**Integrating Everyplay in Unity3D**

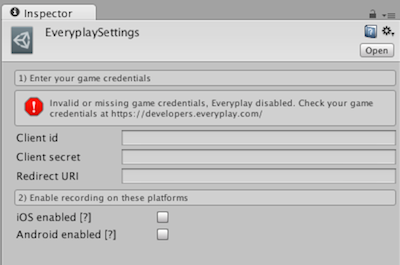
This document is also available at <https://developers.everyplay.com/documentation/Integrating-Everyplay-in-Unity3D> which is recommended to be used.

NOTE! To use the improve Facebook sharing of this SDK, please email your bundle ID to support@everyplay.com!

Integrating Everyplay into a Unity3D-powered game is a very simple task, and this document will guide you through it. If you have not yet done so, then you will need to download the Everyplay SDK from the [Unity Asset Store](https://www.assetstore.unity3d.com/#/content/16005).

**Configuring Everyplay**

First, you need to create an account at the [Everyplay Developers](https://developers.everyplay.com/) website and input your game details. Doing so will give your game a unique *client ID*, *client secret*, and the proper *redirect URI*. You need to input these values under*Everyplay Settings*, which you can bring up by selecting *Edit* > *Everyplay Settings*.



The *Everyplay Settings* panel also allows you to enable / disable Everyplay on a specific platform. Once you have the client credentials in place, you can move on to configuring what to record in your game!

**Recording with Everyplay**

**Checking for Recording Support**

Everyplay is initialised automatically in your game as long as you have enabled the target platform in *Everyplay Settings*. Internally the initialization process is asynchronous, so in some early application load use cases it might be good idea to use Everyplay.ReadyForRecording event.

On Android, due to wide range of driver behaviour, hardware encoders, GPUs and Android version differences out there, Everyplay caches device specific settings online from a remote server.

Until the settings are successfully received, the recording support is automatically disabled. After receiving the server response, the recording support is either enabled or continued to be disabled to workaround devices known to cause trouble. Next time the application is started, the settings are applied from the cache immediately upon startup without requiring network access.

In it's current form, there's a chance of getting invalid status from EveryPlay.IsRecordingSupported() if the method is called before ReadyForRecording(true) event.

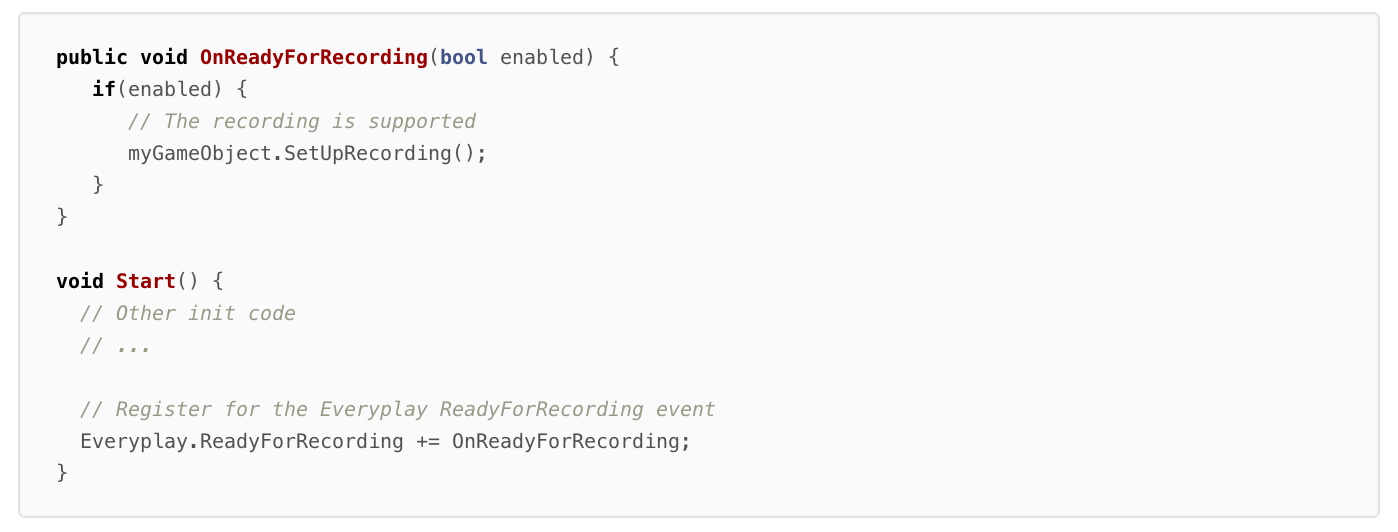
The event can get called multiple times during the game's lifecycle, depending on things like:

* Orientation changes (Android/iOS)
* Activity changes (Android)
* UIView/UIViewController activity (iOS)

On these cases, there can be short periods when the event first gets called with false value, followed by truelater on, so you shouldn't build too much "once in a lifetime" initialization logic on top of ReadyForRecording.

