Flashback Video Recorder

Version 3.0

The latest version of this document can be found at http://www.launchpointgames.com/unity/flashback

Created by LaunchPoint Games
Contact: support@launchpointgames.com

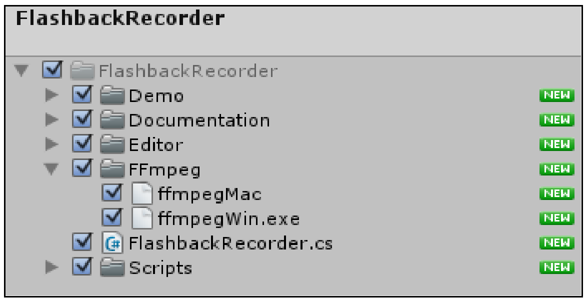
Table of Contents

1. Introduction and Overview

Flashback Video Recorder continuously captures the last few moments of gameplay, which can be written to a GIF or an MPEG-4 file with the press of a button. Only desktop platforms (Windows PC and Mac OSX) are currently supported, and all encoding is done through FFmpeg, which is included in the package.

FFmpeg is a third party, open source, cross-platform tool that lets you easily convert video formats, and is bundled with Flashback Video Recorder. You can learn more here: http://ffmpeg.org

When you import Flashback Video Recorder into your project, the following assets will be added to your project:



Demo/	Stores the scene file and all other assets for a fully functional demonstration of Flashback Video Recorder. When importing to an existing project, you may choose to exclude this folder.
Documentation/	Holds the latest PDF version of this file.
Editor/	Contains resources used in the Unity Editor.
FFmpeg/	Contains the FFmpeg binaries for Windows and Mac OSX. If you are only building for one target platform, you can exclude the file you don't need.
FlashbackRecorder.cs	The interface to Flashback Video Recorder's core logic. You will attach this to your main camera.
Scripts/	Contains helper scripts that launch FFmpeg with the settings specified.

This guide covers adding Flashback Video Recorder to a new or existing project, and provides a detailed explanation on how the package works under the hood.

If you are not sure if Flashback Video Recorder is right for your project, check out the "Key Limitations" section of this document, as well as the "Features Roadmap" section. If you have additional questions, please email us directly at support@launchpointgames.com. We do our best to respond as quickly as possible.

2. Key Limitations

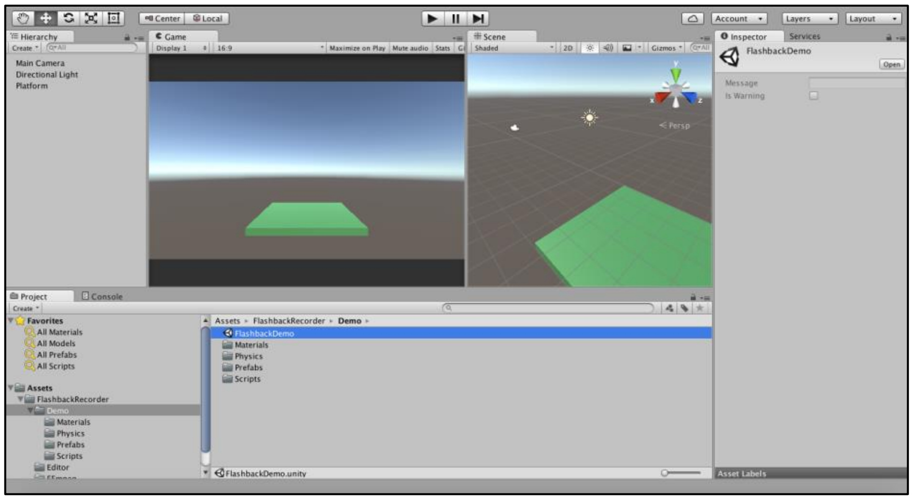
There are some limitations that you should be aware of when deciding whether Flashback Video Recorder is appropriate for your project.

Limitation	**Description**
Desktop Only	Flashback Video Recorder only works on Windows and Mac standalone, or in the editor. Currently, there is no plan to support mobile devices.
Performance Impact	Flashback Video Recorder will have a minor impact on performance, depending on capture settings and computer hardware. See the “FAQ and Troubleshooting” section for ways to improve performance.

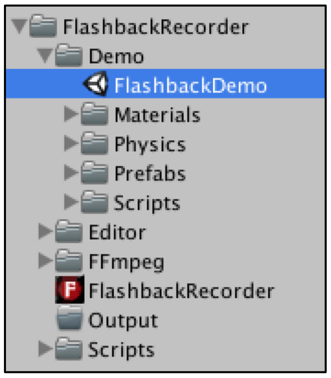
To learn more about planned features, check out the “Features Roadmap” section of this document.

3. Demo Quick Start

Flashback Video Recorder comes with a demo scene you can use for testing. Start by creating a new project and importing all files in the Flashback Video Recorder asset package.



Step 1. Open the Demo Scene located in /Assets/FlashbackRecorder/Demo/



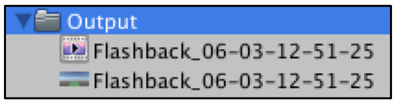
Note for Mac Users

Make sure `FFmpegMac` has executable permissions by going to your project root directory in a terminal and running the command (you only need to do this once):

```
chmod +x
Assets/FlashbackRecorder/Ffmpeg/ffmpegMac
```

Step 2. Play in editor, and after a few seconds, press the SPACE BAR.

Step 3. The recorded GIF and MP4 files are saved with the prefix "Flashback" in the following project folder: /Assets/FlashbackRecorder/Output/



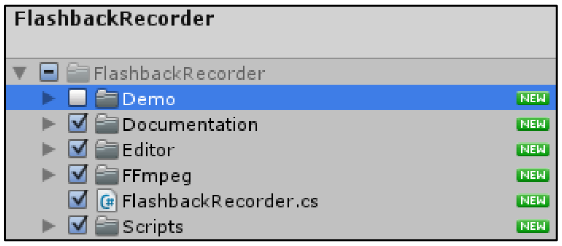
Note: You may have to right-click in the file browser and select "Refresh" for the files to appear in the Unity Project view.

