cpp

```cpp
UE::Online::FUpdatePresence::Params Params;
Params.LocalAccountId = AccountId;
Params.Presence = Presence;
```

Copy full snippet

4. `UserA` calls `UpdatePresence` and processes the result with an `OnComplete` callback:

   UserAPresence.cpp

```cpp
PresenceInterface->UpdatePresence(MoveTemp(Params))
.OnComplete([](const
UE::Online::TOnlineResult<UE::Online::FUpdatePresence> Result)
{
if(Result.IsOk())
{
// we succeeded - UserB is now clear to query presence
}
else
{
// we failed - check the error state in Result.GetErrorValue();
}
});
```

Copy full snippet

5. `UserB` queries the presence of `UserA`. Inside the queries' `OnComplete` callback, `UserB` first checks to ensure `QueryPresence` returned an "Ok" status. If it did, then `UserB` is safe to retrieve the presence of `UserA` and process the result or error of `GetPresence` accordingly:

   UserBPresence.cpp

```cpp
PresenceInterface->QueryPresence({UserA})
.OnComplete([](const UE::Online::TOnlineResult<UE::Online::FQueryPresence> Result)
{
if(Result.IsOk())
{
// we succeeded - now use GetPresence to actually view the presence object

UE::Online::TOnlineResult<UE::Online::FGetPresence> GetPresenceResult = PresenceInterface->GetPresence({UserB});
if(GetPresenceResult.IsOk())
{
// we succeeded!
}
else
{
// we failed - check error state with GetPresenceResult.GetErrorValue();
}

}
else
{
// we failed - check the error state in Result.GetErrorValue();
}
});
```

   Copy full snippet

If all function calls return without error, `UserB` now sees the updated status of `UserA` and `UserB` can choose to make decisions based on this status. For example, `UserB` could access the `GetPresenceResult` to see `UserA` is online and their joinability status is public. Upon setting this status `UserB` could decide to join `UserA` and "Explore the Great Citadel" together.

Converting Code from Online Subsystem

The Online Services plugins are an updated version of the Online Subsystem plugins and will exist alongside one another for the foreseeable future. The API functionality of the Online Services Presence Interface maps approximately one-to-one with the API functionality of the Online Subsystem Presence Interface. A few caveats include:

- `SetPresence` was renamed to `UpdatePresence` to better represent the function's asynchronicity.
- `UpdatePresence` and `QueryPresence` are no longer overloaded.
- We recommend using their renamed functions `PartialUpdatePresence` and `BatchQueryPresence` instead.
  - The overloads for `UpdatePresence` and `QueryPresence` were renamed to `PartialUpdatePresence` and `BatchQueryPresence`, respectively.
- `QueryPresence` was given the `bListenToChanges` parameter.
  - This adds a specific user to the `OnPresenceUpdated` event.
  - The parameter is set to true by default.

More Information

Header File

Consult the `Presence.h` header file directly for more information as needed. The Presence Interface header file `Presence.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.