On Android and unsupported device scenario, there will be a ReadyForRecording(false) event. On iOS the recording is basically supported on all devices (running iOS 5 or later, Everyplay is no-op on older iOS releases).

Example:

****

**Controlling Session Recording**

The gameplay recording starts with calling the method Everyplay.StartRecording() and can be stopped with the method Everyplay.EndRecording(). You can pause (and resume) the recording as well, or check if the recording is running or paused. You should use Everyplay.IsSupported() to verify that the user's current device supports Everyplay recording. All the methods are described below:

| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| StartRecording() | Starts the recording process in your game | None | None |
| StopRecording() | Stops the currently running session recording | None | None |
| PauseRecording() | Pauses the currently running recording session in your game | None | None |
| ResumeRecording() | Resumes a paused recording session in your game | None | None |
| TakeThumbnail() | Takes a thumbnail from the current frame. Thumbnail may be rendered to a texture or a file, but only while recording | None | None |
| IsRecording() | Check whether Everyplay is currently recording | None | **bool** - trueif the current session is being recorded,falseotherwise |
| IsPaused() | Check whether there is a recording process in Everyplay, that is currently in a paused state | None | **bool** - trueif there is a paused recording session,falseotherwise |
| IsSupported() | Check if Everyplay is supported on the current device / iOS version. For iOS 4.3 and older, Everyplay is disabled so this could be used for hiding Everyplay-specific UI functionality | None | **bool** - trueif Everyplay is supported on this device,falseotherwise |
| IsRecordingSupported() | Check if the actual recording feature is supported on the current device / iOS version. The return value also takes into account if the user has calledSetDisableSingleCoreDevices(true) | None | **bool** - trueif the recording is supported on this device,falseotherwise |
| SetDisableSingleCoreDevices(bool) | Disables recording for single core devices. Depending on a game and the device, recording a gameplay may be too heavy on CPU/memory resources left | **bool** - true if recording should be disabled on single core devices, falseotherwise | None |
| SetLowMemoryDevice(bool) | Optimize for low memory devices. Disabled by default | **bool** - true if optimizations should be enabled on current device,falseotherwise | None |
| SetMaxRecordingMinutesLength(int) | Sets the maximum time for the recording | **int** - Maximum minutes to record | None |

**Note**: Especially if you are building for Android, it is important to call IsRecordingSupported() before using Everyplay in your game, as the number of Android devices that don't support Everyplay is higher than on iOS.

**Handling a Recorded Session**

Once a gameplay session has been recorded, you can instruct the Everyplay prefab to play it back for the user with the Everyplay.PlayLastRecording() method. For the purpose of showing a video thumbnail of the recording, you can get a video thumbnail by calling the Everyplay.LoadThumbnailFromFilePath method. The methods are described below:

| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| PlayLastRecording() | Starts the playback of the previously recorded play session | None | None |
| ShowSharingModal() | Shows the Everyplay sharing modal of the previously recorded play session | None | None |
| LoadThumbnailFromFilePath(string, ThumbnailLoadReadyDelegate, ThumbnailLoadFailedDelegate) | Asynchronously loads a thumbnail to a Texture2D object from the given file path. The file path is accessible via theThumbnailReadyAtFilePathevent (more in "Receiving Events from Everyplay") | **string** - The filepath where the thumbnail should be read from **delegate** - ThumbnailLoadReadyDelegate(Texture2D) **delegate** - ThumbnailLoadFailedDelegate(string) | None |

Everyplay always creates one file-based thumbnail from some random point of the video. In case the user triggers the TakeThumbnail method before Everyplay has taken the shot, the random shot will be cancelled.

**Loading a Video Thumbnail from a file**

To load a video thumbnail, you need to use theEveryplay.LoadThumbnailFromFilePath(string path, ThumbnailLoadReadyDelegate, ThumbnailLoadFailedDelegate)method. The method works asynchronously and takes the path as a string parameter, and two delegate functions of that are invoked when the thumbnails are loaded.

The ThumbnailLoadReadyDelegate function takes a Texture2D object as a parameter, which provides the thumbnail from the video. In case of an error, the ThumbnailLoadFailedDelegate is called, and the error message is passed as a string. When you no longer need the texture you should release it with Destroy.

