Auth Interface

Authenticate and verify a local user with online services.

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

The **Online Services Auth Interface** provides an API to authenticate and verify local users with online services. Authentication of a local user returns an account ID, which your project can use to interact with many other online service features.

API Overview

Functions

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

Function	Description
`Login`	Authenticate a local user.

Function	Description
`Logout`	Conclude the auth session for a local user.
`ModifyAccountAttributes`	Modify attributes associated with an authenticated account.
`QueryExternalServerAuthTicket`	Query a ticket to make server-to-server calls on behalf of the signed-in user. Tickets are intended to be single use. Users must call the API again to retrieve a new ticket when making repeated calls where a ticket is required.
`QueryExternalAuthToken`	Retrieve a token for linking the service account with a service account of a different service type.
`QueryVerifiedAuthTicket`	Retrieve a ticket to create a verified authentication session on a remote client.
`CancelVerifiedAuthTicket`	Cancel ticket associated with the verified auth session and clean up any resources associated with the ticket.
Session	
`BeginVerifiedAuthSession`	Start a verified auth session for a remote user.
`EndVerifiedAuthSession`	Clean up the remote verified auth session and any associated resources.
Get Users	
`GetLocalOnlineUserByOnlineAccountId`	Retrieve a logged in user account using Online Account ID.
`GetLocalOnlineUserByPlatformUserId`	Retrieve a logged in user account using Platform User ID.

Function	Description
`GetAllLocalOnlineUsers`	Retrieve all logged in user accounts.
Event Listening	
`OnLoginStatusChanged`	Event triggers as a result of a change in user login status.
`OnPendingAuthExpiration`	Event triggers as a result of a soon-to-expire auth token.
`OnAccountAttributesChanged`	Event triggers as a result of a change to additional attributes associated with an authenticated account.
Helper	
`IsLoggedIn`	Query the login status of a local user.

Enumeration Classes

The Auth Interface defines three enumeration classes that represent a user's login status (`ELoginStatus`), auth ticket audience (`ERemoteAuthTicketAudience`), and auth token method (`EExternalAuthTokenMethod`).

ELoginStatus

Enumerator	Description
`NotLoggedIn`	Player is not logged in or has chosen a local profile.
`UsingLocalProfile`	Player is using a local profile, but is not logged in.

Enumerator	Description
`LoggedInReducedFunctionality`	Player is logged in, but may have reduced functionality with online services.
`LoggedIn`	Player is logged in and validated by the platform specific authentication service.

ERemoteAuthTicketAudience

Enumerator	Description
`Peer`	Generate a ticket appropriate for peer validation.
`DedicatedServer`	Generate a ticket appropriate for dedicated server validation.

EExternalAuthTokenMethod

Enumerator	Description
`Primary`	Acquire an external auth token using the primary method provided by the auth interface.
`Secondary`	Acquire an external auth token using the secondary method provided by the auth interface.

Primary Struct

The primary struct associated with a user and their login information is the `FAccountInfo` struct:

FAccountInfo

Member	Type	Description
`AccountId`	`FAccountId`	The account ID of this user. This represents the user's online platform account.
`PlatformUserId`	`FPlatformUserId`	The platform user ID associated with this online user.
`LoginStatus`	`ELoginStatus`	Login status of this current user.
`Attributes`	`TMap<FSchemaAttributeId, FSchemaVariant>`	Additional account attributes.

Process Flow

Login

`Login` authenticates a local user with the chosen online services. Upon success, `Login` returns an `FAccountInfo` struct. The `FAccountInfo` struct contains an `FAccountId`, which is necessary to use many other functions included in the Auth interface. A successful login also sets the user's `LoginStatus` to `ELoginStatus::LoggedIn`. After a user logs in to their account, the status could change to `ELoginStatus::UsingLocalProfile` or `ELoginStatus::LoggedInReducedFunctionality` in the future based on other conditions.

If there are multiple local users that require login, each user must login individually. Furthermore, platform services may not require an explicit login. On these services, the user or users are implicitly logged in when the application starts. Refer to the specific platform services documentation for more information.

Authenticate with an External Server

Frequently, games have custom web services that provide game-specific functionality. These services need to verify the caller's identity before providing access. `QueryExternalServerAuthTicket` retrieves a single-use ticket for authenticating a user with an external server.

Link Authenticated User with Another Online Service

Many games need to make use of more than one online service. Common cases include a platform service paired with another service which extends the first's functionality. `QueryExternalAuthToken` returns a token appropriate for authentication with a different online service to avoid requiring the user from having to provide separate login credentials for secondary services.

