User Guide

Ultimate Replay 2.0

A simple and effective state-based replay system for Unity

Trivial Interactive

Version 2.1.x

Ultimate Replay 2.0 is a complete state-based replay system ideally suited to kill cams or action replay applications. Due to the state-based nature of the system, it is possible to view replays from any angle or even fly around the scene as playback occurs.

Recommended Uses:

- Action replays in sports of similar games.
- Kill cams / Death cams in shooting games (Demo included).
- Ghost vehicles in racing games (Demo included).
- Multiple angle replays where the same recording is viewed from a number of different vantage points in succession.
- Many more uses…

Features

- Quick and easy setup / integration into existing projects.
- Simple API for replay and playback control means that very little scripting knowledge is required.
- Record and replay as many different objects as you want simultaneously.
- Uses a state-based replay system meaning that you can view playback from different camera angles or even fly around as playback occurs.
- Full support for instantiation or destruction of objects during recording.
- Fully interpolated playback means that you can record at ultra-low frame rates (5fps and less) and retain smooth and accurate replays.
- Supports playback at any speed from ultra-slow motion to 2 or 4x.
- Supports reverse playback which can be used to produce a rewind effect.
- Playback can be paused and resumed at a later date.
- Full playback seek support allows you to jump to any point in a recording.

- Full control over recording frame rate to make sure you capture the best quality recording at the smallest memory cost.
- Recording an object is as simple as attaching a replay component.
- Built-in support for recording transform, audio, particles, animation, and more.
- Easily create your own component recorders to expand the capabilities.
- Memory recording can be setup as continuous or as a rolling buffer configuration.
- File support means that you can create persistent replays for later game sessions.
- Highly optimized file format allows for lengthy replays with minimal file size and high performance streaming.
- ReplayVars allow script variables to be recorded simply by adding an attribute.
- Get useful hints about the storage space requirements for all replay objects.
- Includes example GUI controls for playback manipulation.
- Comprehensive .chm documentation of the API for quick and easy reference.
- Fully commented C# source code included.

Contents

Upgrade Guide ... 7
- Replay Manager ... 7
- Replay Scene ... 7
- Replay Storage Targets ... 7
- Replay Behaviour ... 8

Quick Start ... 9

Replay Considerations ... 13

Replay Concepts ... 14
- Replay Manager ... 14
- Replay Handle ... 14
- Replay Identity ... 14
- Replay State ... 14
- ReplaySnapshot ... 15
- Replay Scene ... 15
- Replay Storage Target ... 15
- Replay Recorder Component ... 15

Replay a Game Object ... 16
- Game Object Hierarchy ... 16
- Main API ... 18
  - BeginRecording ... 18
  - StopRecording ... 18
  - IsRecording ... 19
  - SetPlaybackTime ... 19
  - SetPlaybackTimeNormalized ... 20
  - SetPlaybackDirection ... 20
  - SetPlaybackTimeScale ... 21
  - GetPlaybackTime ... 21
  - BeginPlaybackFrame ... 21
  - BeginPlayback ... 22
  - StopPlayback ... 23
  - IsReplaying ... 23
  - AddPlaybackEndListener ... 23

Replay Settings ... 24

Replay Scene ... 28

Replay Scene Mode.......... 28
Replay Preparers.......... 28
Custom Replay Preparer 29
Replay Prefabs.......... 31
Registering Replay Prefabs 31
Register via script 32
Instantiating Replay Prefabs 33
Using Replay Scene.......... 33
Using Replay Manager.......... 33
Destroying Replay Prefabs.......... 34
Replay Storage Target.......... 37
Replay Memory Target.......... 37
Replay Stream Target.......... 38
Replay File Target 38
Custom Replay Storage Target.......... 39
Replay Object.......... 40
Inspector.......... 40
Identity Transfer 40
Replay Behaviour.......... 42
Replay Messages 42
OnReplayStart.......... 42
OnReplayEnd 42
OnReplayPlayPause 42
OnReplayReset.......... 43
OnReplayCapture 43
OnReplayUpdate.......... 43
OnReplayEvent.......... 43
OnReplaySpawned 43
Replay Events 44
Replay Methods.......... 45
Replay Variables 46
Recorder Components 48
Replay Transform.......... 48
Create Menu.......... 48
Inspector 48
Replay Enabled State 50

Create menu 50
Inspector 50
Replay Component Enabled State 50
Create menu 50
Inspector 50
Replay Animator 52
Create menu 52
Inspector 52
Replay Particle System 54
Create menu 54
Inspector 54
Replay Audio 54
Create menu 54
Inspector 54
Replay Material Change 56
Create menu 56
Inspector 56
Replay Material 57
Create menu 57
Inspector 57
Replay Line Renderer 59
Create menu 59
Inspector 59
Replay Trail Renderer (Unity 2018.2 or newer) 60
Create menu 60
Inspector 60
Custom Recorder Components 61
Replay Controls 63
Record Mode 63
Playback Mode 64
Free Cam Mode 65
Replay Techniques 66
Replay Animation 66
Replay Ragdolls 66
Killcams 67
Replay Statistics 69

Storage Statistics ... 69
Replay Humanoid Configurator ... 71
Integration ... 73
Pooling Support ... 73
How do I… ... 74
Get the replay duration? ... 74
Set playback time? ... 74
Replay in reverse? ... 75
Replay in slow motion? ... 75
Quickly test my scene? ... 75
Create a killcam? ... 75
Create a ghost vehicle? ... 76

Upgrade Guide

If you are upgrading from the original Ultimate Replay asset, then we would like to thank you for you continued support and recommend that you take a look at the following section to see how the asset has evolved. We have designed Ultimate Replay 2.0 from the ground up in order to support many requested features and also offer a more optimized and capable asset. With that said, many of the original concepts of Ultimate Replay still exist such as replay components, Replay objects etc. but you may notice some changes.

Replay Manager

The ReplayManager was a core concept in the original asset and indeed it still is the heart of the asset although in a slightly different form. Previously, the ReplayManager was implemented as a MonoBehaviour component which needed to be added to a scene object in order to work correctly. In version 2.0, the ReplayManager is now a completely static API and any required scene components will be created as required automatically. Most of the original ReplayManager methods still exist in some form, although now they either accept or return a ReplayHandle argument which is used to support an unlimited number of simultaneous replay operations. There is more information later in the document but essentially begin operation such as BeginRecording and BeginPlayback will return a ReplayHandle. You will need to store this object and pass it as an identifier every time you want to change the state of a replay operation or query state information.

In the original asset, the RelayManager also contained the global replay settings such as record FPS and replay prefabs collection and more. This has now been moved into a global asset which is accessible via the menu 'Tools -> Ultimate Replay -> Settings'.

Replay Scene

A new concept in Ultimate Replay 2.0 is the ReplayScene which is used to specify a collection of ReplayObject s which should be associated with a record or replay operation. In the original asset, all ReplayObject s in the active scene would be used in recording and playback which was fine since it was only possible to have one replay operation running at any given time. In version 2.0, you will now have the option of providing a ReplayScene instance to specify which ReplayObject s you want to be affected. There are many static construction methods for the ReplayScene class which makes it easy to get all ReplayObject s in a specific scene or you can manually add ReplayObject s if required. If you do not provide a ReplayScene instance when calling BeginRecording or BeginPlayback, then all ReplayObject instances in the active scene will be used. Take a look at the Replay Scene section for more information.

Replay Storage Targets

The concept of replay storage targets still exist in Ultimate Replay 2.0 however this time they are implemented a little different. In the original asset, storage targets were implemented as behaviours and as a result they had to be attached to a game object, usually next to the ReplayManager component. In version 2.0, the storage targets are now just normal C# objects and can be created using the constructor and static helper methods in some cases. This does mean that some storage targets such as the ReplayFileTarget will need to be manually disposed when no longer needed but it allows for far greater control.

Replay Behaviour

Another notable change is to the ReplayBehaviour component. Previously, this component was used to create custom replay components as well as record events and declare replay variables. In version 2.0, the ReplayBehaviour still exists although some of its functionality has been refactored out. The OnReplaySerialize and OnReplayDeserialize methods that you may already be accustomed to no longer belong the ReplayBehaviour class. Instead they have been moved out to a new type called ReplayRecordableBehaviour. The main reason for this change is so that users can inherit from ReplayBehaviour without needing to implement the serialize methods. Creating a ReplayBehaviour script can be useful for querying the record or replay state, recording variables, events, methods and more as well as receiving various replay events like OnPlaybackStart.

Quick Start

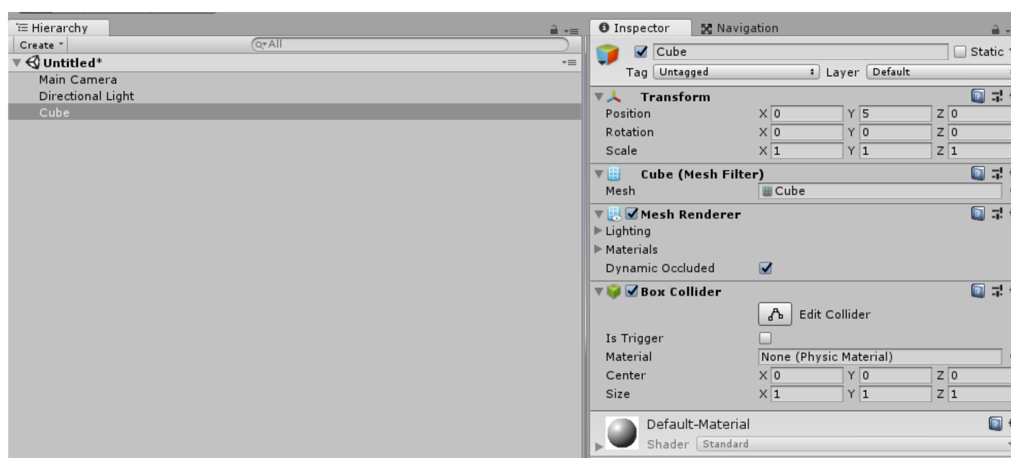
This section will walk you through the bare minimum setup required in order to get a simple replay system up and running as quick as possible. The walkthrough will assume a basic understanding of core replay concepts such as the ReplayManager and RecorderComponents. If you are unfamiliar with these terms then we highly recommend taking a look at the ReplayConcepts section as a minimum to get a basic understanding.

> **Note:** *The completed demo scene which will be created by following the guide below is included in the asset for convenience. You can find this scene at the path 'Assets/Ultimate Replay 2.0/Demo/QuickStart.unity'*

This guide will assume that you have successfully imported the Ultimate Replay 2.0 asset into your Unity project and that you are starting with a blank scene 'File -> New Scene'. If you have trouble importing the asset for any reason then please contact us for support. Contact information can be found at the end of this document.

1. First of all, we will need a game object which will be recorded and replayed by the replay system. This should be a moving object (static objects work fine but will make for a dull replay) so this example will use a simple cube object with a physics rigid body and collider attached.

   First, we will add a cube object to the scene by going to 'GameObject -> 3D Object -> Cube' and position the object at **(0, 5, 0)** in the scene. We will also set the rotation of the cube to **(50, 60, 0).** The purpose of this is that the cube will tumble and roll when colliding with the ground plane making for a more interesting replay.

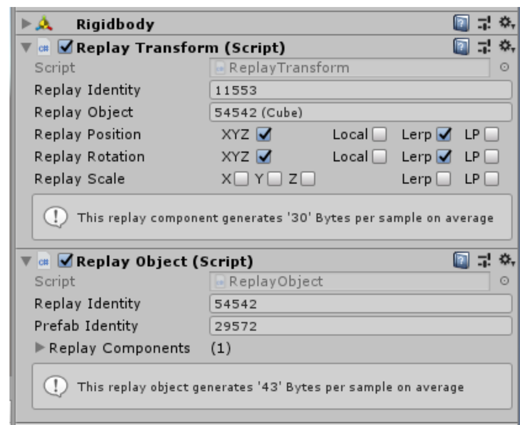
   

   With the cube object still selected, add a RigidBody component by going to 'Add Component -> Physics -> Rigidbody'. This will allow the cube to fall with gravity so that we have a moving object to record.

