
EPIC
GAMES
ONLINE
SERVICES

<eos_init.h>

EOS_Initialize( )

Initialize the Epic Online Services SDK.

```
(EOS_EResult) EOS_Initialize(
      const EOS_InitializeOptions* Options);
```

* Initialize the Epic Online Services SDK.
* Before calling any other function in the SDK, clients must call this function.
* This function must only be called one time and must have a corresponding EOS_Shutdown call.
* **@param Options** - The initialization options to use for the SDK.
* **@return** An EOS_EResult is returned to indicate success or an error.
* EOS_Success is returned if the SDK successfully initializes.
* EOS_AlreadyConfigured is returned if the function has already been called.
* EOS_InvalidParameters is returned if the provided options are invalid.

EOS_InitializeOptions( )

Options for initializing the Epic Online Services SDK.

```
EOS_STRUCT(EOS_InitializeOptions, (
      int32_t ApiVersion;
      EOS_AllocateMemoryFunc AllocateMemoryFunction;
      EOS_ReallocateMemoryFunc ReallocateMemoryFunction;
      EOS_ReleaseMemoryFunc ReleaseMemoryFunction;
      const char* ProductName;
      const char* ProductVersion;
      void* Reserved;
      void* SystemInitializeOptions;
));
```

EOS_Shutdown( )

Tear down the Epic Online Services SDK.

```
(EOS_EResult) EOS_Shutdown();
```

* Once this function has been called, no more SDK calls are permitted; calling anything after EOS_Shutdown will result in undefined behavior.
* **@return** An EOS_EResult is returned to indicate success or an error.
* EOS_Success is returned if the SDK is successfully torn down.
* EOS_NotConfigured is returned if a successful call to EOS_Initialize has not been made.
* EOS_UnexpectedError is returned if EOS_Shutdown has already been called.

Platform Interface:

The **Platform Interface provides access to all other EOS SDK interfaces**, and keeps them running. Once you have created the platform interface, you can use it to retrieve handles to other interfaces, or tell it to run its per-frame update code, known as *ticking*.

EOS_Platform_Create( )

Create a single Epic Online Services Platform Instance.

```
(EOS_HPlatform) EOS_Platform_Create(
      const EOS_Platform_Options* Options);
```

* The platform instance is used to gain access to the various Epic Online Services.
* This function returns an opaque handle to the platform instance, and that handle must be passed to EOS_Platform_Release to release the instance.
* **@return** An opaque handle to the platform instance.

```
EOS_STRUCT(EOS_Platform_Options, (
      int32_t ApiVersion;
      void* Reserved;
      const char* ProductId;
      const char* SandboxId;
      EOS_Platform_ClientCredentials ClientCredentials;
      EOS_Bool bIsServer;
      const char* EncryptionKey;
      const char* OverrideCountryCode;
      const char* OverrideLocaleCode;
      const char* DeploymentId;
      uint64_t Flags;
      const char* CacheDirectory;
      uint32_t TickBudgetInMilliseconds;
));
```

EOS_Platform_Release( )

Release an Epic Online Services platform instance previously returned from EOS_Platform_Create

```
(void) EOS_Platform_Release(
      EOS_HPlatform Handle);
```

* This function should only be called once per instance returned by EOS_Platform_Create. Undefined behavior will result in calling it with a single instance more than once.
* Typically only a single platform instance needs to be created during the lifetime of a game.
* You should release each platform instance before calling the EOS_Shutdown function.

<eos_sdk.h>

Once you have an **EOS_HPlatform** handle, you can use it to gain access to the other EOS SDK interfaces through their handle access functions.

To gaining access to the other interfaces, the Platform Interface keeps them all running.

Call **EOS_Platform_Tick** from your game's main loop every frame to make sure that asynchronous functions continue updating.

EOS_Platform_Tick( )

The Platform Instance is used to gain access to all other Epic Online Service interfaces and to drive internal operations through the Tick.

```
(void) EOS_Platform_Tick(
      EOS_HPlatform Handle);
```

* All Platform Instance calls take a handle of type **EOS_HPlatform** as the first parameter.
* **EOS_HPlatform** handles are created by calling **EOS_Platform_Create** and subsequently released by calling **EOS_Platform_Release**.
* Notify the platform instance to do work. This function must be called frequently in order for the services provided by the SDK to properly function. For tick-based applications, it is usually desirable to call this once per-tick.

