reK – ReplayKit Plugin for iOS

Overview

reK – ReplayKit Plugin for iOS enables you to integrate video recording functionality into your Unity iOS game within a matter of minutes.

Starting with version 1.4, reK also supports **ReplayKit Live**, and thus provides support for live streaming and broadcasting to services such as Mobcrush or Periscope. See below for more information on how to integrate this into your game.

Available on the Asset Store

Unity Asset Store

Support

If you have questions, do not hesitate to contact us directly: rek@rarebyte.com

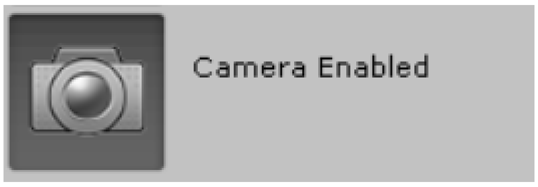
There is also a Unity forum thread available.

Microphone Support

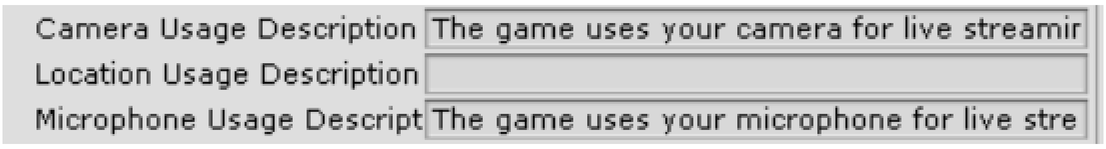
reK – ReplayKit Plugin for iOS supports using the device’s microphone to record additional audio which is great for let’s play videos.

Camera Support

reK – ReplayKit Plugin for iOS supports using the device’s camera to record video from the front camera. A small picture-in-picture view is added to your game if you selected this option.



Important: If you use the camera or microphone in your game, you will need to add the corresponding usage descriptions in your iOS build settings:



Camera Usage Description	The game uses your camera for live streamir
Location Usage Description	
Microphone Usage Descript	The game uses your microphone for live stre

Integration: Native iOS Overlay

Example scene: *UnityReplayKitExamples/UnityReplayKitExampleNativeOverlay*

The easiest way to integrate reK – ReplayKit Plugin for iOS is as simple as that:

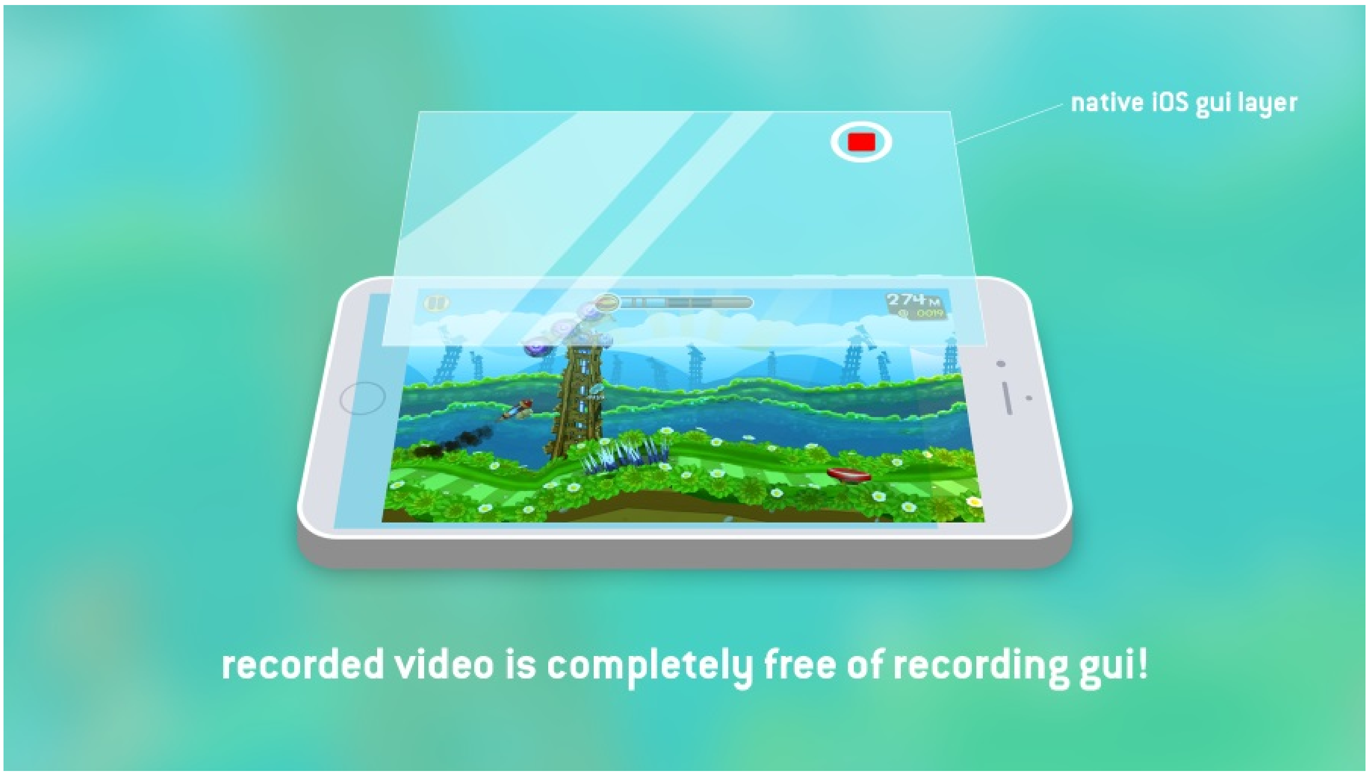
- Drag and drop the ***unity_replay_kit*** prefab into your scene.
- Done!