On most platforms, this token is an OpenId token, which provides a guarantee of user identity. This token is then provided to the Login method of the secondary service by setting `CredentialsType` to `ExternalAuth` and passing the token as `CredentialsToken`.

Verify User Identity with P2P or Dedicated Server

Users need to prove their identity and allow for authentication session tracking when connecting to a game server or peer-to-peer network. On the client side, `QueryVerifiedAuthTicket` retrieves a single-use ticket to be sent to the game server proving the user's identity. `CancelVerifiedAuthTicket` cancels this ticket when the play session on the game server has concluded.

Since tickets obtained from a call to `QueryVerifiedAuthTicket` are single-use tickets, `QueryVerifiedAuthTicket` must be called each time a user begins a new verified auth session. The same logic applies to `CancelVerifiedAuthTicket`. `CancelVerifiedAuthTicket` must be called for every ticket that the user has created.

The game server calls `BeginVerifiedAuthSession` with the client supplied ticket when the client connects. Successful completion begins a verified auth session for the user associated with the supplied ticket. `EndVerifiedAuthSession` cleans up associated resources when the game has concluded.

This process for a user client connecting to a remote game server is summarized as:

- The client obtains a session auth ticket with a call to `QueryVerifiedAuthTicket`.
- The client sends the session auth ticket, the ticket ID, and their account ID to the server.
- The server receives this information and uses it to begin a verified auth session with a call to `BeginVerifiedAuthSession`.
- At the end of the session, the client cancels their auth ticket with `CancelVerifiedAuthTicket` and the server ends the auth session with `EndVerifiedAuthSession`.

In a peer-to-peer network model the above process is different. Each user client creates a new authentication ticket for each remote client it is connecting to. The steps for authenticating with a game server then apply for each remote peer:

- Each new remote peer calls `QueryVerifiedAuthTicket` and retrieves a single-use ticket to authenticate with every other existing peer.
- Every existing peer calls `BeginVerifiedAuthSession` to begin a verified auth session for the new remote peer.

> Not all interface implementations have peer-to-peer support. Refer to your platform services documentation for more information.

Modify Account Attributes

Game code may store additional custom attributes in the `FAccountInfo` struct as needed with a call to `ModifyAccountAttributes`.

> Modified attributes persist in the user's data until `Logout` is called. These attributes will not persist after the auth session is ended. `Logout` destroys the user's `FAccountInfo` struct which contains these attributes.

Logout

`Logout` ends the current authentication session for the local user and cleans up associated resources and structures.

Example

Login with Platform Services

```
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::IAuthPtr AuthInterface = OnlineServices->GetAuthInterface();

UE::Online::FAuthLogin::Params Params;
Params.PlatformUserId = PlatformUserId;
Params.CredentialsType = LoginCredentialsType::ExchangeCode;
Params.CredentialsToken = TEXT("1234567890"); // Exchange code from command-line

AuthInterface->Login(MoveTemp(Params)).OnComplete([](const UE::Online::TOnlineResult<UE::Online::FAuthLogin>& Result)
{
if(Result.IsOk())
{
const TSharedRef<UE::Online::FAccountInfo> AccountInfo = Result.GetOkValue().AccountInfo;
// AccountInfo object is now accessible
}
else
{
FOnlineError Error = Result.GetErrorValue();
// Error can now be processed
}
});

```

Copy full snippet

Walkthrough

1. Retrieve the default online services by calling `GetServices` with no parameters, then access the Auth interface:

```
UE::Online::IOnlineServicesPtr OnlineServices = UE::Online::GetServices();
UE::Online::IAuthPtr AuthInterface = OnlineServices->GetAuthInterface();

```

Copy full snippet

2. Instantiate the parameters necessary to login the user:

```
UE::Online::FAuthLogin::Params Params;
Params.PlatformUserId = PlatformUserId;
Params.CredentialsType = LoginCredentialsType::ExchangeCode;
Params.CredentialsToken = TEXT("1234567890"); // Exchange code from command-line

```

Copy full snippet

3. Handle the `Login.OnComplete` callback by registering the account info upon successful login or processing the resulting error:

```
AuthInterface->Login(MoveTemp(Params)).OnComplete([](const UE::Online::TOnlineResult<UE::Online::FAuthLogin>& Result)
{
if(Result.IsOk())
{
const TSharedRef<UE::Online::FAccountInfo> AccountInfo = Result.GetOkValue().AccountInfo;
// Account Info object is now accessible
}
else
{
FOnlineError Error = Result.GetErrorValue();
// Error can now be processed
}
});

```

Copy full snippet

Converting Code from Online Subsystem

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

More Information

Header File

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