NAT P2P Interface

Interface to send and receive data between users, and related networking functionality.

The **P2P Interface** enables games implementing the **Epic Online Systems** (EOS) SDK to set up and manage **Peer-to-Peer (P2P) Connections** between users. This enables them to send and receive data between one another directly, typically for the purpose of a multiplayer game. Connections made with the EOS P2P Interface are only established between authenticated users, and are secure-by-default using DTLS, which provides two distinct advantages. First, the speed of handling P2P connections is significantly increased, in that EOS's authentication greatly reduces the need for connections to be re-negotiated. Second, the process of securely handling connections is itself greatly simplified for the SDK user, abstracting out the need for detailed network socket management and condensing most functions to what data needs to be sent and to whom.

In order to use the functions within the **EOS P2P Interface**, you must first obtain a valid **EOS_HP2P** handle from the Platform Interface function **EOS_Platform_GetP2PInterface**, as this handle is used in all P2P Interface functions.

EOS_Platform_GetP2PInterface( )

Get a handle to the Peer-to-Peer Networking Interface.

```
(EOS_HP2P) EOS_Platform_GetP2PInterface(
      EOS_HPlatform Handle);
```

* **@return** EOS_HP2P handle

The P2P Interface uses the struct **EOS_P2P_SocketId** as a title-specified identifier for a connection between peers. Most P2P functions related to connections either require an **EOS_P2P_SocketId** to associate with a connection request, or to return one in order to specify what connection a received connection request is associated with. **EOS_P2P_SocketId** is comprised of the following parameters:

```
EOS_STRUCT(EOS_P2P_SocketId, (
      int32_t ApiVersion;
      char SocketName[33];
));
```

```
*@params API Version of the EOS_P2P_SocketId structure
*@params A name for the connection. Must be a NULL-terminated string of
between 1-32 alpha-numeric characters (A-Z, a-z, 0-9)
```

The **SocketName** field can be a single value for all connections, or it can be a secret value known only to specific players in a multiplayer session. As accepted P2P connections expose a user's IP address to the peer that they are connecting to, it is important to not blindly accept any connection request.

A valid **EOS_ProductUserId** is also usually required both for users to identify themselves when sending data, and to specify which user they wish to send data to.

All function parameters and their associated values in the P2P Interface are required unless explicitly marked as optional. This includes out-parameters, which the P2P Interface often uses to output data for asynchronous functions or event responses.

If a function has a return type of **EOS_EResult** and the return value is not **EOS_Success**, then take note that any out-parameters that the function provides will be unset unless specified otherwise.

EOS_P2P_SendPacket( )

P2P functions to help manage sending and receiving of messages to peers.

```
EOS_DECLARE_FUNC(EOS_EResult) EOS_P2P_SendPacket(
	EOS_HP2P Handle,
	const EOS_P2P_SendPacketOptions* Options);
```

* These functions will attempt to perform **NAT-punching**, but will fallback to relays if a direct connection cannot be established
* **Send a packet** to a peer at the **specified address.** If there is already an open connection to this peer, it will be sent immediately.
* If there is no open connection, an attempt to connect to the peer will be made
* result does not guarantee the packet will be delivered to the peer, as data is sent unreliably.
* **@param** Options Information about the data being sent, by who, to who
* **@return** EOS_EResult::EOS_Success - If packet was queued to be sent successfully
* **@return** EOS_EResult::EOS_InvalidParameters - If input was invalid

```
EOS_STRUCT(EOS_P2P_SendPacketOptions, (
	int32_t ApiVersion;
	EOS_ProductUserId LocalUserId;
	EOS_ProductUserId RemoteUserId;
	const EOS_P2P_SocketId* SocketId;
	uint8_t Channel;
	uint32_t DataLengthBytes;
	const void* Data;
	EOS_Bool bAllowDelayedDelivery;
));
```

EOS_P2P_AcceptConnection( )

Accept connections from a specific peer. If this peer has not attempted to connect yet, when they do, they will automatically be accepted.

```
(EOS_EResult) EOS_P2P_AcceptConnection(
	EOS_HP2P Handle,
	const EOS_P2P_AcceptConnectionOptions* Options);
```