By default, the option *Native iOS Overlay* is already selected. The following options are available:

- Anchor: Defines the position of the native widget
- Offset (if not using anchor *Custom*): Relative position to the anchor
- Position (if using anchor *Custom*): Absolute position of the widget
- Auto-scale widget: Whether the native widget should be scaled automatically depending on the device's screen height
- Size (either in percent or in pixels, depending on the auto-scale setting): The size of the native widget

There is also preview of the recording controls widget available in the game view of Unity that reflects all parameter changes immediately.

A big advantage of the Native iOS Overlay is that the recording widget itself is **not** recorded to your video.



Integration: Custom

If you do not want to use the *Native iOS Overlay* method, just set the mode to ***Custom***. The package includes examples for both Unity UI and NGUI (which is still used by a lot of developers). Additionally, we included an code-only example (using the legacy OnGUI system).

Unity UI

Example scene: *UnityReplayKitExamples/UnityReplayKitExampleUnityUI*

NGUI

To use the NGUI example, you need to import *UnityReplayKitNGUIExample.unitypackage* in *UnityReplayKit/Examples*, and – of course – you need to have NGUI imported.

Example scene: *UnityReplayKitNGUI/UnityReplayKitExampleNGUI*

Code-Only

As a reference, take a look at the example scene *UnityReplayKitExamples/UnityReplayKitExampleOnGui*

In some games it is very useful to record complete runs (e.g. in an endless runner) and provide players the option to either edit/share the video at the end of the run. In order to do that the recording can be started and stopped automatically via code:

```
// the run/game starts, start recording
UnityReplayKit.Instance.StartRecording();

// ...

// the run/game ends, stop recording
UnityReplayKit.Instance.StopRecording();

// present the video edit/preview window to the player
UnityReplayKit.Instance.ShowPreview();
```

You can also use actions to react to what’s happening with your recordings, e.g.:

```
UnityReplayKit.Instance.Started += () => Debug.Log("Recording has started");
```

The actions available are:

```
// triggered if the recording cannot be started (e.g. because disk is full)
public Action<ReplayKitError> StartFailed;

// triggered after the recording has been started
public Action Started;

// triggered if stopping the recording has failed
public Action<ReplayKitError> StopFailed;

// triggered after the recording has been stopped
public Action Stopped;

// triggered after a recording has been discarded
```

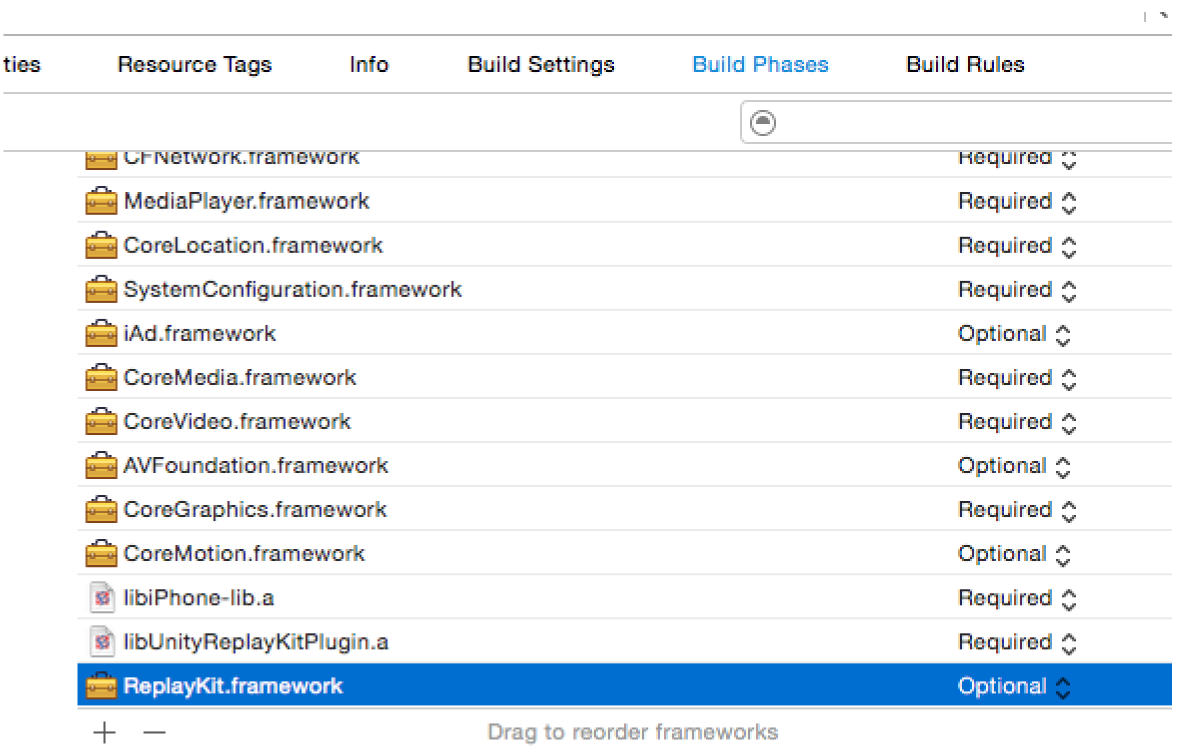
```
public Action Discarded;

// triggered after the user has completed interaction with the
native preview
public Action PreviewCompleted;

// called just before starting the recording
public Action PreStart;

// called just before stopping the recording
public Action PreStop;

// called just before displaying the preview dialog
public Action PreShowPreview;

// called just before discarding a recording
public Action PreDiscard;
```