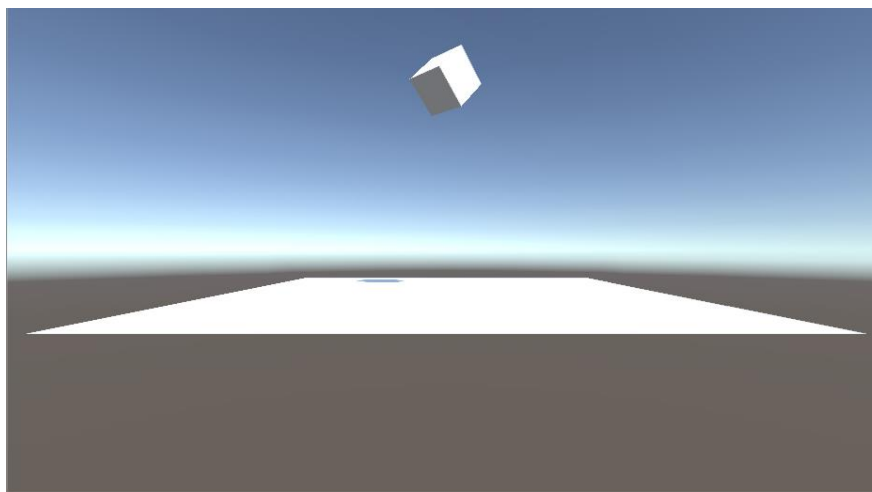
2. The next step is to make the object replayable by adding a recorder component. Ultimate Replay 2.0 offers a number of built-in recorder components which can be used to replay different elements of an object. For this example, we will be interested in using the

ReplayTransform component which is used to record and replay the Unity transform component.

Add a ReplayTransform component to the cube object by going to 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Transform'. This will cause a ReplayTransform and ReplayObject component to be attached to our cube object:



3. Once we have setup the cube object, we will now add a ground plane to the scene so that the cube has something to collide with. Go to 'GameObject -> 3D Object -> Plane' to create a ground plane. Make sure the plane position is set to **(0, 0, 0).**

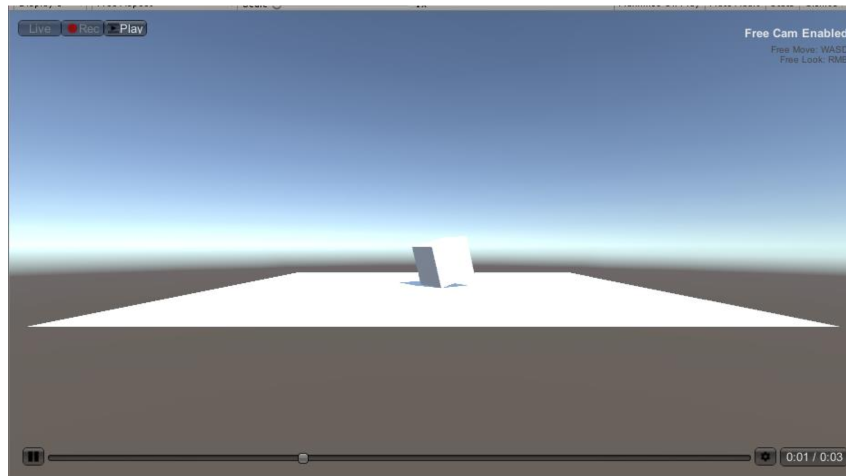


4. That is all the setup done. We could now start recording using the Ultimate Replay API and we would capture a valid replay of the cube falling and tumbling as it contacts the ground plane. There is however an easier way to test the scene without the need for scripting. The ReplayControls allow you to record and view replays using a basic UI interface. We will make use the ReplayControls component in this example as it is the easiest way to test a scene.

   Add a ReplayControls component by going to 'Tools -> Ultimate Replay -> ReplayControls'. This will add a new game object to the scene named 'ReplayControls' which has a ReplayControls script component attached to it. This script component will display the replay UI using the Unity legacy immediate mode GUI.

5. We can now test our replay scene by entering play mode in the editor to start the game. The ReplayControls will start recording automatically as soon as the game starts and the cube

will fall to the ground and tumble. Once the cube has settled, click the 'Play' button of the ReplayControls UI in the top left corner to enter playback mode. You will see that the replay begins and the cubes actions are replayed smoothly and accurately. You can also use the playback slider to jump to different positions in the replay as well as the settings menu which contains options for playback speed and direction.



Note: *While in playback mode, you can use the free-cam controls of WASD + RMB to fly the camera around the scene as the replay is running.*

Congratulations! You have successfully setup your first replay system using Ultimate Replay 2.0 and can now move on to bigger and better things. At this stage it may be worth taking a look at the following sections to get a better understanding of the asset and how it works:

- Replay Considerations – There are a few things to consider when using Ultimate Replay 2.0 in your game. This section will highlight things that you should be aware of.
- Replay Concepts – Learn about the key concepts used by Ultimate Replay 2.0
- Replay a Game Object – Learn how to record and replay a game object in depth.
- Replay Manager – Learn more about the heart of the replay system.
- Replay Prefabs – Need to instantiate or destroy objects in your game? This section will tell you how it can be achieved.
- Replay Recorder Components – Take a look in detail at the built-in recorder components offered by Ultimate Replay 2.0. Also learn how you could create your own recorder components.

There are also a number of demo scenes included with the asset which may be worth taking a look at. All demo scenes are located inside the demo folder at 'Assets/Ultimate Replay 2.0/Demo':

- CubeTest – A basic demo scene that spawns a large amount of physics cubes over a period of a few seconds. A useful demo if you want to instantiate or destroy recorded game objects.
- ReplayGhostVehicle – This demo shows how a ghost vehicle in a racing game could be implemented. This demo uses more advanced techniques like multiple simultaneous record and replay operations, replay identity transfer and multi storage management.

- ReplayKillcam – Demonstrates how a first person killcam could be implemented in a multiplayer game. This demo features ragdoll, particle, audio recording and more. It also demonstrates how a replay can be viewed from the other players perspective (as the shooter) for a true killcam point of view.

Take a look at the How do i? section which will answer common questions about the asset. Can't find the answer to a question? Feel free to contact us and we could add your questions along with the answer to the documentation for all to benefit (Contact details at the end of this document).

Replay Considerations

Ultimate Replay 2.0 uses a state-based storage technique to record game data at fixed intervals as specified by the user. These data samples are known as snapshots and can accurately describe the state of a scene at a given time offset known as the time stamp. Ultimate Replay 2.0 uses these snapshots to recreate the scene during playback using a slideshow effect to show these states quickly in order to give the illusion of seamless animation. There are a few things to consider when implementing Ultimate Replay 2.0 into your game project:

- **Replays are state-based:** Ultimate Replay 2.0 uses a state-based approach to record and replay the scene. This means that a number of snapshots will be captured per second during recording which contain enough state data to be able to recreate the scene at a later time. As a result, there are a few things to consider:
    - Replays are created by restoring these snapshots in quick succession in order to create a smooth playback sequence from a series of snapshots.
    - It is not possible to store recordings as popular video formats such as MP4 because the screen pixel data is simply not captured.
    - Replays are rendered in realtime by the active camera allowing you to switch cameras during playback, add or remove post processing effects during, create a highlight reel with multiple camera angles and more.
- **Scripts don't run during playback:** In order for the replay system to accurately recreate the scene as it was recorded, scripts may be disabled so that they cannot move or otherwise manipulate objects during playback which would cause inaccurate results. Only scripts attached to a replay object will be disabled so this will not affect standalone game systems such as game managers which are not repayable. Scripts will be enabled and disabled automatically by the replay system via the active replay preparer. Note that you can also disable or modify this behaviour if required by creating a custom replay preparer script. See the Replay Preparers section for more information. Note that scripts deriving from ReplayBehaviour are treated specially and will be allowed to run during playback.
- **Physics components are inactive during playback:** Physics components will also be disabled during playback mode to prevent inaccurate replays. The reason for this is that much like scripts, physics components can cause objects to be moved during playback as a result of rigid body updates or collision resolution. This would cause playback to become inaccurate so the components are disabled or deactivated using the active replay preparer.
- **Storage targets cannot be used in multiple operations simultaneously:** If you attempt to start more than one record or replay operation using the same storage target, you will receive an exception as this behaviour is not supported. By design, storage targets can only be in write or read mode at any given time. Even then, only a single replay operation can access the target to avoid many seek operations which could cause potential performance issues, or worse, multiple write operations which could corrupt the data stream.

Replay Concepts

This section will cover some of the essential concepts used by Ultimate Replay 2.0 and how they are used to affect recording and replay behaviour.

Ultimate Replay 2.0 is a state-based replay system meaning that the scene is sampled multiple times during recording to create a series of snapshots. These snapshots contain the necessary state data of all ReplayObject s in the scene such as position and rotation which are used during playback to reconstruct the scene exactly how it was during recording. By reconstructing these snapshots in order very quickly, it is possible to create the illusion of a seamless replay, a bit like a flipbook animation. This state-based approach has a few benefits over traditional screen recording techniques:

1. The replay can be viewed from different angles or cameras since the replays are actually rendered as they are running. You can also move the camera during playback and apply new post processing effects to change the appearance of a replay if required.
2. The state data recorded per scene sample is actually a very small amount of data when compared with screen recording techniques where the screen pixels need to be stored. This means that memory usage and replay file sizes can be very small allowing for long and complex replays to be recorded, especially with the compression techniques offered by Ultimate Replay 2.0.

Replay Manager

The replay manager is the heart of the replay system and is main interface used to interact with Ultimate Replay in your game. It contains all the methods to start, stop, modify and query replay operations and will generally be the only point of contact of the Ultimate Replay 2.0 API unless you are an advanced user. Take a look at the Replay Manager section for more detail.

Replay Handle

A ReplayHandle is returned by any begin operations such as BeginRecording or BeginPlayback and is used to uniquely identify a replay operation. You will need to pass a replay handle instance any time you need to query or change the state of a replay operation via the ReplayManager . Replay Handles were added to allow an unlimited number of simultaneous replay operations to be supported.

Replay Identity

A ReplayIdentity is a unique serialized id value that all replay components are assigned by the replay system when created. The ReplayIdentity is used to identify each replay object, component, data segment etc so that the recorded data is restored to the correct objects. ReplayIdentities will be generated automatically by the replay system but there are occasions when you may like a particular replay object to take on the identity of a different object. This can be useful to record data from one object but replay on a different object as used in some use cases such as ghost vehicles. Take a look at identity transfer for more information.

Replay Scene

Replay State

A replay state is a storage device that is passed to all recorder components and is used to store primitive data types into a data stream. Some Unity types such as Color and Vectors are also

supported for ease of use. Any time you need to manually record or restore any replay data, a ReplayState will be passed which you can use to write to or read from.

ReplaySnapshot

A ReplaySnapshot contains the entire state data of the active ReplayScene at a given time stamp. Multiple snapshots stored in order can be used to recreate the scene over a period of time much like a slideshow or keyframe animation. These snapshots are used as a higher-level storage device and can be stored to and fetched from a ReplayStorageTarget directly. You will not need to deal with ReplaySnapshots directly but it is a good idea to understand what they are and their purpose.

Replay Scene

A ReplayScene represents a collection of ReplayObject s which should be used in a record or replay operation. By default, Ultimate Replay 2.0 will use all ReplayObject s in the active scene for all record and replay operations unless you manually specify a custom replay scene. Replay scenes are also responsible for a process called state preparation which is required for playback. Essentially, any components that could interfere with playback such as rigid bodies, colliders or scripts need to be prepared before playback commences.

Replay Storage Target

A ReplayStorageTarget is an end storage device used to store the data that is recorded by the replay system. The data that is generated by the replay system is in the form of a ReplaySnapshot which is then flushed to the active ReplayStorage when it should be stored. The StorageTarget could be anything from a memory storage target to file storage or more.

Replay Recorder Component

Recorder components are used to record and replay an associated Unity component. For example, the ReplayTransform component is intended to record and replay the Unity Transform component of a specific object. There are many other recorder components built and it is also possible to create your own recorder components for unsupported or 3rd party components.

Replay a Game Object

In order to create a replay for your game using Ultimate Replay 2.0, you will need to decide which game objects should be replayable so that the necessary replay components can be added. The objects which are replayable will depend largely on the specific game but generally you will want to add replay components to any game objects which move, are animated, have effects such as particle systems etc. There are some exceptions but again this will depend on your game. An example would be an animated crowd in a sports game. It may not be necessary to record the whole crowd as you could just continue playing the animations during the replay.