* **@param** Options Information about who would like to accept a connection, and which connection
* **@return** EOS_EResult::EOS_Success - if the provided data is valid
* **@return** EOS_EResult::EOS_InvalidParameters - if the provided data is invalid

```
EOS_STRUCT(EOS_P2P_AcceptConnectionOptions, (
	int32_t ApiVersion;
	EOS_ProductUserId LocalUserId;
```

```
        EOS_ProductUserId RemoteUserId;
        const EOS_P2P_SocketId* SocketId;
));
```

EOS_RESULT_VALUE

EOS_RESULT_VALUE(EOS_Success, 0), Successful result, no error processing needed.
EOS_RESULT_VALUE(EOS_Name, 1-38), Connectivity errors.
EOS_RESULT_VALUE(EOS_Name, 1000-1080), Auth errors.
EOS_RESULT_VALUE(EOS_Name, 2000-2008), Friend-list errors.
EOS_RESULT_VALUE(EOS_Name, 3000-3008), Requested data contains errors.
EOS_RESULT_VALUE(EOS_Name, 4000-4008), Shop/Sale retrieved is stale errors.
EOS_RESULT_VALUE(EOS_Name, 5000-5018), Sessions errors.
EOS_RESULT_VALUE(EOS_Name, 6000-6014), Data request errors.
EOS_RESULT_VALUE(EOS_Name, 7000-7008), Auth errors.
EOS_RESULT_VALUE(EOS_UI_SocialOverlayLoadError, 8000), Social error.
EOS_RESULT_VALUE(EOS_Name, 9000-9017), Lobby errors.
EOS_RESULT_VALUE_LAST(EOS_UnexpectedError, 0x7FFFFFFF), An unexpected error that we cannot identify has occurred.

* Returns a string representation of an EOS_EResult.
* The return value is never null.

* The return value must not be freed.

* Example: EOS_EResult_ToString(EOS_Success) returns “EOS_Success"
* Useful for printing errors in console
EOS_DECLARE_FUNC(const char*) EOS_EResult_ToString(EOS_EResult Result);

* Returns whether a result is to be considered the final result, or false if the callback that returned this result
* will be called again either after some time or from another action.
*
* @param Result The result to check against being a final result for an operation
* @return True if this result means the operation is complete, false otherwise
EOS_DECLARE_FUNC(EOS_Bool) EOS_EResult_IsOperationComplete(EOS_EResult Result);

* Check whether or not the given account unique id is considered valid
* @param AccountId The account id to check for validity
* @return EOS_TRUE if the EOS_EpicAccountId is valid, otherwise EOS_FALSE
EOS_DECLARE_FUNC(EOS_Bool) EOS_EpicAccountId_IsValid(EOS_EpicAccountId AccountId);

* Retrieve a string-ified account ID from an EOS_EpicAccountId. This is useful for replication of Epic account IDs in multiplayer games.
*
* @param AccountId The account ID for which to retrieve the string-ified version.
* @param OutBuffer The buffer into which the character data should be written
* @param InOutBufferLength The size of the OutBuffer in characters.
*   The input buffer should include enough space to be null-terminated.
*   When the function returns, this parameter will be filled with the length of the string copied into OutBuffer.
*
* @return An EOS_EResult that indicates whether the account ID string was copied into the OutBuffer.
*   EOS_Success - The OutBuffer was filled, and InOutBufferLength contains the number of characters copied into OutBuffer excluding the null terminator.
*   EOS_InvalidParameters - Either OutBuffer or InOutBufferLength were passed as NULL parameters.
*   EOS_InvalidUser - The AccountId is invalid and cannot be string-ified
*   EOS_LimitExceeded - The OutBuffer is not large enough to receive the account ID string. InOutBufferLength contains the required minimum length to perform the operation successfully.

EOS_DECLARE_FUNC(EOS_EResult) EOS_EpicAccountId_ToString(EOS_EpicAccountId AccountId, char* OutBuffer, int32_t* InOutBufferLength);

 * Retrieve an EOS_EpicAccountId from a raw account ID string. The input string must be null-terminated.
 *
 * @param AccountIdString The string-ified account ID for which to retrieve the EOS_EpicAccountId
 * @return The EOS_EpicAccountId that corresponds to the AccountIdString
EOS_DECLARE_FUNC(EOS_EpicAccountId) EOS_EpicAccountId_FromString(const char* AccountIdString);

/** A character buffer of this size is large enough to fit a successful output of EOS_EpicAccountId_ToString */
#define EOS_EPICACCOUNTID_MAX_LENGTH 128