# Online Services Overview

Learn about the Online Services Interfaces and how to configure them for use in Unreal Engine.

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

The **Online Services** plugin and its interfaces provide a common way to access the functionality of various online services such as Playstation Network, Xbox Live, Epic, Steam, and so on. The design of the Online Services plugin ensures that the only changes developers need to make when working on a game that ships on multiple platforms, or supports multiple online services, are configuration adjustments for each supported service.

## Design Philosophy

The Online Services plugin is organized into modular, service-specific **Interfaces** that group supported features. For a list of interfaces and the feature groups each supports, refer to the Interfaces table on this page.

The Online Services plugin is designed to handle asynchronous communication with a variety of services. Interactions with these systems take an unpredictable amount of time due to

fluctuating network connection speeds, server delays, and unknown backend services times. To overcome these problems, the Online Services Interfaces return a `TOnlineAsyncOpHandle` for all remote operations that guarantees the `OnComplete` event callback to the handle will be called.

Interfaces are provided for every feature group, on each online service that supports them. Specific functions that a particular online service does not support will return `Errors::NotImplemented` from the `OnComplete` callback. This functionality ensures that developers can use the same code for all online services.

# Event Callback and Listening

The `OnComplete` callback provides the following functionalities:

- It responds to requests as they finish.
- It can query in-flight requests.
- It uses a single code path.

This last point is important as it removes the need for developers to write custom code to catch different success or failure conditions.

## Callback Format

Depending on the way developers pass in the parameters for the `OnComplete` callback (the event callback of Online Services) or event listening, the appropriate delegate is constructed automatically. Different delegate-creation functions will be called depending on the type of `this` in the following example using the `QueryStats` function from the Stats Interface:

```
Stats->QueryStats(MoveTemp(Params)).OnComplete(this, &MyClass::OnQueryStatsCompl
```

Copy full snippet

Or in this example using `OnStatsUpdated`:

```
Stats->OnStatsUpdated().Add(this, &MyClass::OnStatsUpdated);
```

Copy full snippet

In either of these examples, we have the following behavior:

- If this is a UObject, the underlying delegate's `CreateUObject` is called.
- If this derives from `TSharedFromThis`, `CreateThreadSafeSP` or `CreateSP` (if it is a non-thread safe shared pointer) is called.
- In any other case, `CreateRaw` is called.

In general, the safest delegate-creation function call will be used.

# Interfaces

The following interfaces are included in the Online Services plugin.

| Interface | Feature group description |
| --- | --- |
| Achievements | List all achievements in a game, unlock achievements, and check your own unlocked achievements, as well as those of other users. |
| Auth | Authenticate and verify a local user with the online services. |
| Commerce | Retrieve the categories and specific offers available for in-game purchase. |
| Connectivity | Get or get notified of the connection status of online services. |
| ExternalUI | Open the built-in user interfaces for specific hardware platforms or online services. In some cases, services grant access to certain core features exclusively through this interface. |
| Leaderboard | Access online leaderboards, including registering your own scores and checking leaderboards for scores from your friends list or other players from around the world. |
| Lobbies | Create and join lobbies to play with friends. |

| Interface | Feature group description |
| --- | --- |
| Presence | Set the way that a user's online status and joinability will appear to other users. Status' include "Online", "Offline", "Away", and so on. |
| Privileges | Query the privileges of a user such as age restrictions, communication restrictions, crossplay settings, and so on. |
| Session | Create, destroy, and manage online game sessions, including searching for sessions and matchmaking systems. |
| Social | Add users to your friends list, block users, unblock users, and list players you have recently met online. |
| Stats | Upload stats to the backend to complete corresponding features such as stats queries, achievements progress, leaderboards standings, and so on. |
| Title File | Enable titles to read files that are not packaged with the shipping title, but are uploaded to the backend services and downloaded to the current title at runtime. |
| User File | Interface with user file storage. |
| User Info | Collect metadata about a user. |

# Functions

Each interface contains a variety of synchronous and asynchronous functions. We now give a brief overview on how to pass parameters to a function and process the result when a function returns. For more detailed information about specific interface functions, refer to the Online Services Interface module in the Unreal Engine C++ API Reference.

## Parameters

Parameters for Online Services Interface functions are created using the `Params` member of each function's associated struct. These parameters are then passed to the relevant function with `MoveTemp`, UE's equivalent of `std::move`, or through a {}-delimited list.