On Windows, you may get an error when importing MP4 files into Unity if QuickTime is not installed. This is expected, and you can ignore the error. There is more information in the "FAQ and Troubleshooting" section of this document.

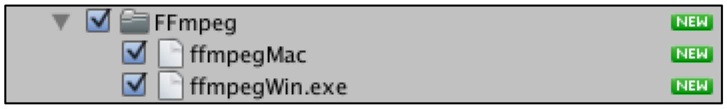
4. Flashback Recorder Integration Guide

Step 1: Import the Flashback Video Recorder Package

Import the Flashback Video Recorder package into your project. If you are working with an existing project, you can choose to exclude the "Demo" folder:



If you are planning to build for only one platform (either Windows or Mac), then you can exclude the FFmpeg file that you won't be using. If you are unsure, you can keep both files:



Note for Mac Users

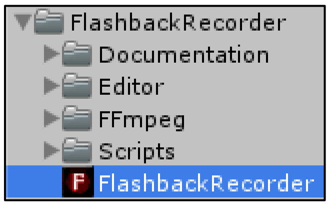
Make sure `FFmpegMac` has executable permissions by going to your project root directory in a terminal and running the command (you only need to do this once):

```
chmod +x
Assets/FlashbackRecorder/Ffmpeg/ffmpegMac
```

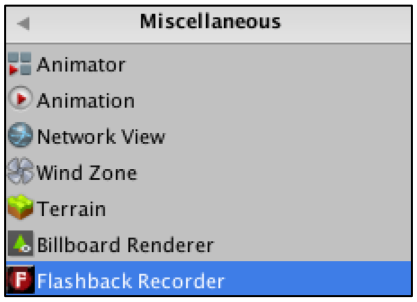
Step 2: Attach the Flashback Video Recorder to the Camera

After you have imported the Flashback Recorder package, you will need to attach the FlashbackRecorder script to the main camera. There are two ways to do this:

Option 1. Drag the "FlashbackRecorder/FlashbackRecorder.cs" script onto the camera object.

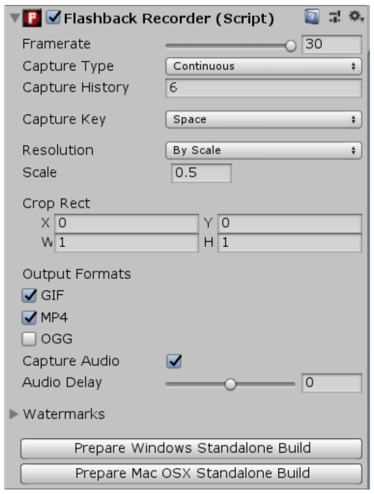


Option 2. Click on the camera object in the Hierarchy window, select "Add Component" then click "Miscellaneous" and "Flashback Recorder".



Step 3: Configure the Flashback Recorder’s Properties

After you've attached the FlashbackRecorder component to a camera, the inspector allows you to set the following properties:



Flashback Recorder (Script)

Framerate 30
Capture Type Continuous
Capture History 6
Capture Key Space
Resolution By Scale
Scale 0.5
Crop Rect
X 0 Y 0
W 1 H 1
Output Formats
✓ GIF
✓ MP4
☐ OGG
Capture Audio ✓
Audio Delay 0
Watermarks

Prepare Windows Standalone Build
Prepare Mac OSX Standalone Build

Framerate – The number of frames to capture every second. This should be lower than your actual camera framerate, otherwise you'll see stuttering in the output video. Lower values will result in smaller file sizes, but decreased quality. By default, you can capture up to 30 frames per second.

Capture Type – You can choose to capture continuously, or toggle recording on/off with a button press.

> *Continuous* – The last few seconds (as specified by “Record Time”) will be written to disk when the “Capture Key” is pressed.
>
> *Toggle* – The first time the “Capture Key” is pressed, recording will start, the second time it is pressed, the recording will stop and the output will be written to disk.

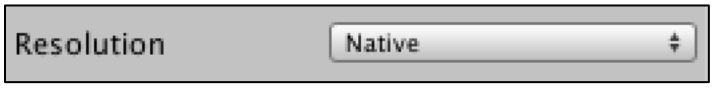
Capture History – The amount of video history to save to disk (in seconds) if the Capture Type is set to “Continuous”. Be aware that larger values will require more memory and will take longer to process when writing to disk.

Capture Key - The key you want to assign to video capture. By default, it is the SPACE BAR, but you can select any key on the keyboard. If you'd like to implement your own method of capturing the video output, set the Capture Key to “None” and then call "SaveCapturedFrames" in the FlashbackRecorder component directly.

Resolution – There are three different ways to specify the output resolution of the captured video, each with different options.

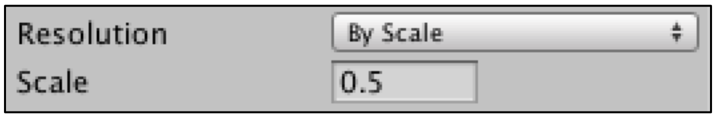
Keep in mind that no matter which option you choose, the higher the resolution, the more memory and processing time is required.

> *Native* – The video will be captured at the screen's current (native) resolution. The height and width of the video will be set at runtime during the `Start()` function call.



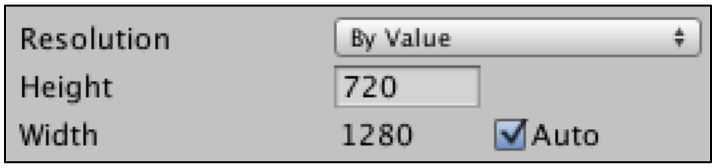
Resolution: Native

> *By Scale* – The video will be scaled from the native resolution. A value of 1 will match the screen's height and width (which is the same as choosing *Native*), a value of 0.5 will be half the screen's height and width, and a value of 2 will be twice the native height and width.



Resolution: By Scale
Scale: 0.5

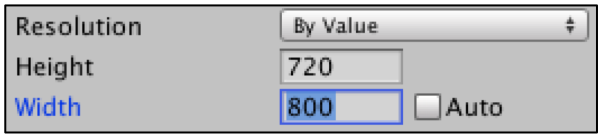
> *By Value* – This gives you the option to explicitly specify the pixel height of the captured video, and optionally, the width. By default, the width is automatically calculated to maintain the camera's aspect ratio.