Good candidates for replay objects are game objects that are core to the gameplay such as the player, enemies, projectiles, effects, sound effects, etc. Basically, any game object which is not stationary and whose behaviour does not consist of static continuous animation or similar. In some cases, you may need to replay custom gameplay elements which are not supported by built in components. For example: a GUI overlay which displays chat text or similar.

Once you have determined which game objects should be replayable, you then need to add the appropriate replay components. There are many replay components built in to Ultimate Replay 2.0 which are covered in the Recorder Components section. For game objects that move, you will want to add a ReplayTransform component so that the game object transform will be recorded and replayed. For Animator components you will want to add a ReplayAnimator component and so on.

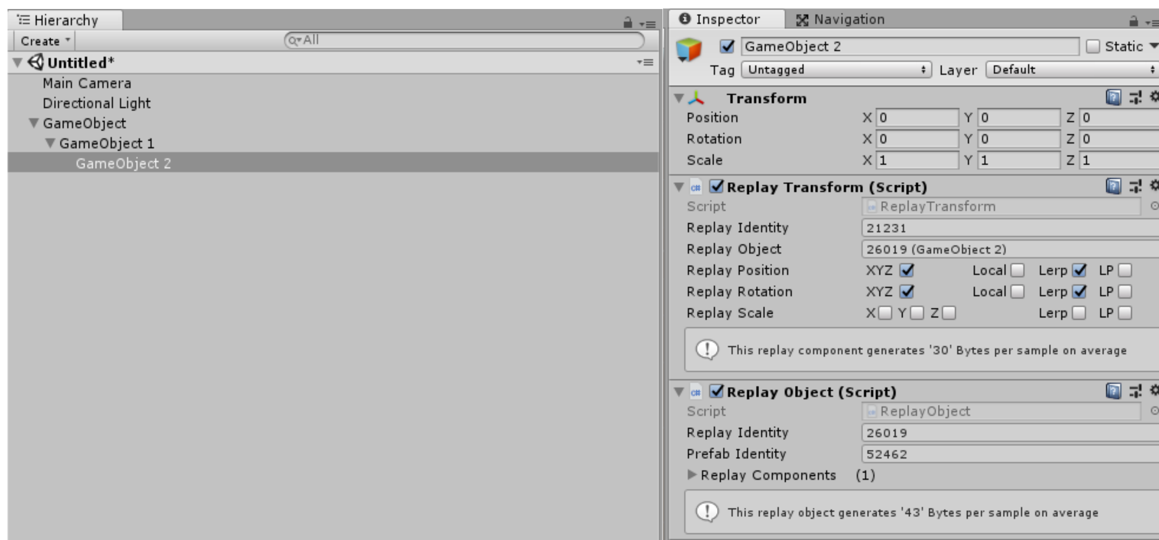
> **Note:** *A ReplayObject component will automatically be added when you attach a replay component to a game object. This is a required component and will manage one or more recorder components.*

Once you have added the necessary replay components, you should have a working replay system which will record and replay the state of all observed components. You just need to use the ReplayManager API to start and stop recording and playback. To quickly test out your newly added replay components, you can use the built in Replay Controls interface which will handle the record and replay API calls while providing a simple and easy to use interface.

Take a look at the included demo scenes to see how various replay techniques can be implemented and which replay components are used to do so.

Game Object Hierarchy

If you have a game object with one or more children that need to be recorded then there are some things to consider. It is important that a ReplayObject component is attached to the highest-level game object that has a replay component attached. The ReplayObject component will usually be added automatically when attaching a replay component to a game object but in some cases it may not be added at the correct level. Take the following examples:



As you can see from the screenshots, 'GameObject' 2 has a ReplayTransform component added which causes a ReplayObject component to be added automatically at the same level. This is fine because there are no replay components attached to game objects higher in the hierarchy.

If we wanted to add a ReplayTransform component to 'GameObject 1' then this would cause an issue. If we selected 'GameObject 1' and then went to 'Tools -> Make Selection Replayable -> Replay Transform' then we would end up with the same components as 'GameObject 2'. The problem is that both objects will have a ReplayObject component attached which is inefficient and may cause issues. In scenarios like this, it is enough to simply remove the ReplayObject component attached to 'GameObject 2' and now the hierarchy setup is perfectly valid. The highest-level replay component in the hierarchy has the only ReplayObject component attached which will now manage both ReplayTransform components.

> **Note:** *An exception to this rule is if 'GameObject 2' in the above example was a prefab instance, in which case multiple ReplayObjects would be the way to go in order to support dynamic creation and destruction.*

For animated objects with a humanoid structure you can use the Replay Humanoid Configurator in order to add ReplayTransform components to all bones in the hierarchy.

Replay Manager

The Replay Manager is the main interface for Ultimate Replay and is used to control and query all replay operations using the static API. If you are coming from Ultimate Replay 1.0 you may already be familiar with the Replay Manager however in version 2.0 there are some major differences. It may be worth taking a look at the Upgrade Guide section if you haven't already as this section covers some of the major changes from the original asset.

The ReplayManager is a type that defines the main API of Ultimate Replay 2.0 and will be used by your games scripts to control the replay system. The ReplayManager is implemented as a static API for ease of use, meaning that you can call its methods from any script without requiring an object reference. This means that there is no scene representation of the ReplayManager unlike the original asset. This means that scene changes do not cause issues or interfere with the replay system in any way.

Main API

The following section will cover many essential or useful methods of the ReplayManager that you may need to use in your game. Note that not all methods are covered in this section and it is recommended that you take a look at the included scripting reference for a full API overview.

BeginRecording

Use this method to start a new recording operation using the specified ReplayScene and ReplayStorageTarget. Ultimate Replay 2.0 supports any number of simultaneous record operations and the returned ReplayHandle is used to identify the operation.

C# Code

```
public static ReplayHandle BeginRecording(ReplayStorageTarget,
ReplayScene, bool, bool, ReplayRecordOptions)
```

- ReplayStorageTarget **(recordTarget):** The ReplayStorageTarget used to store the recorded data. When null is passed, the ReplayManager.DefaultStoragetTarget is used.
- ReplayScene **(recordScene):** The ReplayScene used to record data from. The scene cannot be empty unless allowEmptyScene is enabled. When null is passed, ReplayScene .CurrentScene is used for recording.
- **Bool (cleanRecording):** Should the storage target be cleared before recording starts. Default is true.
- **Bool (allowEmptyScene):** Can an empty scene be passed to this method. When enabled, you can pass a ReplayScene instance with no registered ReplayObject s. This may be useful if you intend to instantiate one or more ReplayObject s during recording. See replay prefabs for more information. Default value is false.
- **ReplayRecordOptions (recordOptions):** The replay record options use to specify many record preferences. When null is passed, the global project options editable via the settings window will be used.

StopRecording

Use this method to stop an already running record operation that was previously started using BeginRecording.

C# Code

```
public static void StopRecording(ref ReplayHandle)
```

- ReplayHandle **(recordHandle):** The ReplayHandle of the record operation that should be stopped. The handle must be passed by reference and will be disposed by the replay system meaning it should no longer be used.

IsRecording

Use this method to determine whether a recording operation is currently running with an associated replay handle.

C# Code

```
public static bool IsRecording(ReplayHandle)
```

- ReplayHandle **(recordHandle):** A ReplayHandle to check for running record operations. Disposed replay handles can be passed which will cause a value of false to be returned.

SetPlaybackTime

Use this method to seek to a specific time in the replay. The time value is in seconds and must be between 0 and the recording duration. The specified ReplayHandle should be associated with a valid running or pause playback operation.

C# Code

```
public static void SetPlaybackTime(ReplayHandle, float,
PlaybackOrigin)
```

- ReplayHandle **(recordHandle):** The ReplayHandle of the playback operation that should have its playback time offset changed.
- **Float (playbackTimeOffset):** The time offset value in seconds used to calculate the final seek time based upon the 'origin' parameter. This time value acts as a negative offset when an origin value of 'End' is specified.
- **PlaybackOrigin (origin):** Used to specify the replay marker where the time offset should be relative to. By default, this value is set to 'Start' meaning that the specified time value is absolute. Options are:
    - **Start:** The specified time offset value is relative to the start of the recording, or the 'zero' time stamp.
    - **Current:** The specified time offset value is relative to the current replay position. For example: If the current replay position is '5 seconds' and a time offset value of '2 seconds' is specified, then playback will seek to the '7 second' mark, assuming that the time value is within the bounds of the recording duration. Negative time offset values can be used with this origin option if required.
    - **End:** The specified time offset value will be taken as a negative value (A positive value must be specified) and will represent the amount of time in seconds from the end of the recording. For example: If the recording duration is '10 seconds' in length and a time offset value of '3 seconds' is specified, then playback will seek to the '7 second' mark.

SetPlaybackTimeNormalized

Use this method to seek to a position in the replay using normalized offset values. This method is useful if you want to seek though a replay without taking the duration of the recording into account.

C# Code

```
1  public static void SetPlaybackTimeNormalized(ReplayHandle, float,
2  PlaybackOrigin)
```

- ReplayHandle **(handle):** The ReplayHandle of the playback operation that should have its time offset changed.
- **Float (playbackNormalizedOffset):** A normalized value between 0-1 which is used to represent the offset value between to replay marker points.
- **PlaybackOrigin (origin):** Used to specify the replay marker where the normalized offset value should be used to represent the time offset value. By default, this value is set to 'Start' meaning that normalized offset indicates the absolute time offset relative to the replay duration. Options are:
  - **Start:** The specified normalized offset is taken from the start of the recording. This means that a value of '0' represents the start of the recording and a value of '1' represents the end of the recording. Passing a value of '0.5' will cause playback to seek to the middle of the replay.
  - **Current:** The specified normalized offset is taken from the current replay position. For example: If the current replay position is set to the absolute middle of the replay, passing a value of '0.5' will cause playback to seek to the ¾ mark in the replay since the normalized value represents the offset between the current and end replay markers.
  - **End:** The normalized offset is used to represent the offset from the end of the recording. This means that a value of '0' represents the last frame in the replay (Or a time stamp equal to the duration of the recording) and a value of '1' represents the very start of the recording. I.e. the offset is normalized and inverted.

SetPlaybackDirection

Use this method to change the direction of a replay. Useful if you want to add rewind effects or simply view a replay in reverse.

C# Code

```
1  public static void SetPlaybackDirection(ReplayHandle,
2  PlaybackDirection)
```

- ReplayHandle **(handle):** The ReplayHandle of the playback operation that you wish to modify. The handle should represent a valid and active playback operation started using BeginPlayback.
- **PlaybackDirection (direction):** The direction that you want playback to run. The default value if 'Forward' which will play the replay in the normal forward direction. Options are:
  - **Forward:** Play the replay in the normal forward direction.
  - **Backward:** Play the replay in reverse direction.

SetPlaybackTimeScale

Use this method to change the playback speed. The timescale represents the speed that a replay will run where a value of ‘1’ is standard playback speed and a value of ‘2’ is twice the standard playback speed. Values between 0-1 will cause playback to be slowed where a value of ‘0.5’ represents half speed.

C# Code

```
1  public static void SetPlaybackTimeScale(ReplayHandle, float)
```

- ReplayHandle **(handle):** The ReplayHandle of the playback operation that you want to modify the playback speed of. The handle should represent a valid and active playback operation started using BeginPlayback.
- **Float (timescale):** A time scale value used to specify the playback speed of a replay. The default value of ‘1’ represents the standard playback speed. Note that negative values can be specified to cause reverse playback as an alternative to SetPlaybackDirection.

GetPlaybackTime

Use this method to retrieve the ReplayTime information for a specific playback operation. The ReplayTime result contains information such as absolute playback time stamp, playback frame delta and current time scale.

C# Code

```
1  public static ReplayTime GetPlaybackTime(ReplayHandle)
```

- ReplayHandle **(handle):** The ReplayHandle of the playback operation to query. The handle should be associated with a valid and active playback operation started by calling BeginPlayback.

BeginPlaybackFrame

Use this method to enter playback mode in fixed frame mode. Fixed frame mode means that playback mode is active but the replay will not update meaning only a single playback frame will be shown. Full playback operations such as seeking are supported. Useful for showing a fixed frame of a replay. Paused playback is an alternative.