# Return Types

Functions defined in the Online Services interfaces have three different return types:

- `TOnlineResult`
- `TOnlineAsyncOpHandle`
- `TOnlineEvent`

## TOnlineResult

Synchronous functions return a `TOnlineResult<T>` where `T` is the struct associated with the function in question. To determine whether the return was successful, we call `IsOk` or `IsError`. Both of these return a boolean value of whether the result is ok to access or resulted in an error. Lastly, if `IsOk` returns true, the `T::Result` can be accessed with a call to `GetOkValue`. Similarly, if `IsError` returns true, the `FOnlineError` can be accessed with a call to `GetErrorValue`.

## TOnlineAsyncOpHandle

Functions that require asynchronous communication return a `TOnlineAsyncOpHandle<T>`. Adding an `OnComplete` callback on this handle will listen to any final change in state for this handle — whether successful completion, failure, timeout, or otherwise. The callback's `TOnlineResult<T>` parameter will contain either the successful result data or an `FOnlineError` describing why the function failed. This callback accepts a unique function, so unique pointers and heavy data types may be moved into the capture scope of the lambda if a lambda function is used.

## TOnlineEvent

Functions used for event listening return a `TOnlineEvent<T>`. Similar to a `TOnlineAsyncOpHandle`, you can listen to the event callback with the `Add` function. `Add`

will then fire that callback with signature `T` whenever an event matching the conditions is detected. Multiple callbacks can be added to the same event. Calling Add will return a `FOnlineEventDelegateHandle` — this delegate callback will be unbound if this handle is destructed, so make sure to keep it alive for the lifecycle of your system that is listening to this event, and to properly destruct/call `Unbind` on this handle alongside the destruction of the associated system.

# Using Online Services or Online Subsystem

**Unreal Engine (UE)** now provides two frameworks for accessing online services: Online Services and **Online Subsystem**. Read on to determine which is right for your project.

## Online Services

The Online Services plugins have not been tested in shipping titles. As of UE 5.1, the Online Services plugins are an API-complete version for developers to use with the intention that they will be shipping on a future version of the engine. We also recommend using Online Services for developers targeting their own backend, or those who will be incorporating a number of UE upgrades beyond 5.1 into their project before shipping.

## Online Subsystem

Use the Online Subsystem for any title shipping in the near future, or when you do not plan to incorporate any engine upgrades beyond UE 5.1 into the project.

# Configuration

The base module for the Online Services plugins is `OnlineServices`. This module defines and registers service-specific modules with UE. All access to online services will go through this module. `OnlineServices` tries to load the default online service module specified in `DefaultEngine.ini` during initialization. Add the following code to your `DefaultEngine.ini` file to enable online services and specify a default online service:

```
[OnlineServices]
DefaultServices=<DEFAULT_PLATFORM_IDENTIFIER>
```

Copy full snippet

`DEFAULT_PLATFORM_IDENTIFIER` is a variable that you must substitute with one of the following supported platform identifiers:

- Null
- Epic
- Xbox
- PSN
- Nintendo
- Steam
- Google
- GooglePlay
- Apple
- AppleGameKit
- Samsung
- Oculus
- Tencent

The `DefaultServices` specified in `DefaultEngine.ini` is available using the function `UE::Online::GetServices` when no parameter is specified:

```
TSharedPtr<IOnlineServices> GetServices(EOnlineServices OnlineServices = EOnline
```

Copy full snippet

Additional online services are loaded on-demand when a call to `UE::Online::GetServices` requests them. Invalid identifiers or failure to load the module will return `null`.

# Use an Interface

The header files for the various Online Services interfaces are located in the engine directory:

```
UNREAL_ENGINE_ROOT/Engine/Plugins/Online/OnlineServices/Source/OnlineServicesInt
```

Copy full snippet

We encourage you to consult the header files in this directory for more information about Online Services and its various interfaces.

Each of the Online Services Interface documentation pages contains either code examples or a sample process flow to help you get started using the Online Services plugins.

## Run an Interface with a Console Command

You can also run an Online Services Interface using a console command. See the Online Services Console Commands documentation for more information about using Online Services plugin console commands and the syntax they use.

**Online Services Console Commands**

Use console commands to debug and test the Online Services plugin during gameplay.