Resolution: By Value
Height: 720
Width: 1280 ✓ Auto

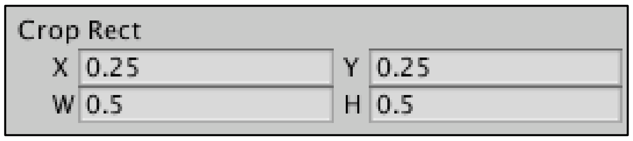
If "Auto" is selected, the width will be calculated at be set at runtime during the `Start()` function call.

You can always manually choose to set the width by unchecking "Auto".

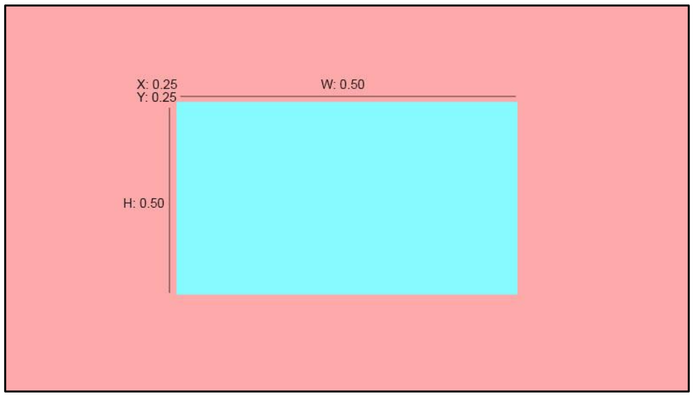


Resolution: By Value
Height: 720
Width: 800 ☐ Auto

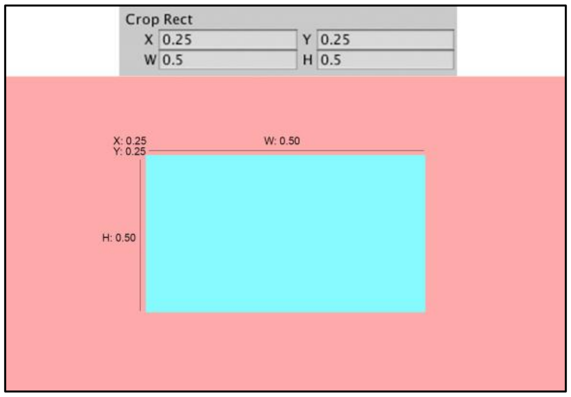
Crop Rect – You can choose to crop the video, entering values as a percentage of the screen size. You should specify the top left corner (X and Y coordinates) and then the width and height of the crop area. For example, the following settings will only capture the center of the screen.

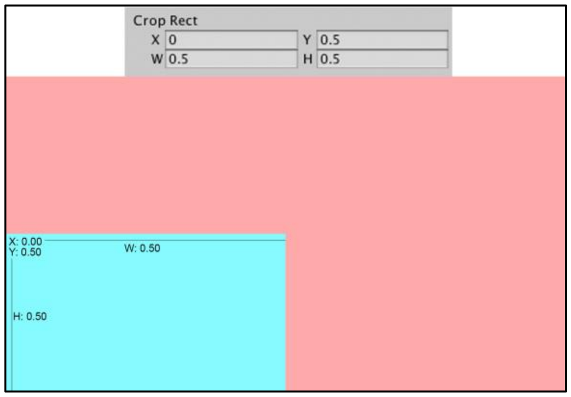

Crop Rect
X 0.25 Y 0.25
W 0.5 H 0.5

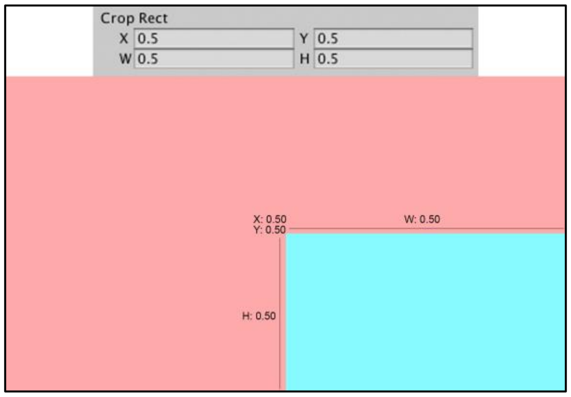
For a screen resolution of 1600x900, and assuming you are capturing at a native scale, this would result in a video that only contains the center of the screen at a resolution of 800x450 pixels. Here is an example of the full screen, with the cropped area shown in red:

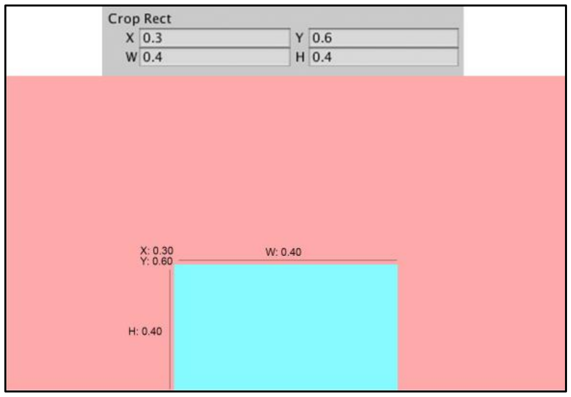


Here are some other examples for reference:


Crop Rect
X 0.25 Y 0.25
W 0.5 H 0.5

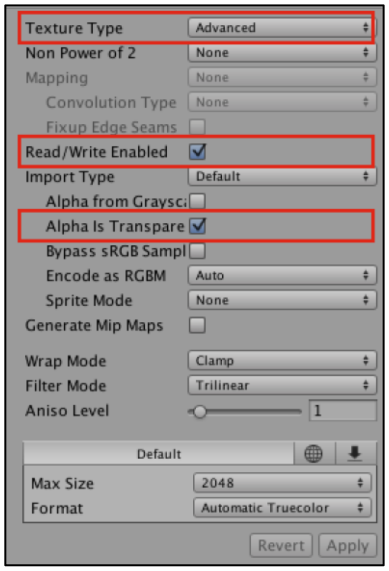

Crop Rect
X 0 Y 0.5
W 0.5 H 0.5


Crop Rect
X 0.5 Y 0.5
W 0.5 H 0.5


Crop Rect
X 0.3 Y 0.6
W 0.4 H 0.4

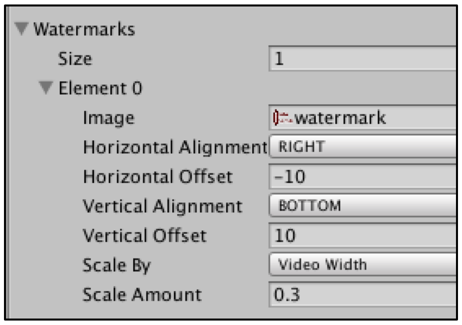
Watermarks – By adding a watermark, you can brand your video with an image or set of images. Watermarks are only added after processing is complete, so they will not appear during gameplay.

