Identity Interface

Online subsystems for managing player identity.

Connecting to game sessions and interacting with other players over the Internet usually requires logging in to a registered account on a third-party online service. These services can be run by social media sites, hardware platform owners, or gaming services. In Unreal Engine (UE), the **Identity Interface** handles account-related interactions with these services, providing the ability to authenticate users and obtain access tokens.

Authentication Functions

Login

The `Login` function takes a local player's account credentials (`FOnlineAccountCredentials`) and attempts to login to an online service. To generate account credentials automatically, or from command-line arguments, the `AutoLogin` function can be called instead. Once the online service responds, whether the login attempt succeeds or fails, the `FOnLoginCompleteDelegate` will be called. In addition, the `FOnLoginStatusChangedDelegate` will be called whenever login status for a specific local player changes.

Each local user logs in separately. This is especially important in games with local multiplayer, so that players can compete online, earn scores for leaderboards, invite friends, and earn achievements under their own individual account names.

If a call to `Login` fails, the ExternalUI interface can help by giving the user a chance to log in manually through the online service's built-in user interface. Not all online platforms use `FOnlineAccountCredentials` as part of their authentication process; in some cases, the built-in user interface may be the only supported way to log in.

Logout

To log a user out of an online service, use the `Logout` function. When the action has completed, `FOnLogoutCompleteDelegate` will be called. In addition, the `FOnLoginStatusChangedDelegate` will be called whenever login status for a specific local player changes.

Checking Current Login Status

A local player can be logged in online, logged into a local profile (but not online), or not logged in at all. To find the player's current status, use the `GetLoginStatus` function. Since login status is determined based on the most recent communication with the online service, there are no delegates to bind. However, if you are waiting for a specific status change, such as the player logging in, and don't want to poll this function periodically, binding `FOnLoginStatusChangedDelegate` can help.

On some systems, users can reassign physical input devices to different players, which will change their local player index values within the Unreal Engine. This can switch the local user index values of logged-in users without actually changing any user's login status. The `OnControllerPairingChanged` delegate will be called in this case, providing the controller's index and the `FUniqueNetId` values of the users involved.

Player Identification And Information

Converting Between Identity Systems

Within Unreal Engine's networking environment, an opaque `FUniqueNetId` value is associated with a local player automatically upon a successful login attempt. In addition to being used throughout Unreal Engine's networking code, it also serves as an abstraction for the proprietary identity data types of each online service.

If you need to replicate a user's `FUniqueNetId`, `FUniqueNetIdRepl` provides a `ToString` method that converts an `FUniqueNetId` to a replication-safe string, which can then be converted back by `CreateUniquePlayerId`.

The Identity Interface has a few functions that can bridge the gap between these different systems and identity types. You can retrieve a player's `FUniqueNetId` either by calling `CreateUniquePlayerId` with that player's service-specific identity, or through `GetUniquePlayerId`, called with the player's local user index. Using the player's `FUniqueNetId`, you can call `GetPlatformUserIdFromUniqueNetId` to get the player's service-specific identity (of type `FPlatformUserId`), although this should not be needed in most cases. All of these functions use locally-available information, and thus there are no delegates or callbacks involved.

The `GetSponsorUniquePlayerId` will return the sponsoring player's `FUniqueNetId`, but this function is only implemented for the Xbox Live service.

User Account Information

The abstract class, `FOnlineUser`, represents the base information for user accounts related to any online subsystem, and serves as the common interface for accessing publicly-visible information for local or remote users. An extension of this class, `FUserOnlineAccount`, provides access to all available information for local, logged-in users.

In some cases, an online system's `FOnlineUser` child class may be extended to suit the needs of a specific online service. The base class supports functionality for returning the user's `FUniqueNetId`, real and displayed name (depending on the online service being used), and any string-based attributes that may be associated with the user, although storing these attributes must be implemented in child classes.

The `FUserOnlineAccount` class is also abstract, but establishes a framework for setting user attributes and storing metadata about a local, logged-in user, including a service-specific access token or other data. Some subsystems use these tokens to access features, and you can use them to make RESTful calls, or coordinate with your own backend services.

Retrieving Locally-Known Accounts

Many online services keep track of user accounts that have logged in from (or were created on) the local machine. For services that do this, the function `GetAllUserAccounts` will be implemented to return an array listing all of these known accounts. These accounts will be returned as `FUserOnlineAccount` data. In order to see if a specific player is on the list, `GetUserAccount` can be called to map the player's `FUniqueNetId` to a known `FUserOnlineAccount`, if one is present in the list.

Player Display Names

Some online services provide the option for users to enter "display names" or "nicknames" that differ from their account login names. This name may be preferred over the account name for use in your game's chat, scoreboard, character labels, or similar user-facing displays. Using a player's `FOnlineUser`, call `GetDisplayName` to retrieve that player's display name or nickname from locally-cached user account data.

User Privilege

Online services can act as a gateway, granting or denying access to certain online features, most notably the ability to play games online with other service users. The `GetUserPrivilege` function reports whether or not a user has a specific privilege (defined in `EUserPrivileges`). This function requires contacting the online service, and will respond to user requests through `FOnGetUserPrivilegeCompleteDelegate`, returning a result of type `EPrivilegeResults`.