C# Code

```
1  public static ReplaHandle BeginPlaybackFrame(ReplayStorageTarget,
   ReplayScene, bool, ReplayPlaybackOptions)
```

- **ReplayStorageTarget (replaySource):** The ReplayStorageTarget used as the playback source. The replay will be streamed by fetching snapshots from the storage target on demand. Pass ‘null’ to use the default storage target.
- ReplayScene **(playbackScene):** The ReplayScene used to specify which scene objects should be replayed. This allows you to mask the captured data if required so that some objects are not replayed, even though they were recorded. Pass the default value of ‘null’ in order to use all active ReplayObject s in the current scene.
- **Bool (allowEmptyScene):** A value which determines whether an empty ReplayScene can be passed or not. An empty ReplayScene will have no valid ReplayObejcts to record but may still cause some metadata to be recorded which may be undesirable. Passing ‘false’ will

cause the method to thrown an exception if the specified ReplayScene does not contain any ReplayObejcts. In some scenarios, it may be desirable to use and empty ReplayScene . For example: When dynamically spawned objects are used, an empty scene may be allowable in which case you can pass 'true' to disable empty scene exceptions.

- **ReplayPlaybackOptions (playbackOptions):** The options used by the playback service which determines much of the playback behaviour such as frame rate and end behaviour. The default value of 'null' causes the global project playback options to be used which are accessible via the settings window. You can also programmatically create the playback settings per playback operation if required.

BeginPlayback

The main method used to start a previously captured replay. This will cause the replay system start a new playback operation using the specified storage target as the data source and the specified ReplayScene to determine which objects are replayed. The specified ReplayScene does not necessarily have to match up to the storage target although a harmless warning may be generated if not. In some cases, you may record 2 objects but only want to replay 1 object. This can be achieved by using the ReplayScene to create a mask by removing 1 of those objects. Even though the storage target will contain information about 2 objects, only the objects added to the ReplayScene will be replayed. This method will return a ReplayHandle value which will be used for all subsequent state change and query operations.

C# Code

```
1  public static ReplaHandle BeginPlayback(ReplayStorageTarget,
   ReplayScene, bool, ReplayPlaybackOptions)
```

- **ReplayStorageTarget (replaySource):** The ReplayStorageTarget used as the playback source. The replay will be streamed by fetching snapshots from the storage target on demand. Pass 'null' to use the default storage target.
- ReplayScene **(playbackScene):** The ReplayScene used to specify which scene objects should be replayed. This allows you to mask the captured data if required so that some objects are not replayed, even though they were recorded. Pass the default value of 'null' in order to use all active ReplayObject s in the current scene.
- **Bool (allowEmptyScene):** A value which determines whether an empty ReplayScene can be passed or not. An empty ReplayScene will have no valid ReplayObejcts to record but may still cause some metadata to be recorded which may be undesirable. Passing 'false' will cause the method to thrown an exception if the specified ReplayScene does not contain any ReplayObejcts. In some scenarios, it may be desirable to use and empty ReplayScene . For example: When dynamically spawned objects are used, an empty scene may be allowable in which case you can pass 'true' to disable empty scene exceptions.
- **ReplayPlaybackOptions (playbackOptions):** The options used by the playback service which determines much of the playback behaviour such as frame rate and end behaviour. The default value of 'null' causes the global project playback options to be used which are accessible via the settings window. You can also programmatically create the playback settings per playback operation if required.

StopPlayback

Use this method to stop a replay that was previously started by calling one of the BeginPlayback methods. This method will cause any replaying objects to be rest to live mode, triggering the associated ReplayPreparer to restore component states. You must also pass the ReplayHandle for the playback operation by reference as the handle will be set to an invalid state since the playback operation will no longer exist.

C# Code

```
1  public static void StopPlayback(ref ReplayHandle, bool)
```

- **Ref** ReplayHandle **(handle):** The handle for a running playback operation started using BeginPlayback. The handle must be passed by reference using the 'ref' keyword because it will be no longer be valid and will have its state reset to reflect this.
- **Bool (restorePreviousSceneState):** An optional value which determines whether the original scene state prior to commencing playback should be restored or not. The replay system will automatically record the scene state upon entering playback mode so that it can be reset if required. Passing 'false' will cause all replaying objects to be left in their current position. This could potentially leave objects in mid air or in overlapping collision states so it is recommended that most users use the default value of 'true' to reset to a known safe state.

IsReplaying

Use this method to determine whether a playback operation is currently running with an associated replay handle.

C# Code

```
1  public static bool IsReplaying(ReplayHandle)
```

- ReplayHandle **(handle):** A ReplayHandle to check for running playback operations. Disposed replay handles can be passed which will cause a value of false to be returned.

AddPlaybackEndListener

Use this method to add an event listener for when a playback operation reaches the end of a replay. This is useful to know when a replay has finished so that you can perform some other action.

C# Code

```
1  public static void AddPlaybackEndListener(ReplayHandle, Action)
```

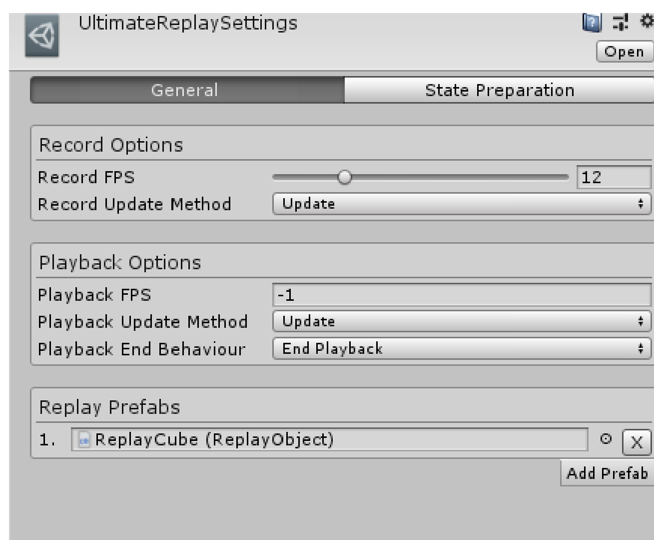
- ReplayHandle **(handle):** The handle of a valid playback operation that was started using BeginPlayback.
- **Action (playbackEndCallback):** The delegate which will be invoked when the associated playback operation reaches the end of a replay. The specified delegate must have a return type of 'void' and not have any parameters in order to be accepted.

Note: *A matching 'Remove Listener' method also exists to remove an added listener but has been omitted from this documentation. See the included scripting reference for more information.*

Replay Settings

Ultimate Replay 2.0 has a number of global settings which affect various aspects of the asset. Usually the default settings will be OK for most games but they can easily be changed by going to 'Tools -> Ultimate Replay -> Settings' which will open the global settings in the inspector window. The settings window is also where you will add prefab references to objects which may be destroyed or instantiated dynamically while recording. Take a look at the Replay Prefabs section for more information.

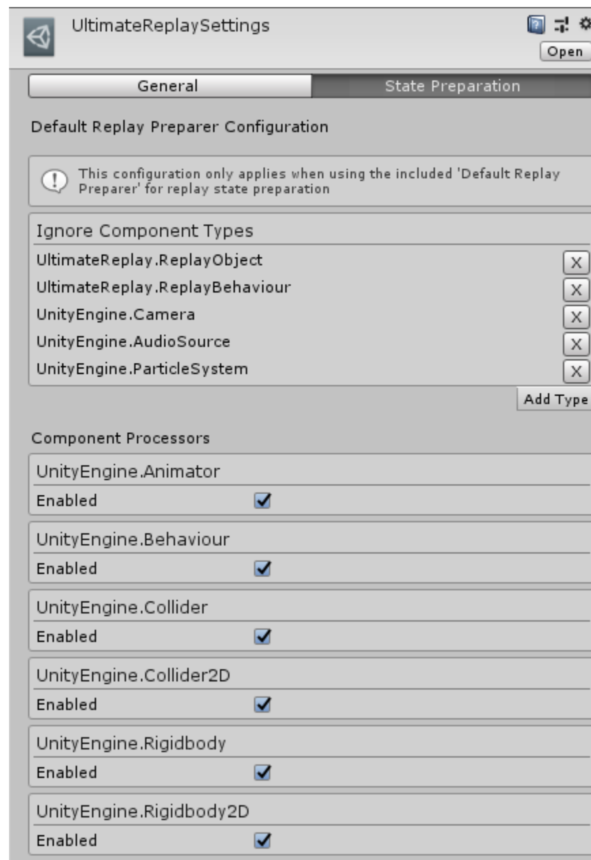
The following section will cover the main settings which can be found on the default 'General' tab:



- **Record FPS:** The record frame rate used to determine how many snapshot frames are captured per second. The default value is '12' which will usually be more than enough for most games when interpolation is used.
- **Record Update Method:** The Unity update event used to update the entire replay system during the recording phase. This option is mostly for compatibility reasons and allows replay captures to occur at different points in the game loop. The default option is Update which will be fine for most games and all supported options are:
  - **Update:** Use the main Unity update method to update the recording phase of the replay system.
  - **Late Update:** Use the Unity late update method to update the recording phase of the replay system.
  - **Fixed Update:** Use the Unity physics update method to update the recording phase. This is not recommended unless you are having issues with the first 2 update methods.
- **Playback End Behaviour:** Determines what happens when a replay reaches the end of its recording. Options are:
  - **End Playback:** The playback operation will automatically finish and cause the associated ReplayObject s to be prepared for gameplay by switching to live mode.
  - **Stop Playback:** The replay will stop on the very last frame of the recording but will remain in playback mode. This means that playback operations such as seeking will still work as expected and you will need to manually call StopPlayback to exit replay mode.

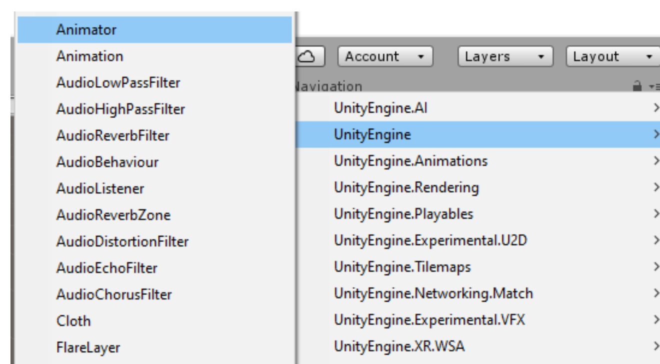
    - **Loop Playback:** The replay will loop around to the start when the end frame is reached resulting in an infinite looping replay. You will need to call StopPlayback manually to exit replay mode. This mode is useful for after gameplay highlights that run until player input is received.
- **Playback FPS:** The replay frame rate used to determine how often playback operations are updated. Higher playback frame rates will result in smooth replays, even if the record fps is low. This is because playback runs interpolation on the recorded snapshots meaning that it is possible to create virtual snapshots in between recorded snapshots with the effect of interpolated smoothing, much like in keyframe animation systems. The default value is '-1' which means that playback will run at the game FPS which is recommended. Lower playback FPS values can be used if you experience playback performance issues but playback frame rates lower than the recorded FPS are not recommended.
- **Playback Update Method:** The Unity update event used to update the entire replay system during the replay phase. This option is mostly for compatibility reasons and allows replay updates to occur at different points in the game loop. The default option is Update which will be fine for most games and all supported options are:
    - **Update:** Use the main Unity update method to update the playback phase of the replay system.
    - **Late Update:** Use the Unity late update method to update the playback phase of the replay system.
    - **Fixed Update:** Use the Unity physics update method to update the playback phase. This is not recommended unless you are having issues with the first 2 update methods.
- **Replay Prefabs:** A collection of prefab references to ReplayObject s that may be instantiated or destroyed during recording. In order for Ultimate Replay 2.0 to support dynamic object creation and destruction during recording, you will need to inform the replay system about any prefabs that may be dynamic by adding them to the prefabs collection. Instantiating or destroying a prefab instance during recording that is not added to this collection will result in playback accuracy issues along with a warning message. Take a look at the Replay Prefabs section for more information.

The next section will cover additional settings which can be found on the 'State Preparation' tab and relate to the built in Default Replay Preparer. As of version 2.1.0, the default replay preparer is now fully configurable to make it easier for users to change its behaviour.



- **Ignore Component Types:** A collection of component types which will be ignored by the default replay preparer. This means that they will not be processed or affected in any way when switching between playback and live modes.