To add a watermark, you must first import an image into your project with the following settings (make sure you click Apply after making any changes):



Texture Type	Advanced
Non Power of 2	None
Mapping	None
Convolution Type	None
Fixup Edge Seams	☐
Read/Write Enabled	✓
Import Type	Default
Alpha from Graysc	☐
Alpha Is Transpare	✓
Bypass sRGB Sampl	☐
Encode as RGBM	Auto
Sprite Mode	None
Generate Mip Maps	☐
Wrap Mode	Clamp
Filter Mode	Trilinear
Aniso Level	1
Default	
Max Size	2048
Format	Automatic Truecolor

You can then add that image to the Watermark array.



Watermarks	
Size	1
Element 0	
Image	watermark
Horizontal Alignment	RIGHT
Horizontal Offset	-10
Vertical Alignment	BOTTOM
Vertical Offset	10
Scale By	Video Width
Scale Amount	0.3

Image – The imported texture that will be used as a watermark.

Horizontal Alignment – The image can be LEFT, CENTER, or RIGHT aligned.

Horizontal Offset – The watermark can be shifted horizontally by the specified number of pixels. Useful for making sure the image doesn't touch the left or right side of the video. Positive values will shift the image to the right. Negative values will shift the image to the left.

Vertical Alignment – The image can be TOP, CENTER, or BOTTOM aligned.

Vertical Offset – The watermark can be shifted vertically by the specified number of pixels. Useful for making sure the image doesn't touch the top or bottom of the video. Positive values will shift the image up. Negative values will shift the image down.

Scale By – You can specify a frame of reference when resizing the watermark. For example, you can set the watermark to be scaled by its original size, the width of the video, or the height of the video. This ensures that the watermark size is consistent across multiple screen resolutions. The aspect ratio of the image will always be preserved.

Scale Amount – How much to resize the watermark based on the frame of reference.

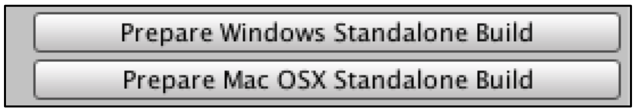
Output Formats – You can choose the type of file that will be saved to disk when the user presses the "Capture Key". The capture framerate and resolution will have a direct impact on the time it takes to process the output files, and the disk space required.

In general, GIF files will be much larger than MP4 or OGG files, and will have a limited color palette. If you are only interested in capturing GIF files, set a lower framerate (15 or lower), and a low resolution (By Scale set to 0.25, or By Value with a height less than 400 pixels or so).

Capture Audio – This option will only appear if the output formats include MP4 or OGG video. If selected, the video file will include the in-game audio track.

Audio Delay – This option will only appear if you are capturing audio. Some video inputs have an audio delay associated with them (for example, capturing through a webcam). This results in AV sync issues which you can manually correct for by adjusting the audio delay. In a majority of cases, this should be left at the default of 0 seconds.

Prepare a Windows or Mac OSX Standalone Build – Before building for a specified target platform, make sure you click the appropriate button in the Flashback Recorder inspector window:

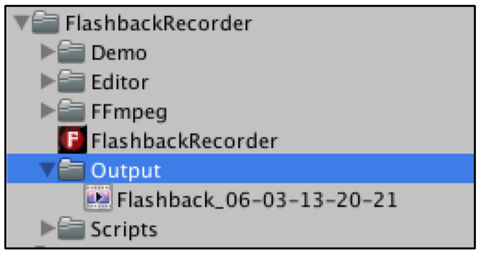

Prepare Windows Standalone Build
Prepare Mac OSX Standalone Build

This will deploy the FFmpeg file to the StreamingAssets folder, so that it is included with your build, and so video can be captured at runtime. See Step 5 for more detail.

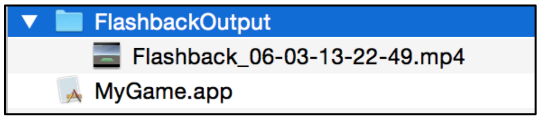
Step 4: Using the Flashback Recorder

When running the game, either in the editor or in a standalone build, the specified amount of video history will be stored in memory. The captured frames will be written to disk when the user presses the button associated with the "Capture Key".

By default, when running in the editor all files will be saved in the "Assets/FlashbackRecorder/Output/" folder in your project root folder with the prefix “Flashback_”.



If running in a standalone build, files are saved in a folder called "FlashbackOutput" at the root level of your executable (or .app file on a Mac).

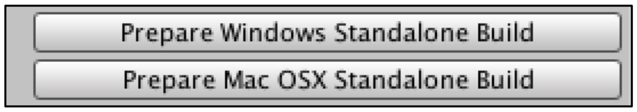


You can change this programmatically using the API. See the “Code API” section for more details.

Step 5: Preparing a Standalone Build

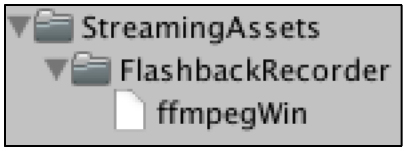
To run a standalone build with Flashback Video Recorder, you will need to make sure the FFmpeg binary is accessible at runtime. To do this, the appropriate file (either FFmpegWin.exe for Windows or FFmpegMac for OSX) must be in the StreamingAssets/FlashbackRecorder folder when you build your game.