Compatibility

reK – ReplayKit Plugin for iOS depends on Apple's ReplayKit framework that has the following requirements:

- iOS version: 9 or higher
- Supported CPUs: A7, A8 or higher

Important: If you need your game to run on older versions of iOS, you need to set the ReplayKit framework dependency to *Optional* in the *Build Phases* settings in your Xcode project, just like in this screenshot:


ties Resource Tags Info Build Settings Build Phases Build Rules

CFNetwork.framework	Required
MediaPlayer.framework	Required
CoreLocation.framework	Required
SystemConfiguration.framework	Required
iAd.framework	Optional
CoreMedia.framework	Required
CoreVideo.framework	Required
AVFoundation.framework	Optional
CoreGraphics.framework	Required
CoreMotion.framework	Optional
libiPhone-lib.a	Required
libUnityReplayKitPlugin.a	Required
ReplayKit.framework	Optional

You can also check if ReplayKit is available by using the *IsReplayKitAvailable* property like this:

```
if(UnityReplayKit.Instance.IsReplayKitAvailable) {
  // available
} else {
  // not available, e.g. hide buttons etc.
}
```

Integrating Live Streaming/Broadcasting Support

Using reK you can add live streaming/broadcasting support to your game. It supports all third party applications that provide a *Broadcast UI extension*. In the game, players would have to perform these tasks to start a broadcast:

- Select a broadcast service (e.g. Mobcrush) via the native iOS broadcast service selection dialog
- Configure broadcast service options (dialog coming directly from the broadcasting app), e.g. enter an epic title for the broadcast
- Start streaming

Some services also support a pause/resume functionality which can also be accessed via reK.

Integration reK live streaming into your game is very easy. Take a look at the example scene *UnityReplayKitExampleBroadcasting*. All you have to do is drag the unity_replaykit prefab into your scene, set it to *Custom* (unless you also want the native recording UI available as described above).

To let the user select a broadcasting service, just call the following method via code or wire up a button with:

```
UnityReplayKit.Instance.SelectBroadcastingService();
```

After the user has selected and configured the broadcasting service, start broadcasting with:

```
UnityReplayKit.Instance.StartBroadcast();
```

To stop the broadcast, just call:

```
UnityReplayKit.Instance.StopBroadcast();
```

There are some callbacks from the plugin that are quite useful:

```
// Is called just after a broadcast service has been selected
public Action BroadcastServiceSelected;

// Is called when the broadcast service selection has failed
public Action<ReplayKitError> BroadcastServiceSelectionFailed;

// Is called just after broadcasting has started successfully
public Action BroadcastStarted;

// Is called when the broadcast has failed to start
public Action<ReplayKitError> BroadcastStartFailed;

// Is called just after the broadcast has initiated stopping
successfully
public Action BroadcastStopped;

// Is called when trying to stop the broadcast has failed
public Action<ReplayKitError> BroadcastStopFailed;

// Is called just after the broadcast has been finished
```

```
public Action BroadcastFinished;

// Is called when finishing the broadcast has failed
public Action<ReplayKitError> BroadcastFinishFailed;
```

You can easily add your custom code to those actions like this:

```
UnityReplayKit.Instance.BroadcastServiceSelected += () =>
Debug.Log("Broadcast service selected");
UnityReplayKit.Instance.BroadcastServiceSelectionFailed +=
(error) => Debug.Log("Broadcast selection failed: " +
error.ToString());
```