  This collection will already contain a few types which are added by default and we recommend that they remain in most cases. Adding new types can be done easily by clicking the 'Add Type' button and selecting your desired component type from the resulting context menu. Types are organised by '[namespace] -> [type name]' to make them easier to find:



- **Component Processors:** This section contains settings for each component processor that is used by the default replay preparer. A component processor is simply a special type which prepares a specific component for either playback or live modes. Typically, these processors will either deactivate or disable their target component when entering playback mode and restore the target component to their original state when exiting playback mode.

Each component processor has the following options:

- **Enabled:** Is the component processor enabled. A disabled component processor will not run and as a result, will have no effect on the target component when the scene is being prepared.

Replay Scene

A replay scene is used to represent a collection of ReplayObject s which should be recorded or replayed. When you start a record or replay operation using the ReplayManager, you will usually need to pass a ReplayScene instance which will describe which objects should be included in the operation. You can create any number or ReplayScene s at a given time although you should take care not to start multiple operations on a single scene at the same time as this is not supported. For example, multiple record and replay operations can occur at the same time but not with the same scene instance. The ReplayManager will throw an exception is a scene instance is already in use.

Replay Scene Mode

A replay scene is a state object and can either be in live or playback modes. Changing the scene mode will cause all registered ReplayObject s to be prepared using the active replay preparer so that they are ready to receive record or replay updates.

- **Live Mode:** All ReplayObject s are reset to their initial state using the active replay preparer. This means that all scripts, physics components etc. are restored to their initial state. Usually meaning that they are re-enabled or re-activated so that they can interact with the game as usual.
- **Playback Mode:** All ReplayObject s are prepared for replay updates by the active replay preparer. Scripts and physics components will be disabled to prevent them from manipulating the objects during playback allowing the replay system to replicate the recording accurately.

Replay Preparers

A replay preparer is a special script which is executed on every ReplayObject in the ReplayScene when the scene mode is changed. The purpose of the replay preparer is to find and modify any components on the specified ReplayObject which could potentially interfere with playback accuracy. Components such as Rigidbodies or scripts are likely candidates as they could potentially move an object during a replay causing it to become out of place according to the recorded data. The state of such components is then saved or restored by the preparer depending upon the state change so that the component can be deactivated during playback.

By default, a built-in replay preparer called the DefaultReplayPreparer will be used to prepare ReplayObject s but it is possible to create your own replay preparer script if required. The default preparer will potentially affect the following component types:

- **MonoBehaviour scripts (Unless they derive from ReplayBehaviour):** Scripts will be disabled so that the 'Update' methods do not run.
- **Physics rigid body 2D / 3D:** Rigid bodies will be set to kinematic mode so that forces such as gravity do not affect the object.
- **Physics colliders 2D / 3D:** Colliders will be disabled so that collision resolution cannot affect playback.
- **Animator:** Animators will be disabled so that animation poses cannot be applied during playback.

Note: *Script components deriving from ReplayBehaviour will not be modified by the default replay preparer. You can derive from this class if you need to prevent a script from being disabled during playback on a particular replay object.*

Note: *Replay preparers work on a per object basis and will only affect ReplayObjects which were added to a ReplayScene. Entering or exiting playback mode will trigger the preparer to run on all ReplayObjects which are added to the active ReplayScene instance.*

Custom Replay Preparer

If you find that the default replay preparer is affecting components that you do not want it to, you could implement your own replay preparer so that you have full control over which components are affected.

Creating a custom replay preparer is as simple as implementing an interface and then registering it with the replay system. First you will need to implement the 'UltimateReplay.Core.IReplayPreparer' interface which has 2 methods:

- **PrepareForPlayback:** This method will be invoked when potential interfering components should be deactivated because the replay system is entering playback mode. The method will be invoked a number of times for each ReplayObject in the scene. You will need to use the Unity API to find such components and handle them accordingly.
- **PrepareForGameplay:** This method will be invoked when exiting playback mode as a result of calling StopPlayback and will run on all ReplayObject s in the associated ReplayScene . This method should be used to restore any modified components to their initial state.

If you wish to implement a custom ReplayPreparer then it may be worth taking a look at how the default preparer is implemented by examining the source code. You can find the default preparer implementation in the following source file (Not available in the trial version): 'Assets/Ultimate Replay 2.0/Scripts/Core/DefaultReplayPreparer.cs'. It may also be worth checking out the dedicated component preparers which can be found inside the 'StatePreparation' folder.

C# Code

```
1  class ExamplePreparer : IReplayPreparer
2  {
3      public void PrepareForPlayback(ReplayObject replayObject)
4      {
5          foreach(Collider collider in
6  replayObject.GetComponents<ReplayObject>())
7              collider.enabled = false;
8      }
9
10     public void PrepareForGameplay(ReplayObject replayObject)
11     {
12         foreach(Collider collider in
13 replayObject.GetComponents<ReplayObject>())
14             collider.enabled = true;
15     }
   }
```

Once you have implemented a custom replay preparer, the next step is to register it so that the replay system can make use of it. A replay preparer instance is associated with every ReplayScene instance as the preparer needs to be run on every ReplayObject in the ReplayScene . This means

that you can provide an IReplayPreparer implementation in the constructor of the ReplayScene type. If no preparer is passed, then the default preparer is used automatically.

C# Code

```
class Example : MonoBehaviour
{
    void Start()
    {
        ReplayScene scene = new ReplayScene(new ExamplePreparer());
        scene.AddReplayObject(...);

        ReplayManager.BeginPlayback(null, scene);
    }
}
```

Note: *It is possible to implement different replay preparers for different playback operations if required. For example: you may want some replays to have colliders enabled so that the player can interact with them while you may want another replay to be unaffected. This is possible by creating multiple ReplayScene instances.*

Replay Prefabs

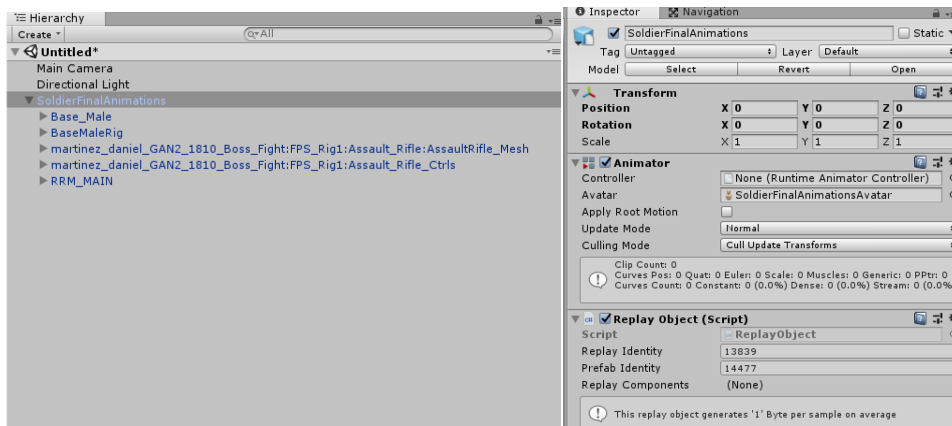
Ultimate Replay 2.0 is able to record and replay game objects which are instantiated or destroyed dynamically using the Unity API. There are some things to consider in order for this behaviour to be supported though:

- The target prefab should have a ReplayObject component attached to the root object. This component is used to identify the prefab and any amount of recorder components can also be attached in order to record and replay specific elements of the object like transform.
- Prefab objects need to be registered with Ultimate Replay 2.0 via the settings window otherwise dynamic instantiation/creation will not work. This allows the replay system to know which prefab the game object instance was created from during the recording phase so that the replay system can then destroy or create an identical instance during playback.
- Parenting is only supported when the prefab is instantiated and attached as a child to a game object which also has a ReplayObject component attached. For example: If you wanted to instantiate a weapon during recording and attach it to the players hand bone in the hierarchy, then the hand bone should have a ReplayObject component. The replay system will then re-create this hierarchy structure during recording. This is one case where it is OK to have multiple ReplayObject components on the same game object, one at the player root, and one at the hand bone.
- After instantiating a prefab, you will need to add a reference to any applicable ReplayScene s in order for the replay system to be notified of the object creation.

Registering Replay Prefabs

As previously mentioned, all prefab objects that need to be instantiated or destroyed during the recording phase need to be registered with Ultimate Replay 2.0 so that identical instances can be created on demand during playback. This is an easy process and only takes a couple of steps.

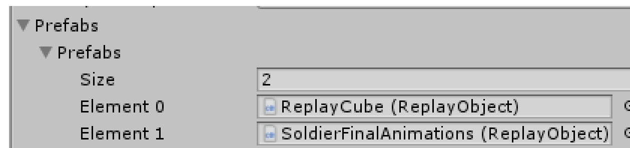
The first thing you will need to do is identify any prefabs which will need to be instantiated or destroyed while recording. Usually this may include prefabs like bullets or projectiles, effects that are spawned, and maybe even enemy characters. Once you have identified these prefabs then you should ensure that the prefab root has a ReplayObject component attached to it as the replay system will use this for identification purposes. Any number or replay recorder components such as ReplayTransform can also be added at the same hierarchy level or lower.



You can add a ReplayObject component to the selected object by going to 'Tools -> Ultimate Replay -> Make Selection Replayable -> ReplayObject '.

Once you have your prefabs setup correctly, the next step is to register them with Ultimate Replay 2.0 which is done using the settings window. Open the settings window by going to 'Tools -> Ultimate Replay -> Settings' where you will see a foldout named 'Prefabs'. Expand this section until you see an array property:

You will need to add any dynamic prefabs to this array by resizing the array and then using drag and drop to assign your prefabs.



Repeat this step for any other dynamic prefabs that you may need to register with the replay system. You can always add or remove the prefab references at a later date as you develop and refactor your game.

Register via script

Using the settings window is the easiest and recommended way of registering your prefabs with the replay system but it is also possible to do it from a script. It is also quite a simple process and requires only a single method call but there are some things to note:

- You will need to ensure that you register the dynamic prefab before you begin recording otherwise there may be issues with instantiating or destroying those prefab instances.
- The method accepts a game object argument which should be the prefab object and not a prefab instance. Passing a prefab instance may cause issues during playback.
- The prefab object passed to the method must have a ReplayObject component attached at the prefab root.

To register a prefab dynamically from a script you will need to use the 'ReplayManager .RegisterReplayPrefab' method as shown here:

C# Code

```
1  class Example : MonoBehaviour
2  {
3      // Assign in inspector
4      public GameObject myPrefab;
5
6      void Start()
7      {
8          // Register the prefab
9          ReplayManager.RegisterReplayPrefab(myPrefab);
10
11         // Record operations can now start
12         ReplayManager.BeginRecording(...);
13     }
}
```

Note: *Registering a dynamic prefab via script is not persistent like the settings approach so it will need to be performed on every game session. There is also no way to unregister a replay prefab once it has been added.*

Instantiating Replay Prefabs

Once you have registered your dynamic prefabs with Ultimate Replay 2.0, you are now able to call 'Instantiate' during the recording phase in order to create an instance of that prefab. Once you have instantiated a prefab, you will then need to add it to a replay scene manually so that it will be recorded like all other ReplayObject s. There are a few ways to achieve this:

Using Replay Scene

As covered previously, a ReplayScene will usually be passed to the replay begin methods such as BeginRecording. This means that the creation and management of this scene instance is handled by the user and as a result, adding a replay prefab to that scene is trivial:

C# Code

```
class Example : MonoBehaviour
{
    private ReplayScene recordScene;

    // Assign in inspector
    public GameObject myPrefab;

    void Start()
    {
        // Create a scene instance
        recordScene = ReplayScene.FromCurrentScene();

        // Start recording
        ReplayManager.BeginRecording(null, recordScene);

        // Create instance as usual in Unity
        GameObject go = Instantiate(myPrefab);

        // Add replay prefab instance to scene so that it is
recorded
        recordScene.AddReplayObject(go.
GetComponent<ReplayObject>());
    }
```

Using Replay Manager

An alternative and possibly easier approach is to use the ReplayManager to add the newly created replay object. The main benefit of this approach is that it is possible to add the replay object to all active recording scenes if multiple scenes are currently recording.