There is a simple way to deploy the correct asset – before you build your game, click on “Prepare Windows Standalone Build” or “Prepare Mac OSX Standalone Build” as appropriate.

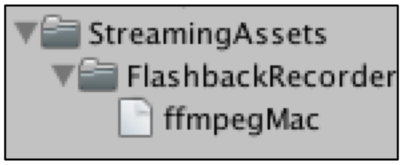

Prepare Windows Standalone Build
Prepare Mac OSX Standalone Build

Clicking either of these buttons will first delete the /Assets/StreamingAssets/FlashbackRecorder folder (if it exists) so that the wrong binary won’t be included with the build, which would needlessly increase your game folder size. Then the appropriate FFmpeg file will be copied from Assets/FlashbackRecorder/FFmpeg to the StreamingAssets folder.

After preparing a Windows standalone build, you will see the following folder structure on your PC:



Or you will see the following folder structure on your Mac:



5. Code API

The core functionality of the Flashback Video Recorder is exposed through the FlashbackRecorder class. This allows you to make changes to the configuration at runtime, and capture video files based on events in your game.

FlashbackRecorder API

The following is a list of all publically accessible methods in the FlashbackRecorder class.

Event Delegate	**Description**
`OnFlashbackFileCreated(string file)`	Executed after a file has been written to disk. Ex: `recorder.OnFlashbackFileCreated += MyDelegate;`

Enumerators	**Description**
`ResolutionType {Native, Scale, Manual}`	Describes how the capture resolution was set. *Native* for using the Screen height and width. *Scale* for resizing the native resolution by a constant. *Manual* for explicitly setting the height and width using pixel values.

Accessors	**Description**
`int GetFrameRate()`	The number of frames captured every second
`int GetNumberOfCapturedFrames()`	The total number of captured frames
`float GetMaxCaptureTime()`	The max allowed capture history, in seconds
`float GetCurrentCaptureTime()`	The current amount of capture history, in seconds
`KeyCode GetCaptureKey()`	The key code that, when pressed, will save the captured frames to disk
`bool CaptureAudio()`	True if the audio is being captured
`float AudioDelay()`	The time (in seconds) the audio will be offset from video
`bool ContinuousCapture()`	True if frames are continuously being captured (ie, Toggle is not selected)
`ResolutionType GetResolutionType()`	Returns *Native*, *Scale*, or *Manual* depending on how the capture resolution was determined

`int GetHeight()`	The capture resolution height in pixels
`int GetWidth()`	The capture resolution width in pixels
`Rect GetCrop()`	Returns the rect used to crop the output video (all values will be between 0 and 1 as percentage of screen size)
`int GetWatermarks()`	Returns the array of applied watermarks
`float GetResolutionScale()`	How much the resolution was scaled (ex: 0.5 if the output is half the size of the native resolution). Dynamically set unless the ResolutionType is set to *Scale*
`bool AutomaticWidth()`	True if the Width was calculated automatically. Can only be true if the ResolutionType is set to *Manual*
`int GetNumberOfPendingFiles()`	The number of files that are being written to disk
`bool SaveMP4()`	True if the output will be written to an MP4 file
`bool SaveGIF()`	True if the output will be written to a GIF file
`bool SaveOGG()`	True if the output will be written to a OGG file

Recorder Interface	**Description**
`bool SetCaptureKey(KeyCode key)`	Set the keyboard key associated with file creation. Pass in KeyCode.None if you wish to call SaveCapturedFrames directly. Returns `true` on success.
`bool SetContinuousCapture(bool continuous)`	If true, frames will be automatically captured. If false, the recorder will need to be toggled on/off. Returns `true` on success.
`bool SetResolutionToNative()`	Sets the capture resolution to the native screen size. Requires that the frame buffer be cleared (i.e., all captured history will be lost). Returns `true` on success.
`bool SetResolutionByScale(float scale)`	Sets the capture resolution to the native screen size multiplied by the specified scale. Requires that the frame buffer be cleared (i.e., all captured history will be lost). Returns `true` on success. Ex: `recorder.SetResolutionByScale(0.5f);`
`bool SetResolutionByValue(int height, int width)`	Allows you to explicitly set the capture resolution. The frame will be rendered, then scaled. If the width is set to -1, then it will automatically be calculated to match the camera's aspect ratio. Returns `true` on success.

	Ex: `recorder.SetResolutionByValue(720, -1);`
`bool CropVideo(Rect crop)`	Crops the output video. All values need to be percentage of screen/height width. Returns `true` on success. Ex: `recorder.CropVideo(new Rect(0, 0, 0.2f, 0.2f));`
`static Rect CheckCropValues(Rect crop)`	Verifies that the crop `Rect` stays within the screen bounds (all values between 0 and 1).
`string GetRootFolder()`	The root folder of the game binary at runtime. Can be used to explicitly set the file output directory.
`bool SetOutputDirectory(string dir)`	Sets the base directory where all captured videos will be stored. Returns `true` on success. Ex: `recorder.SetOutputDirectory(recorder.GetRootFolder() + “myCaptureFolder/”);`
`bool ClearStoredFrames()`	Clears the frame buffer history. Returns `true` on success.
`bool UpdateCaptureSettings(float captureTime, int framerate)`	Sets the frames per second that are captured and how much video history is stored. Requires that the frame buffer be cleared (i.e., all captured history will be lost). Returns `true` on success or `false` if nothing changed.
`bool CaptureAudio(bool saveAudio)`	Sets whether the audio will be recorded and saved with the video output (only for MP4)
`void SetAudioDelay(float amt)`	The time, in seconds, to delay the audio. You can use this to manually adjust any AV sync issues. The default is 0 seconds, which assumes audio and video are perfectly matched.
`void SaveMP4(bool saveMP4)`	Sets whether an MP4 file will be written to disk.
`void SaveGIF(bool saveGIF)`	Sets whether a GIF file will be written to disk.
`void SaveOGG(bool saveOGG)`	Sets whether a OGG file will be written to disk.
`bool StartCapture()`	If `ContinuousCapture()`returns false, toggles video capture on. Returns true if capture started successfully.
`bool StopCapture()`	If `ContinuousCapture()`returns false, toggles video capture off. Returns true if capture was stopped.
`bool IsCapturingVideo()`	Returns true if frames are currently being stored. ie, if `ContinuousCapture()`is true or if video capture was toggled on through `StartCapture()`

`bool CanSaveCapturedFrames()`	When a file is being written to disk, a new video file cannot be saved. Returns true if `SaveCapturedFrames(…)` will execute.
`bool SaveCapturedFrames()`	Will write the captured frames to files using an automatically generated name following the format: “Flashback_MM-dd-HH-mm-ss”. Ex: “Flashback_06-03-14-45-39.mp4” Returns true if the file creation process was started.
`bool SaveCapturedFrames(string filename)`	Will write the captured frames to files using the specified file name. Do not include an extension, since it will be set automatically based on the output formats. Ex: `recorder.SaveCapturedFrames(“myFile”);` Returns true if the file creation process was started.