Example:

**

You may change the size of the file based thumbnail with SetThumbnailWidth method before you start the recording.

| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| SetThumbnailWidth(int) | Sets the file-based thumbnail maximum side width | **int** - maximum side width of the texture | None |

**Rendering a Video Thumbnail to a texture**

Instead of using a file-based target for the thumbnail, you may render the thumbnail directly to an OpenGL texture. Using a texture is a lot better for performance and the thumbnail may be shown even during the recording without a significant performance hit. When rendering the thumbnail to a texture, Everyplay does not automatically take a thumbnail but you may request one at any time during the recording by calling the TakeThumbnail method. To make the memory management a bit easier for games which support multiple orientations (landscape and portrait) the thumbnails are always rendered in landscape.

| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| SetThumbnailTargetTextureId(int) | Sets the thumbnail target texture | **int** - OpenGL texture handle id | None |
| SetThumbnailTargetTextureWidth(int) | Sets the target texture width | **int** - width of the texture | None |
| SetThumbnailTargetTextureHeight(int) | Sets the target texture width | **int** - height of the texture | None |

You could easily create an animated thumbnail from the recording by rendering thumbnails to multiple textures from selected highlight points (or at random). To make this even easier, the Everyplay Unity package includes a couple of helper files to achieve this. See the following documentation for more information on this:

* [**Animated Thumbnail integration to Unity3d game**](https://developers.everyplay.com/documentation/Everyplay-Animated-Thumbnail-integration-to-Unity3d-game)

**Interacting with the Everyplay Service**

You can start the Everyplay service in your code by calling the Everyplay.Show().

| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| Show() | Shows the Everyplay service | None | None |
| ShowWithPath(string) | Shows the Everyplay service with a certain path. For now, the following paths are supported besides the default without parameters: **/feed/game** - Loads the latest feed of the current game | **string** - The path key | None |

**Receiving Events from Everyplay**

Everyplay provides the following events that your code can receive, to be notified of key actions in the recording process. These are:

| **Event** | **Required delegate** | **Description** |
| --- | --- | --- |
| ReadyForRecording | ReadyForRecordingDelegate(bool isSupported) | The ReadyForRecording event is dispatched when the Everyplay initialisation has been completed and the recording support of the current device has been analysed. |
| RecordingStarted | RecordingStartedDelegate() | The RecordingStarted event is dispatched to any receiving delegates when the Everyplay recording service starts recording the gameplay |
| RecordingStopped | RecordingStoppedDelegate() | The RecordingStopped event is dispatched to any receiving delegates when the Everyplay recording service stops recording the gameplay |
| UploadDidStart | UploadDidStartDelegate(int videoId) | The UploadDidStart event is dispatched to any receiving delegates when an upload is started |
| UploadDidProgress | UploadDidProgressDelegate(int videoId, float progress) | The UploadDidProgress event is dispatched to any receiving delegates in regular intervals during the upload process |
| UploadDidComplete | UploadDidCompleteDelegate(int videoId) | The UploadDidComplete event is dispatched to any receiving delegates when an upload is completed |
| ThumbnailReadyAtFilePath | ThumbnailReadyAtFilePathDelegate(string filePath) | The ThumbnailReadyAtFilePath event is dispatched to any receiving delegates when the Everyplay recording has generated a video thumbnail to a file from the previous or current recording. Please note that this event may be dispatched*while the recording is still running*. It is recommended that your code not attempt thumbnail loading when a game session is still in progress, just store the given file path, as thumbnail loading may cause a performance hit in your game |
| ThumbnailReadyAtTextureId | ThumbnailReadyAtTextureIdDelegate(int textureId, bool portrait) | The ThumbnailReadyAtTextureIdevent is dispatched to any receiving delegates when the Everyplay recording has rendered a video thumbnail to a texture from the previous or current recording |
| WasClosed | WasClosedDelegate() | The WasClosed event is dispatched to any receiving delegates when the Everyplay service/videoeditor is closed |

Remember to remove your listeners when your objects are destroyed.

**Example of using events:**

***See online documentation at*** [***https://developers.everyplay.com/documentation/Integrating-Everyplay-in-Unity3D***](https://developers.everyplay.com/documentation/Integrating-Everyplay-in-Unity3D) ***for code samples***

**Adding Metadata**

You can also add metadata to uploaded videos to provide additional information about the video, such as the score the user achieved, the level they were playing, etc. The metadata is set using the Everyplay.SetMetadata() method. You can call it during or after the recording (affects the last known session).