C# Code

```
class Example : MonoBehaviour
{
    private ReplayScene recordScene;

    // Assign in inspector
    public GameObject myPrefab;

    void Start()
    {
        // Create a scene instance
        recordScene = ReplayScene.FromCurrentScene();

        // Start recording
        ReplayManager.BeginRecording(null, recordScene);

        // Create instance as usual in Unity
        GameObject go = Instantiate(myPrefab);

        // Add to recording scenes
        ReplayManager.AddReplayObjectToRecordScenes(go.
GetComponent<ReplayObject>());
    }
}
```

As you can see by the name of the method, this will cause the replay object to be added to all active ReplayScene s which are in use by a running record operation.

Destroying Replay Prefabs

Destroying prefab instances during recording is an even easier process. There is no need to unregister the prefab instance when you are destroying it since it is detected automatically when the reference becomes null. This means that it is just a case of destroying your game object like you would normally using the Unity API:

C# Code

```
class Example : MonoBehaviour
{
    private ReplayScene recordScene;

    // Assign in inspector
    public GameObject myPrefab;

    IEnumerator Start()
    {
        // Create a scene instance
        recordScene = ReplayScene.FromCurrentScene();

        // Start recording
        ReplayManager.BeginRecording(null, recordScene);

        // Create instance as usual in Unity
        GameObject go = Instantiate(myPrefab);

        // Add to recording scenes
        ReplayManager.AddReplayObjectToRecordScenes(go.
GetComponent<ReplayObject>());

        yield return new WaitForSeconds(2f);

        // Destroy the prefab instance
        Destroy(go);
    }
}
```

If you are using a pooling system to instantiate and destroy your prefab instances, then you will need to manually unregister the replay object from the replay system manually since the object reference will not become null. This is due to the fact that most pooling systems will simply deactivate game objects but keep them in memory so the reference remains valid.

Unregistering a replay object from a ReplayScene manually is quite simple to do and unlike the instantiate approach, there is only one way that this can be done. You will need a reference to the ReplayScene that you are using for recording and the you can unregister the object like so:

C# Code

```
class Example : MonoBehaviour
{
    private ReplayScene recordScene;

    // Assign in inspector
    public GameObject instance;

    IEnumerator Start()
    {
        // Create a scene instance
        recordScene = ReplayScene.FromCurrentScene();

        // Start recording
        ReplayManager.BeginRecording(null, recordScene);

        // Use your favourite pooling asset to despawn the object
        SomePoolingAPI.Despawn(instance);

        // Remove the object from the scene and it will no longer be
recorded
        recordScene.RemoveReplayObject(go.
GetComponent<ReplayObject>());
    }
}
```

After following these steps, you should now be able to create and destroy prefab instances during recording and have them replayed without issue. Objects that were instantiated during recording will be created automatically during playback on demand and objects that were destroyed will also disappear during playback as you would expect.

Take a look at the included cubes demo scene which demonstrates this technique in a working example.

Replay Storage Target

In order to record objects in the scene, a storage device is required which has the responsibility of storing the recorded data in some form. These storage devices are known as ReplayStorageTarget s and there are many different types included with Ultimate Replay 2.0 by default. In addition, it is also possible to create your own storage target if the built-in targets do not meet your needs by implementing an abstract class. This is only recommended for advanced users though.

Any time you need to start recording or replaying, you will need to provide a ReplayStorageTarget as the source to read data from, or the target to write data to.

Replay Memory Target

A replay memory target is used to store replay data in memory as the name suggests. The data may be compressed to reduce storage space but you should still be wary about memory usage when recording large or complex scenes with a high number of recorder components. Take a look at the replay statistics section which shows how you can identify the memory used by stored data.

The replay memory target supports limited, unlimited and rolling recording modes to support a variety of use cases however you should take care with memory usage in large or complex scenes with many replay components. A limited or rolling buffer memory target is recommended in most cases.

C# Code

```
ReplayStorageTarget target;

// Create an unlimited memory target
target = ReplayMemoryTarget.CreateUnlimited();

// Create a rolling memory target to record the last 15s of gameplay
target = ReplayMemoryTarget.CreateTimeLimitedRolling(15);

// Create a memory target limited to a size of 65565 bytes in size
target = ReplayMemoryTarget.CreateMemorySizeLimited(65565);
```

- **Limited:** A limited memory target can be used to record a replay up to s specified memory size without exceeding that value. The constructor accepts an integer value which represents the maximum number of bytes that the memory target can store. If a write operation occurs when the memory target is full, an OutOfMemory exception will be raised.

  > **Note:** *The memory size specified does not include any overhead that may be allocated by the memory target in order to support storage, compression and read operations.*

- **Unlimited:** A memory target that is not limited in any way to the amount of data that can be stored. This means that you can theoretically record data until the system runs out of memory. We highly recommend that you only use an unlimited memory target if you target if you will be recording for short periods or will carefully monitor the memory usage on many test systems to ensure that it remains acceptable. You can use the replay statistics to get usage information and also the Unity profiler window.
- **Rolling Buffer:** A rolling buffer memory target is used to record the last X amount of gameplay continuously. The time value can be specified via the constructor in seconds and

the memory target will record data until the target time is met. From then on, every new second of data recorded will cause the very first second of the recording to be deleted resulting in the target containing only the last X amount of seconds of gameplay. This is a common technique used in games that make use of killcam or similar where you are only interested in recording the players last few seconds before they are killed.

Replay Stream Target

A ReplayStreamTarget is a storage target that is able to write to and read from a System.Stream object. The replay data is stored in a proprietary binary format that is highly optimized for reduced storage space and loading time. Any System.Stream implementation is supported as long as the following features are implemented:

- **Stream.Seek:** Required when reading the replay data as data is stored in chunks which are discoverable via lookup tables.
- **Stream.Position:** Required during reading and writing to calculate offsets.
- **Stream.Length:** Required during reading and writing to calculate negative offsets and data sizes.

The ReplayStreamTarget uses threading to stream chunks of replay data without blocking the main thread. This allows for seamless playback as chunk prediction algorithms are also used to pre-fetch data that may be required while in playback mode. Note that some platforms such as WebGL do not support threading in which case these async operations will be loaded back onto the main thread automatically which may cause stutters or jitter in replays. A ReplayMemoryTarget is recommended on these platforms if possible as there is much less overhead in terms of compression/decompression.

A ReplayStreamTarget storage device can be created for recording or replaying purposes using the following method:

C# Code

```
1 // Create a stream to hold the data
2 Stream stream = new MemoryStream();
3
4 // Create a storage target from the specified stream
5 ReplayStorageTarget target =
6 ReplayStreamTarget.CreateReplayStream(stream);
```

Note: *This approach works for both saving and loading a replay using a stream object. If you want to load a replay, then ensure that you pass a Stream object containing valid replay data and everything will work as expected.*

Replay File Target

A ReplayFileTarget can be used to stream a recording or a replay to or from file. This is highly useful if you need to create recordings that persist over multiple game sessions or even upload them to a sharing service or similar for other players to view. The ReplayFileTarget is built on top of the ReplayStreamTarget component and as a result, will generate the same data output using the same streaming techniques.

A ReplayFileTarget storage device can be created as shown below, depending upon whether you want to create a file, or read from an existing replay file. The resulting storage target can then be passed to BeginRecording or BeginPlayback in order to record or replay using the target.

C# Code

```
ReplayStorageTarget target = null;

// Create a new file with the specified path for recording
target =
ReplayFileTarget.CreateReplayFile("ReplayFiles/example.replay");

// Create a new file with a unique generated name for recording
target = ReplayFileTarget.CreateUniqueReplayFile("ReplayFiles/",
".replay");

// Load an existing replay file
target =
ReplayFileTarget.ReadReplayFile("ReplayFiles/example.replay");
```

Note: *If you have already created a replay file and have a reference to that storage target, you can simply pass the reference to BeginPlayback and the target will automatically switch into read mode so that the replay can be streamed.*

Custom Replay Storage Target

If the built-in storage targets do not meet your needs for any reason, then it is possible to create your own storage target by implementing and abstract class. You will need to implement the 'UltimateReplay.Storage.ReplayStorageTarget' abstract class and is only recommended for advanced users. The details of the implementation will not be covered here but the API is covered in the included scripting reference. You can also take a look at the source code for the built-in targets such as ReplayMemoryTarget to see how the storage device is implemented.

Replay Object

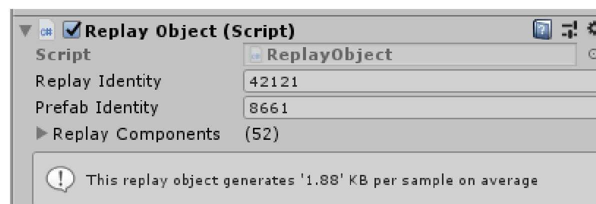
A ReplayObject is an essential replay component which must be attached to any game object that will record or replay data. The ReplayObject component acts as a managing component and is responsible for uniquely identifying the game object and managing any recorder components that may be attached, including child objects. Generally, you will only have a single ReplayObject component attached to the root of a game object hierarchy unless you are dealing with prefab instances. Recorder components such as ReplayTransform s can then be attached to the same game object or any child objects and will be automatically detected and managed by the Immediate parent or adjacent ReplayObject .

The main purpose of the replay object is to uniquely identify a game object in the replay system so that the necessary replay data can be distributed to the correct game objects during playback. A ReplayObject is identified by the Replay Identity property which is a persistent id value for the object. Any recorded data is stored along with the ReplayIdentity.

> **Note:** *ReplayObject components will be added automatically where applicable when adding a component deriving from ReplayBehaviour. A ReplayObject component will be added when no suitable adjacent or parent ReplayObject is found.*

Inspector

The ReplayObject component has a custom inspector drawer which displays useful information about the component.



- **Replay Identity:** A unique and persistent id value used by the replay system to identify the game object. This value will be generated automatically when adding a ReplayObject component or on game start for prefab instances.
- **Prefab Identity:** A unique id value that is used to identity the associated prefab if any. Note that this value will still be generated even if the parent game object is not a prefab instance. See replay prefabs for more information.
- **Replay Components:** A list of replay components that are managed by this ReplayObject . Components will be displayed by their object name and component type and will be indented according to their hierarchy depth for clarity. Note that this list is read-only and is updated automatically when replay components are added or removed.

Identity Transfer

A ReplayObject and replay recorder components all have unique id values which are generated by the replay system to identify each component to the replay system. These id values are then used during playback to route the necessary replay data to the correct component for deserialization. This works well and is an efficient means of storage since an object can be identified by a 2-byte value (2 bytes by default but 4 bytes is supported).

In some scenarios, it may be desirable to modify the identity of a game object for replay purposes so that it is able to replay the recorded data from another object. An example where this may be useful would be a ghost vehicle in a racing game where you would want to record the player vehicle but replay onto another ghost vehicle object. This can be achieved by transferring the identity of a source object onto a target object using identity transfer. Ultimate Replay 2.0 has a quick and easy way to do this but there are some things to consider:

- The source and target objects should have the same hierarchy structure where replay components are attached.
- The source and target replay objects should have the same observed component count and order.

A good way to ensure that these replay components are structured the same is to duplicate the source object (The player vehicle in this example) and create the new target object by modifying the duplicate (The ghost vehicle).

Once you have 2 objects that share the same replay hierarchy then you can transfer replay identities at any time by calling the following method:

C# Code

```
bool ReplayObject.CloneReplayObjectIdentity(ReplayObject,
ReplayObject)
```

This method takes a source object as the first parameter and the target object as the second parameter. Using the ghost vehicle example, we would pass the player vehicle object first and the ghost vehicle object second. The method will then copy and apply the replay identities of all attached replay components onto the target object.

There is also an overload method which accepts 2 game objects for ease of use. This method will simply get all ReplayObject components from the source and target objects and clone each one.

C# Code

```
bool ReplayObject.CloneReplayObjectIdentity(GameObject, GameObject)
```

Replay Behaviour

A ReplayBehaviour is a component that derives from MonoBehaviour and has many useful contextual replay properties and events. It may be useful to create script components that derive from ReplayBehaviour when they will be attached to a recorded object. In addition, ReplayBehaviour components are not affected by Replay Preparers so it may be useful to derive from this base class if your script need to run while in playback mode.

The ReplayBehaviour type has a number of useful properties such as IsRecording and IsReplaying which will allow you to determine the current state of the associated game object. These properties are contextual and not global like in the original asset. That means that some game objects may be in recording mode while others may be in playback mode at the same time since multiple simultaneous replay operations are now supported. These contextual properties allow you to determine the true replay state of a specific object.