FFmpeg File Management	**Description**
`void ConfigureFFmpegDirectories()`	Called by the FlashbackRecorder and the FlashbackEditor to set up the locations for the FFmpeg binaries. You should never have to call this explicitly.
`string FFmpegPackageDir()`	Path to the FFmpeg binaries in the Flashback package.
`string FFmpegStandaloneDir()`	Path to the FFmpeg binaries when running standalone.
`string FFmpegWin`	The name of the Windows FFmpeg binary. Defaults to “ffmpegWin.exe”
`string FFmpegMac`	The name of the Mac FFmpeg binary. Defaults to “ffmpegMac”

API Examples

Example 1: Capture Delay

```
// Waits for a specified amount of time before saving the video so that
// the moment of interest is in the middle, not at the end, of the
// captured video
// You can set the FlashbackRecorder Capture Key to “None” in
// the Inspector so the video isn’t captured twice.

void Update(){
	if( Input.GetKeyDown(KeyCode.Space) )
		StartCoroutine( SaveVideo(2.0) );
}

IEnumerator SaveVideo(float delay){
	yield return new WaitForSeconds (delay);
	rec = GetComponent<FlashbackRecorder> ();
	rec.SaveCapturedFrames("myFileName");
}
```

Example 2: Capture Duration

```
// Change the video history to 14 seconds at 30 frames per second
FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
rec.UpdateCaptureSettings(14, 30);
```

Example 3: Using the File Created Delegate

```
// Calls the delegate after files have been captured
void Start(){
	FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
	rec.OnFlashbackFileCreated += FileCreated;
}

void OnDestroy(){
	FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
	rec.OnFlashbackFileCreated -= FileCreated;
}

public void FileCreated(string file){
	//Define your own functionality here
	Debug.Log(“File Created: ” + file);
}
```

Example 4: Set the Capture Resolution by Value

```
// Change the capture resolution to 100 pixels high, and preserve
// the screen aspect ratio
FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
rec.SetResolutionByValue (100, -1);
```

Example 5: Set the Capture Resolution by Scale

```
// Set the capture resolution to 25% of the screen dimensions
FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
rec.SetResolutionByScale (0.25f);
```

Example 6: Changing the Output Directory

```
// Change the root directory and capture a file with a specific name.
// In this case: "C:/myfolder/videos/myVideo.mp4"

void CaptureVideo(){
      string path = "C:/myfolder/videos/";
      string videoName = "myVideo";

      FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
      rec.SetOutputDirectory(path);
      rec.SaveGIF(false);
      rec.SaveMP4(true);
      rec.SaveCapturedFrames(videoName);
}
```

Example 7: Add a Watermark at Runtime

```
// You can add and remove watermarks at runtime, to indicate specific
// levels or stages, for example.

// Must be imported as "Read/Write Enabled"
public Texture2D myTexture;

// Creates a watermark that is aligned at the bottom left of the video
// with 20 pixels from the left edge, and 10 pixels from the top edge.
// It will be scaled so that covers 20% of the total width of of the
// video.
Watermark myWatermark = new Watermark ()
      .SetImage (myTexture)
      .SetHorizonatalAlignment (Watermark.HorizontalAlignment.LEFT)
      .SetHorizontalOffset (20)
      .SetVerticalAlignment (Watermark.VerticalAlignment.BOTTOM)
      .SetVerticalOffset (10)
      .SetScaleMode (Watermark.ScaleType.VideoWidth)
      .SetScaleMode (0.2f);

FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
rec.AddWatermark (myWatermark);

//If you wish to remove the watermark at runtime, you can simply call
rec.RemoveWatermark(myWatermark);
```

Example 8: Crop the Video at Runtime

```
// Crop the video at runtime starting at pixel (400,120) with a
// resolution of 320 by 180 pixels. You can substitute your own values
// as you wish:

int x = 400;
int y = 120;
int width = 320;
int height = 120;

Rect cropRect = new Rect( (float)x/(float)Screen.width,
      (float)y / (float)Screen.height,
      (float)width / (float)Screen.width,
      (float)height / (float)Screen.height );

FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
rec.CropVideo (cropRect);
```

6. FAQ and Troubleshooting

How does Flashback Video Recorder work?

There are three distinct steps to how Flashback Video Recorder captures and saves video output:

Step	**Capture**	**Preprocessing**	**File Output**
Timeframe	Continuously	On Capture Key Press	After Preprocessing
Description	Based on the capture framerate, the recorder saves the raw image data to memory. Each image is captured, scaled, and converted to a byte array to be saved in a queue.	When the user presses the Capture Key, a thread is created with a copy of the queue containing the raw frame data. This raw data is saved to a temporary file on disk.	FFmpeg is launched to convert the raw video file to the specified output formats (MP4 and/or GIF). After file output, the temporary file is deleted.

What is FFmpeg and why is it included with Flashback Video Recorder?

FFmpeg is an open source, cross platform tool that allows users to capture, convert, and stream audio and video files. Flashback Video Recorder uses FFmpeg to convert raw frame data to the supported formats: GIF and MP4.

Neither Flashback Video Recorder, nor its creator, LaunchPoint Games, is associated with FFmpeg in any way. You are free to replace the included FFmpeg binaries with your own version should you choose.

You can learn more about FFmpeg and access the source code here: http://ffmpeg.org and http://www.ffmpegmac.net/

How can I improve performance when using Flashback Video Recorder?