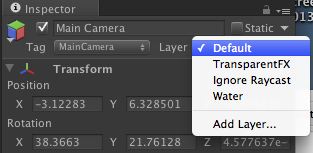
| **Method** | **Description** | **Parameters** | **Return value(s)** |
| --- | --- | --- | --- |
| SetMetadata(string,object) | Adds a single metadata item to the video with the given key, and the given value | **string** - The metadata key  **object** - The metadata value | None |
| SetMetadata(Dictionary<string,object> values) | Add multiple metadata values from the provided Dictionary | **Dictionary<string,object>** - The key-value pairs of metadata to add to this video | None |

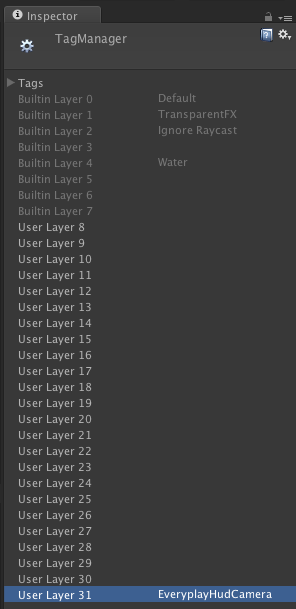
Example of adding metadata

Everyplay.SetMetadata("level", levelNumber); Everyplay.SetMetadata("level\_name", levelName); Everyplay.SetMetadata("score", score)

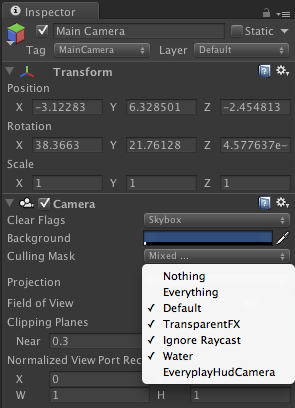
**HUD-less recording**

1. Add a new layer for the HUD game objects, call it EveryplayHudCamera for example

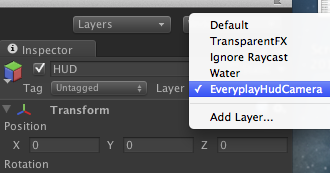




1. Remove EveryplayHudCamera layer from your Main Camera's culling mask. Do the same for other cameras you might have



1. Drag the EveryplayHudCamera prefab to the root of your hierarchy (position 0, 0, 0, rotation 0, 0, 0)
2. Set layer to EveryplayHudCamera on game objects you wish to **not** be recorded on video



1. You're done! :)

**Project Build Issues**

**Note** When building a project with Everyplay for the first time, make sure to do a *clean* build by selecting *File* >*Build Settings* > *Build and Run*.

**Note** When building for Android and running Unity 4.2, you need to disable Version Control support during the build process. You can do this via *Edit* > *Project Settings* -> *Editor*.

In the majority of cases, building your Unity3D project with Everyplay enabled for iOS should succeed without any configuration changes necessary for your Xcode project. In a few cases, however, some manual configuration may be required. These can be caused by:

* Multiple version of the Everyplay SDK on your computer
* Spotlight being disabled on your computer

There are a few alternatives to correcting any build process problems caused by Everyplay:

1. If your computer has Spotlight disabled, the easiest way is to enable Spotlight
2. If you have multiple versions of the Everyplay SDK on your computer, you can create an empty file namedACTIVE\_EVERYPLAY\_SDK\_DIR in the directory of the desired Everyplay SDK version that contains the .framework and .bundle files
3. If you don't have Spotlight, and don't want to enable it, you can create a symbolic link called.active\_everyplay\_sdk\_dir that points to the directory that contains the .framework and .bundle files. The symlink should reside in your home directory. Example of creating a symlink:

ln -sf /home/dev/downloads/everyplay-ios-sdk-1.x/ ~/.active\_everyplay\_sdk\_dir

1. Instead of relying to Spotlight or use of symlinks, you can also add Everyplay.framework and Everyplay.bundle into your Unity project directory, ideally outside of the Assets directory

Should build errors related to Everyplay still persist, you can contact us directly via email (support@everyplay.com) or use Github Issues at <https://github.com/Everyplay/everyplay-ios-sdk/issues>, thanks!

**About Localization**

The Everyplay SDK supports Japanese and Korean in addition to English. To see these languages in Everyplay, please follow these steps:

1. After the integration of Everyplay.unitypackage, export your unity project to an iOS project.
2. Under the iOS project root directory, create directories and files as follows.

mkdir {en,ja,ko}.lproj touch {en,ja,ko}.lproj/Localizable.strings

1. In Xcode, add all the Localizable.strings files to your project to enable the Japanese and Korean languages (Right click the project and select "Add Files to YOUR\_PROJECT").