Replay Messages

The ReplayBehaviour component has a number of virtual methods that can be overridden and will be called by the replay system at various times. These message events may be useful to your game components when the state of the replay system changes. Note that these messages are contextual and will only be called on the relevant ReplayBehaviour s since different recording and replay states are possible for different objects.

OnReplayStart

Called when the parent game object is about to start replaying. This is triggered as a result of a call to BeginPlayback.

C# Code

```
1  public virtual void OnReplayStart();
```

OnReplayEnd

Called when the parent game object is about to end playback. This could be triggered as a result of an EndPlayback call or if the replay reaches the end of the recording.

C# Code

```
1  public virtual void OnReplayStart();
```

OnReplayPlayPause

Called when the parent game object is about to pause or resume playback. This is triggered as a result of calling PausePlayback or ResumePlayback. The bool value passed to the event indicates whether the pause state is enabled or disables where a value of ‘true’ means paused.

C# Code

```
1  public virtual void OnReplayPlayPause(bool);
```

OnReplayReset

Called when the behaviour should reset or discard any cached values or data as playback may be about to begin. This is useful for resetting any interpolation data or similar that may be stored between replay frames.

C# Code
```
1  public virtual void OnReplayReset();
```

OnReplayCapture

Called when the behaviour should submit any replay data for recording. This event will only be called during recording and is useful to submit event or method record data by calling RecordEvent or RecordMethodCall. Note that replay data can also be submitted via update methods or similar but you must take care to only record data during recording phases.

C# Code
```
1  public virtual void OnReplayCapture();
```

OnReplayUpdate

Called when the behaviour should update during playback. This method will only be called during playback and will be passed the current ReplayTime information for the associated replay. This event is useful for updating any replay elements such as interpolation at full game speed. The passed time value includes information about the current playback time and delta time between frames which may be useful for interpolation.

C# Code
```
1  public virtual void OnReplayUpdate(ReplayTime);
```

OnReplayEvent

Called when the behaviour should read replay event data. When you record an event using RecordEvent, this method will be invoked during playback with the event id and state data. The id value is used to identify the event type and the event data is an option data state that was passed to the RecordEvent method. You will need to read the state data in the correct format and order to avoid errors.

C# Code
```
1  public virtual void OnReplayEvent(ushort, ReplayState);
```

OnReplaySpawned

Called when the parent game object was instantiated for playback purposes by the replay system. A replay registered prefab instance must be instantiated during recording in order for this method to be triggered during playback. The initial position and rotation of the object as passed.

C# Code
```
1  public virtual void OnReplaySpawned(Vector3, Quaternion);
```

Replay Events

Replay events can be used when you want to record a change in state or similar that does not happen often. You can record an event during the recording phase and the replay system will invoke the OnReplayEvent callback of the ReplayBehaviour component when that event was reached during playback. A ReplayEvent is made up of an event id value which is specified by the user, along with an optional ReplayState containing any data associated with the event. It is the responsibility of the user to specify unique event id values and to ensure that any event data is serialized and deserialized in the correct format.

In order to record a ReplayEvent, you will first need to create a script deriving from ReplayBehaviour . You can then call the RecordEvent method while the object is recording in order to record a replay event. You can make use of the IsRecording property of the ReplayBehaviour component to check for the recording phase. Alternatively, you can also override the OnReplayCapture event which will only be called during the recording phase.

C# Code

```
class Example : ReplayBehaviour
{
    float lastTime = 0;

    void Update()
    {
        // Record a replay event every second
        if(IsRecording == true && Time.time > lastTime + 1f)
        {
            // Add optional data to the event
            ReplayState state = ReplayState.pool.GetReusable();
            state.Write("Hello World");

            // Record an event
            RecordEvent(1, state);

            lastTime = Time.time;
        }
    }
}
```

As you can see in the above code example, a replay event is recorded every second that the object is being recorded. Note that we need to pass an event id value which in this case is set to a value of '1' but the event id can be any value between 0-65565. We also pass an optional ReplayState to the RecordEvent method containing a simple string, but we could also add much more useful information here or event pass 'null' if no data is required.

Receiving replay events during playback is just as simple and involves overriding the OnReplayEvent callback of the ReplayBehaviour script:

```csharp
class Example : ReplayBehaviour
{
    public override void OnReplayEvent(ushort eventID, ReplayState eventData)
    {
        switch(eventID)
        {
            case 1:
            {
                Debug.Log("Event 1: " + eventData.ReadString());
                break;
            }
        }
    }
}
```

The OnReplayEvent callback will be invoked by the replay system for every event type. You will then need to filter the events by event id as shown in the code example and handle the event accordingly. In this case, we simply check for our event id of '1' and print out the string value that was recorded to the console.

Replay Methods

Replay methods can be used to record and replay method calls of any static or instance method of a ReplayBehaviour script. Only methods that accept primitive parameters such as int, string, bool etc. are supported and the replay system will log an error if an unsupported parameter type is used. In order to record a method, you can simply use the following ReplayBehaviour methods:

```csharp
public void RecordMethodCall(Action method)
public void RecordMethodCall<T>(Action<T> method, T arg)
public void RecordMethodCall<T0, T1>(Action<T0, T1> method, T0 arg0, T1 arg1)
public void RecordMethodCall<T0, T1, T2>(Action<T0, T1, T2> method, T0 arg0, T1 arg1, T2 arg2)
public void RecordMethodCall<T0, T1, T2, T3>(Action<T0, T1, T2, T3> method, T0 arg0, T1 arg1, T2 arg2, T3 arg3)
```

As you can see, there are a number of overload methods which allow any method with any parameter type to be supported, limited to a parameter count of 4. The following code demonstrates how to use these methods:

C# Code

```
class Example : ReplayBehaviour
{
    void Update()
    {
        if(IsRecording == true)
        {
            RecordMethodCall(DoSomething);
            RecordMethodCall(DoSomethingElse, "Hello World", 3);
        }
    }

    void DoSomething()
    {
        Debug.Log("Hello");
    }

    void DoSomethingElse(string message, int count)
    {
        for(int i = 0; i < count; i++)
            Debug.Log(message);
    }
}
```

As you can see in the example, the methods are passed as delegates as the first argument and then you may or may not need to specify the additional arguments for the target method. Parameters at index 1 and onwards will be the arguments for the target method in order. You can see in the example code that a string and integer argument need to be passed when recording the DoSomethingElse method otherwise a compiler error will be generated.

Method recording should only be done when the ReplayBehaviour is currently recording which can be determined by checking the IsRecoreding property. Alternatively, you can do all of the recording inside the OnReplayCapture event which will only be called during the recording phase.

Calling RecordMethodCall during recording will cause the method to be invoked immediately with the specified arguments, as if calling it directly. The method information will then also be serialized by the replay system along with the specified arguments so that the method can be called again during playback.

Note: *Methods that return a value cannot be recorded by the replay system. If you want to record a method that returns a value, then you should create a void wrapper method to call that target method, disregarding the return value. You can then record the wrapper method as normal.*

Replay Variables

Replay variables are a simple and convenient way of recording and replaying primitive class variables without the need to write the serialize and deserialize code. Any field defined in a class deriving from ReplayBehaviour can use replay variables as long as the type is a primitive such as int and string, or if the type is a Unity Vector3, Quaternion or Color. You can mark a variable as replayable by adding the ReplayVar attribute as shown below:

C# Code

```
class Example : ReplayBehaviour
{
    [ReplayVar]
    public int myValue = 50;
}
```

Once a field has been marked as a replay variable, you can change its value during recording and any changes will be restored during playback. This is all done automatically by the replay system which makes it quick and easy to implement.

C# Code

```
class Example : ReplayBehaviour
{
    [ReplayVar]
    public int myValue = 50;

    void Update()
    {
        if(IsRecording == true)
        {
            myValue++;
        }
        else if(IsReplaying == true)
        {
            Debug.Log(myValue);
        }
    }
}
```

Note: *ReplayVariables may have a little more overhead in the way of storage space when compared with custom recorder components. This is because extra metadata needs to be stored along with the data.*

Replay variables also fully support interpolation if the type support it. By default, values of type int, float or similar will be interpolated during playback to smooth out transitions which may or may not be desirable. If you are recording values like state id's or index values, then interpolation should preferably be disabled to prevent strange behaviour. This can be done by simply passing 'false' in the attribute declaration.

C# Code

```
class Example : ReplayBehaviour
{
    [ReplayVar(false)]
    public int myValue = 50;
}
```

Recorder Components

Recorder components are special components that can be attached to a game object and are intended to record and replay the behaviour of another component. For example, A ReplayTransform component is used to record and replay the Unity transform component. Ultimate Replay 2.0 has a number of built in recorder components which can be used to record a number of Unity components which may or may not be useful for your game.

Every recorder component must have a ReplayObject component attached to the same game object or to a game object higher in the hierarchy. If no suitable ReplayObject component is found when attaching a recorder component, Ultimate Replay 2.0 will automatically add the ReplayObject component to the same object.

Recorder components will also display information about the amount of data they generate per recording sample. This information will usually be displayed at the bottom of the inspector window for the component in a help box. Note that some components may only be able to display this data when in play mode.

Replay Transform

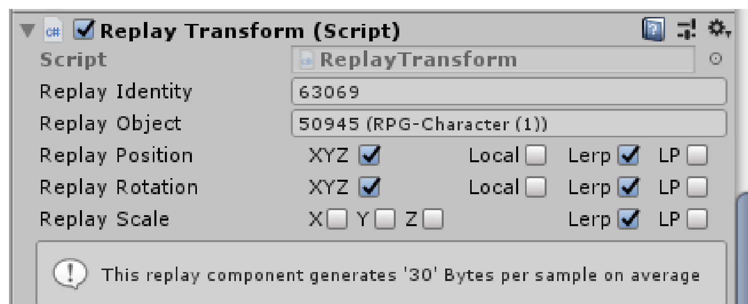
The ReplayTransform component can be added to a game object and will cause the transform of that object to be recordable and replayable by Ultimate Replay 2.0. Position, rotation and scale values can be recorded and replayed in various configurations and the component also supports frame interpolation for smoother playback.

Create Menu

A ReplayTransform component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Transform'. This will cause a ReplayTransform component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayTransform component has a number of properties which can be edited via the inspector window. Multi-object editing is supported for this component.



- **Replay Identity:** The unique id value given to the component by the replay system. This value is auto-generated.
- **Replay Object:** The unique id value of the associated replay object that is managing the component. Name information may also be included for quick lookup.
- **Replay Position:** Should the positional aspect of the transform be recorded.

- **XYZ:** When enabled, all elements of the position will be recorded. You can disable this toggle to select individual positional elements if required.
    - **Local:** When enabled, location position will be recorded as opposed to world rotation. Local recording is recommended when the game object is a child of another object.
    - **Lerp:** Should linear interpolation be used during playback for smooth frame transitions. Interpolation is highly recommended as it allows for much lower recording frame rates.
    - **LP:** Should the recorded data be stored in low precision. This is only recommended for objects that do not move much, are not in the main focus of a camera view and are not too fare from the world centre (+-1000 units max). This will cause the data to be stored in half precision so some accuracy may be lost.
- **Replay Rotation:** Should the rotational aspects of the transform be recorded.
    - **XYZ:** When enabled, all axis of rotation will be recorded and stored as a quaternion data structure to avoid gimbal lock. When disabled and all axis are not selected, the data will be stored as Euler angles for the selected axis. Disable this toggle to select individual axis elements for recording.
    - **Local:** When enabled, local rotation will be recorded as opposed to world rotation. Local recording is recommended when the game object is a child of another object.
    - **Lerp:** Should linear interpolation be used during playback to smooth rotation between playback frames. Interpolation is highly recommended and allows for much lower recording frame rates.
    - **LP:** Should the recorded rotation be stored in low precision. This is only recommended for objects that do not move much, are not in the main focus of a camera view and are not too fare from the world centre (+-1000 units max). This will cause the data to be stored in half precision so some accuracy may be lost.
- **Repay Scale:** Should the scale aspects of the transform be recorded.
    - **XYZ:** When enabled, all elements of the transform scale will be recorded. Disable this toggle to select individual record axis. By default, scale recording is disabled as it is not often required.
    - **Lerp:** Should linear interpolation be used during playback to smooth scale between playback frames. This option is disabled by default for the scale element as it may cause undesirable behaviour if you intend to snap between 2 scale values.
    - **LP:** Should the scale data be recorded in low precision.