There are two areas of performance that may be impacted by the Flashback Video Recorder: framerate and memory.

In our testing, having the Flashback Recorder enabled could add between 3 to 6 ms per frame, depending on hardware. To reduce this impact, you can lower the framerate or the capture resolution.

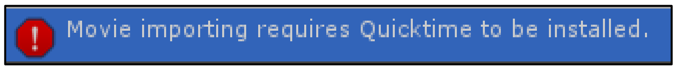
Saving raw video can also have a direct impact on memory usage. A video captured for 6 seconds at 30 frames a second with a resolution of 1280x720 can consume as much as 474.6 MB of memory. To reduce memory usage, select a lower resolution, reduce the framerate setting, and decrease the capture time.

To calculate memory usage in bytes, you can use the following equation:

Mem_Usage = frame_height * frame_width * 3 * framerate * capture_time

Why do I get a QuickTime error when importing MP4 files?

You may get the following error if QuickTime is not installed on your Windows PC and you import MP4 videos in the Project view:



Movie importing requires Quicktime to be installed.

This is to be expected, and the error is harmless. For example, double clicking on an MP4 video in the Project View will launch it with the default player on your PC.

You may choose to install QuickTime, but be aware that Apple no longer officially supports the Windows version. You can find a link to QuickTime, along with Apple's statement, here: https://support.apple.com/kb/DL837

Also, keep in mind that this is a Unity3D error, and is not specific to Flashback Video Recorder. It will be up to Unity to implement another way to handle MP4 videos in the Editor.

Why does my captured video look jittery?

There are several factors that impact the smoothness of the video playback. First, if the framerate of your game drops below the capture framerate, the resulting video will appear to stutter. You will either need to choose a lower capture framerate, reduce the size of the captured video resolution, and/or find other optimizations in your project to ensure a high(er) framerate.

Another factor to take into consideration is that Flashback Video Recorder captures frames several times a second. Adding a motion blur image effect will help smooth the transitions between frames, reducing jerkiness.

How can I reduce the file size of my captured GIFs?

GIFs, by their very nature, are inefficient. The same capture might result in a 200kb MP4 video and a 30 MB GIF. It is strongly recommended that you choose a lower capture framerate (ex: 15 per second), a lower capture resolution (ex: 240x180), and a lower capture history (ex: 5 seconds) to reduce file size.

Why do my GIFs appear low quality?

Each frame in a GIF can only store a maximum of 256 colors. This means that GIFs tend to be lower quality, or have artifacts, that are not present in either raw video or MPEG-4 video.

Video games which have many color shifts or a moving camera can exacerbate the problem.

Can I capture more than 30 frames per second?

We chose the upper limit of 30 frames per second for two main reasons:

First, most video displayed at over 30 frames per second tends to look “cheap” because the motion is too smooth (think of the visual style in Peter Jackson’s Hobbit movies).

Second, most desktop games tend to have a target framerate of 60 frames per second or higher. Increasing the capture framerate requires more processing time and memory.

Since most video is between 20 and 30 frames per second, the additional resource overhead for a higher framerate did not seem to add much benefit.

That being said, there are two different ways to increase the capture framerate, both involve making changes in code.

Option 1: In the FlashbackRecorder.cs file, change the slider range values:

```
//Changes the capture framerate to a maximum value of 60
// frames per second
[SerializeField, Range(1, 60)]
int m_FrameRate = 15;
```

Option 2: You can set the capture framerate programmatically, which is not limited by the variable range:

```
    //Records the last 8 seconds at 60 frames per second
    FlashbackRecorder rec = GetComponent<FlashbackRecorder> ();
    rec.UpdateCaptureSettings (8, 60);
```

Why is video capture not working?

There are several things to try when troubleshooting video capture. Skip to Step 7 if the Flashback Video Recorder works in the editor, but not in your standalone build.

Step 1: If you are developing on a Mac, make sure FFmpegMac has executable permissions by opening a terminal, navigating to your project's root directory, and running the following command:

```
chmod +x Assets/FlashbackRecorder/Ffmpeg/ffmpegMac
```

You should only need to do this once, and it should not be necessary to run a similar command on Windows since FFmpegWin.exe should be executable by default.

Step 2: Make sure the Flashback Video Recorder is attached to a Camera object and ensure it is enabled (the checkbox next to the name of the component is ticked).

Step 3: Make sure a "Capture Key" is selected, or if it is set to "None", then make sure you are actually calling "SaveCapturedFrames".

Step 4: Ensure FFmpeg is in the correct folder. When running in the Unity Editor, FFmpeg must be in the /Assets/FlashbackRecorder/FFmpeg/ directory. If you are running on a Windows machine, you will need the "FFmpegWin.exe" in that folder, and if you're running on a Mac, then you'll need the "FFmpegMac" binary in that folder.

Step 5: Ensure you are looking in the correct output directory. You will need to verify that your game has permission to read, write, delete, and create files in the specified output folder.

Step 6: If running in the editor, make sure you right click in the "Project" view and select "Refresh" since the Asset manager might not import the generated files automatically.

Step 7: If running standalone, make sure you click one of the "Prepare Standalone Build" buttons (for the appropriate platform) in the component window. This will write the FFmpeg binary to the /StreamingAssets/FlashbackRecorder/ folder, which is required for Flashback Recorder to work in the final build.

Why do shadows disappear when I’m using Flashback Video Recorder?

Flashback Video Recorder uses video memory to store frames during capture. Since this is a shared resource, Unity will dynamically allocate shadow texture memory based on the total memory available. In other words, Unity will reduce shadow quality, then remove them entirely, if Flashback Video Recorder is using too much video RAM.

You can learn more here: https://docs.unity3d.com/Manual/LightPerformance.html

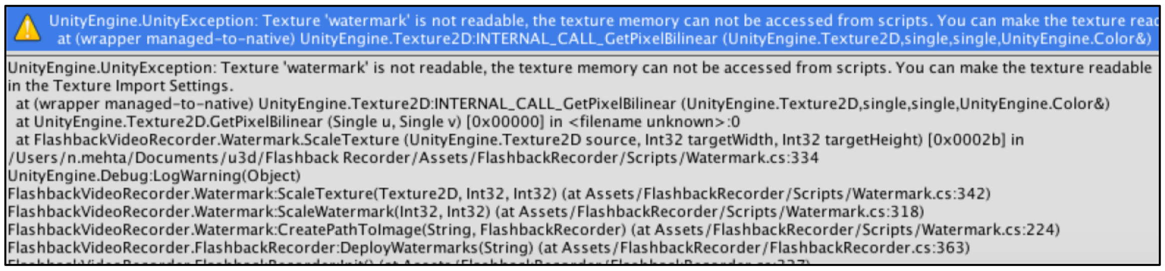
To mitigate this problem (short of upgrading your video card), you can do the following:

1. Reduce capture resolution. Cutting the resolution in half will cut RAM use by 4x.
2. Reduce capture time. The more video that needs to be recorded, the more GPU memory is required.
3. Reduce capture framerate. The fewer frames that need to be captured, the less video memory is required.

Why are my watermarks not working?

There could be a few reasons why your watermarks are not appearing in the output video.

First, verify the Texture2D you want to use as a watermark has the correct import settings. If the texture is not readable, it cannot be accessed from script, and you will get the following exception at runtime:



```
UnityEngine.UnityException: Texture 'watermark' is not readable, the texture memory can not be accessed from scripts. You can make the texture reac
 at (wrapper managed-to-native) UnityEngine.Texture2D:INTERNAL_CALL_GetPixelBilinear (UnityEngine.Texture2D,single,single,UnityEngine.Color&)
```

```
UnityEngine.UnityException: Texture 'watermark' is not readable, the texture memory can not be accessed from scripts. You can make the texture readable
in the Texture Import Settings.
  at (wrapper managed-to-native) UnityEngine.Texture2D:INTERNAL_CALL_GetPixelBilinear (UnityEngine.Texture2D,single,single,UnityEngine.Color&)
  at UnityEngine.Texture2D.GetPixelBilinear (Single u, Single v) [0x00000] in <filename unknown>:0
  at FlashbackVideoRecorder.Watermark.ScaleTexture (UnityEngine.Texture2D source, Int32 targetWidth, Int32 targetHeight) [0x0002b] in
/Users/n.mehta/Documents/u3d/Flashback Recorder/Assets/FlashbackRecorder/Scripts/Watermark.cs:334
UnityEngine.Debug:LogWarning(Object)
FlashbackVideoRecorder.Watermark:ScaleTexture(Texture2D, Int32, Int32) (at Assets/FlashbackRecorder/Scripts/Watermark.cs:342)
FlashbackVideoRecorder.Watermark:ScaleWatermark(Int32, Int32) (at Assets/FlashbackRecorder/Scripts/Watermark.cs:318)
FlashbackVideoRecorder.Watermark:CreatePathToImage(String, FlashbackRecorder) (at Assets/FlashbackRecorder/Scripts/Watermark.cs:224)
FlashbackVideoRecorder.FlashbackRecorder:DeployWatermarks(String) (at Assets/FlashbackRecorder/FlashbackRecorder.cs:363)
```

To make the texture readable, click on the image in the Unity Editor and change the following properties in the Inspector:

Texture Type – Advanced
Read/Write Enabled – Checked
Alpha is Transparency – Checked

Make sure you apply the settings after making any changes.

Second, check the scale and offset values of the watermark. Make sure the Scale Amount is reasonable based on how the watermark is being scaled. For example, if the width of the captured video is 100 pixels, and you are scaling by “Video Width” with a Scale Amount of 0.02, then the watermark will have a width of 2 pixels.

Also ensure the horizontal and vertical offsets do not move the watermark outside the video bounds. For example, if the “Horizontal Offset” is -150, the watermark will be moved to the LEFT by 150 pixels. If the “Vertical Offset” is 90, then the watermark will be moved UP by 90 pixels in the video output.

What if I have improvements or new features for Flashback Video Recorder?

If you have modified Flashback Video Recorder with a new feature or improvement (performance, quality, etc.), then we want to hear from you in order to give proper credit! Just send an email to support@launchpointgames.com and point out that you have made a change to Flashback Video Recorder. Include a brief description of what you have added or changed, as well as why you think its valuable, and someone will get in touch shortly.

If your update is non-trivial and/or teaches us something new, we want to say thank you. We would even consider giving a reward, such as a full or partial refund, depending on the size, complexity, and time it took to make the improvement.

How do I use the Audio Delay control?

In most cases, the audio and video capture will be synced. But in certain cases, where audio and video are being rendered through different mechanisms (such as being processed through a webcam), the captured audio and video might be out of sync.

In cases like this, you can manually adjust the offset by changing the audio delay so the two inputs are in sync again. This will take trial and error since there are a number of external factors to consider (additional processing, etc).

7. Terms & Conditions

These terms and conditions outline the rules and regulations for the use of Flashback Video Recorder. You must purchase Flashback Video Recorder from the Unity Asset Store, or have written permission from LaunchPoint Games, in order to obtain a license to use Flashback Video Recorder in your personal or commercial project. One license is required per seat.

Unless otherwise stated, LaunchPoint Games and/or its licensors own the intellectual property rights for all material found in Flashback Video Recorder.

Flashback Video Recorder contains FFmpeg, which is covered by the GNU Lesser General Public License (LGPL) version 2.1. Full legal terms can be found on the FFmpeg website along will the source code.

FFmpeg License and Legal Considerations: http://ffmpeg.org/legal.html
FFmpeg Download and Source: http://ffmpeg.org/download.html#get-sources

You may use and modify the Flashback Video Recorder source code for use in your projects. However, you may not distribute the package, or any components included with the package covered by this agreement, without written consent from LaunchPoint Games. Flashback Video Recorder is provided "as is" and all other intellectual property rights are reserved.

LaunchPoint Games is not responsible for any damages as a result of using Flashback Video Recorder, including, but not limited to, loss of productivity, loss of data, or legal fees or damages.

If you do not agree to the above terms and conditions, you must immediately stop using Flashback Video Recorder, remove any and all components deployed by the Flashback Video Recorder asset package from your projects, and delete the asset package from your local storage devices, remote storage devices, and version control software.