Replay Enabled State

The ReplayEnabledState component is used to record and replay an objects active state as set using SetActive. This is useful to record objects that may become active or inactive during recording and will cause the same behaviour during playback.

Create menu

A ReplayEnabledState component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Enabled State. This will cause a ReplayEnabledState component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

This component does not have any editable inspector properties but does display information that may be useful.

- **Replay Identity:** The unique id value given to the component by the replay system. This value is auto-generated.
- **Replay Object:** The unique id value of the associated replay object that is managing the component. Name information may also be included for quick lookup.

Replay Component Enabled State

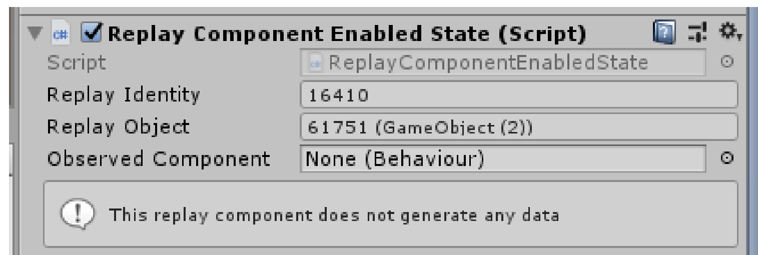
The ReplayEnabledComponentState component is much like the ReplayEnabledState component but is used to record the state of behaviour components. This is useful to record the state of components such as scripts deriving from ReplayBehaviour which will be replayed. Note that non-ReplayBehaviour scripts will usually be disabled when entering playback mode by the ReplayPreparer so you should only record non-script behaviours or scripts deriving from ReplayBehaviour which are treated specially by the ReplayPreparer.

Create menu

A ReplayComponentEnabledState component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Component Enabled State. This will cause a ReplayComponentEnabledState component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

This component does not have any editable inspector properties but does display information that may be useful.



- **Replay Identity:** The unique id value given to the component by the replay system. This value is auto-generated.
- **Replay Object:** The unique id value of the associated replay object that is managing the component. Name information may also be included for quick lookup.
- **Observed Component:** A reference to a behaviour component that should have its enabled state recorded. It is recommended that the assigned behaviour is attached to the same object hierarchy although it is not required. If no behaviour is assigned then no data will be recorded.

Replay Animator

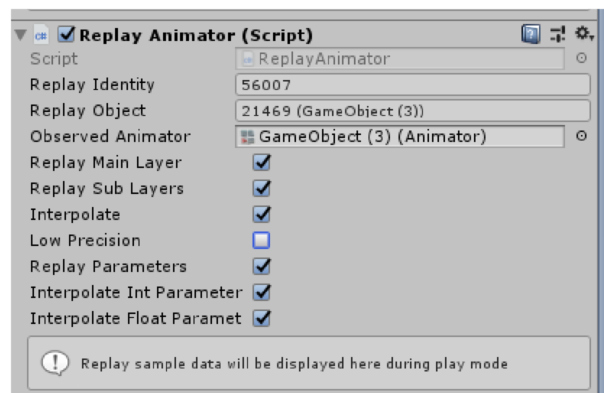
The ReplayAnimator component is used to record and replay the animation state driven by the Unity Animator component. This is useful for recording animated characters or objects which use the Animator component. The component will serialize all necessary state data for the animator including parameters so that the animation state can be recreated during playback.

Create menu

A ReplayAnimator component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Animator. This will cause a ReplayAnimator component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayAnimator component has al number of inspector properties which affect how and what data is recorded. The default options will usually be sufficient for most games however it may be desirable to play around with the options to get the most accurate playback results using as little storage space as possible. This is especially true for Animator components which do not define any parameters.



- **Replay Identity:** The unique id value given to the component by the replay system. This value is auto-generated.
- **Replay Object:** The unique id value of the associated replay object that is managing the component. Name information may also be included for quick lookup.
- **Observed Animator:** The animator component that should be recorded and replayed. When adding the component, the observed animator property may be automatically filled out with any animator component attached to the same game object. It is recommended that the assigned Animator component is attached to the same game object or hierarchy.
- **Replay Main Layer:** When enabled, the main layer of the Animator state machine will be recorded and replayed.
- **Replay Sub Layers:** When enabled, all additional layers of the Animator state machine will be recorded and replayed.
- **Interpolate:** Should animation poses be interpolated between replay frames to give a smoother playback result in low record FPS scenarios. Interpolation is highly recommended to produce smooth results with minimal stored data.
- **Low Precision:** When enabled, data will be stored in low precision where supported in order to reduce the storage space required. This is not recommended for animated objects that are close to the camera as there may be some data loss causing slight inaccuracies during playback.

- **Replay Parameters:** Should the Animator state machine parameters be recorded. It is highly recommended that parameters are recorded if they are used in order to create an accurate replay.
- **Interpolate Int Parameters:** Should integer parameters of the animator state machine be interpolated during playback. Interpolation is not recommended if the int parameter is used as an index/state value or similar and not as a numerical value as it could cause strange behaviour during playback.
- **Interpolate Float Parameters**: Should floating point parameters of the animator state machine be interpolated during playback. This may be useful for parameters such as move speed or similar which can be smoother gradually between replay frames.

Replay Particle System

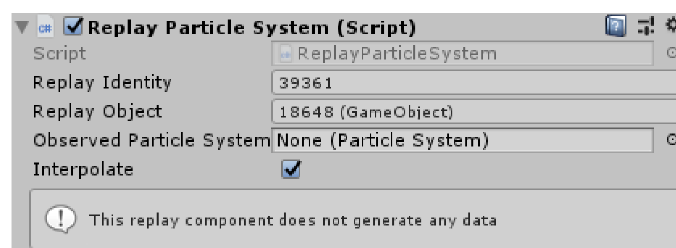
The ReplayParticleSystem component is used to record and replay the Unity particle system component. The component works by serializing the particle system simulation time during the recording phase and then re-simulating to that timestamp during the playback phase.

Create menu

A ReplayParticleSystem component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Particle System. This will cause a ReplayParticleSystem component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayParticleSystem has some required and optional properties in order for the component to work correctly which can be setup via the inspector window.



- **Replay Identity:** The unique id value of the recorder component which is auto-generated by the replay system when the component is added.
- **Replay Object:** The unique id value of the managing ReplayObject that is responsible for this recorder component. The property may also include component name information for easy lookup.
- **Observed Particle System:** A reference to the Unity particle system component that should be recorded and replayed. This is a required value and leaving it empty will cause the component to do nothing.
- **Interpolate:** Should the particle system by interpolated during playback to produce smoother results where low record frame rates are used.

Replay Audio

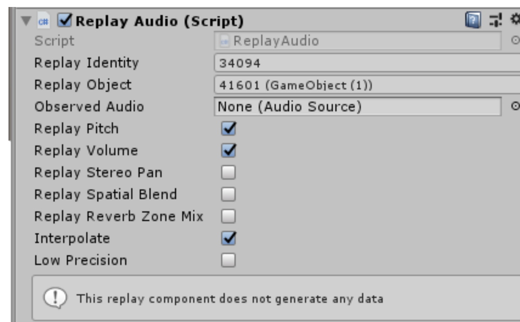
The ReplayAudio component can be used to record and replay a Unity audio source so that game sound effects can be used in replays. The audio component works by detecting audio played from a specific AudioSource component and then storing data such as sample rate and time values so that the audio can be replayed during playback.

Create menu

A ReplayAudio component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Audio. This will cause a ReplayAudio component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayAudio component has a number of inspector properties that determine which data is recorded and can be used to optimize for playback accuracy vs storage size.



- **Replay Identity:** A unique ID value used to identify the component which is auto-generated by the replay system when the component is added.
- **Replay Object:** The unique ID value of the associated managing ReplayObject component which is responsible for updating this recorder component.
- **Observed Audio:** A reference to a Unity AudioSource component which will be used to record and replay any emitted audio. If this value is not assigned, then the ReplayAudio component will do nothing. Note that the observed audio source should have a single audio clip assigned which should not be changed at any time.
- **Replay Pitch:** Should the pitch value of the audio source be recorded. You can disable this value if the pitch value of the audio source will never change during recording or playback.
- **Replay Volume:** Should the volume value of the audio source be recorded. You can disable this value if the volume will never change during recording or playback.
- **Replay Stereo Pan:** Should the stereo pan value of the audio source be recorded. Only required if the stereo pan value will change during recording or playback.
- **Replay Spatial Blend:** Should the spatial blend value of the audio source be recorded. Only required if the spatial blend value of the audio source will change during recording or playback.
- **Reverb Zone Mix:** Should the reverb zone mix value of the audio source be recorded. Only required if the reverb zone mix of the audio source will change during recording or playback.
- **Interpolate:** Should the audio source time sample value be interpolated to create smoother audio playback. Disabling this option may cause strange audio playback if low record rates are used.
- **Low Precision:** Should the component record supported data in low precision mode to consume less storage space.

Replay Material Change

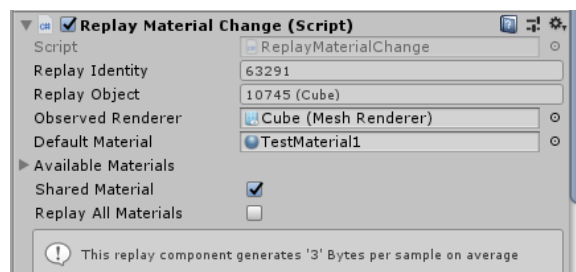
The ReplayMaterialChange component can be used to record and replay any material changes of a renderer component. Material changes will be detected automatically while recording and the component will attempt to restore the correct material during playback from a pool of possible materials that you can setup.

Create menu

A ReplayMaterialChange component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Material -> Material Change'. This will cause a ReplayMaterialChange component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayMaterialChange component has some inspector properties which can be used to control how the renderer material is recorded. There is also useful information displayed here such as the replay id used to identify the component in the replay system.



- **Replay Identity:** The unique ID value used to identify the recorder component in the replay system. This value is auto-generated when the component is added to a game object.
- **Replay Object:** The unique ID value of the managing ReplayObject component that is responsible for updating this recorder component.
- **Observed Renderer:** A reference to a Unity renderer component whose material should be recorded and replayed.
- **Default Material:** A fallback material instance that will be used when the assigned material could not be restored. Typically, this will occur when you assign a material instance that has not been added to the 'Available Materials' array. By default, this material will be set to the main material of the renderer at the time of adding the component.
- **Available Materials:** A collection of material instance that could potentially be assigned to the observed renderer. The ReplayMaterialComponent is only able to restore materials that have been added to this collection and assigning a different material will cause the component to fallback to the 'Default Material' during playback.
- **Shared Material:** Should the component record and replay using the 'sharedMaterial' property of the renderer. This is highly recommended because the 'material' property of the renderer will be used if this value is disabled. The 'material' property will allocate a new material instance on access.
- **Replay All Materials:** Enable this option if your target renderer has more than one material. This will ensure that material changes for all material slots are recorded and replayed.

Replay Material

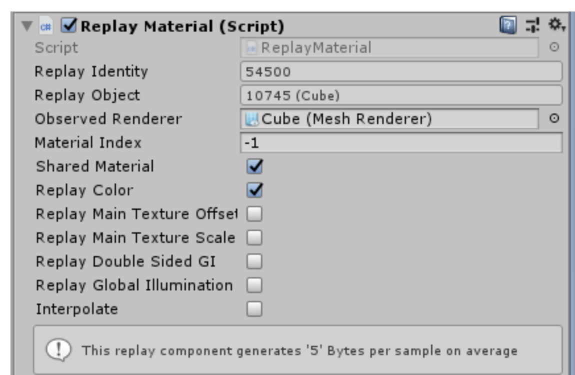
The ReplayMaterial component can be used to record and replay properties of a material assigned to a specified renderer component slot. This is useful for recording material properties such as color change over time.

Create menu

A ReplayMaterial component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Material -> Material Properties. This will cause a ReplayMaterial component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayMaterial component has some inspector properties which can be used to control how the renderer material is recorded. There is also useful information displayed here such as the replay id used to identify the component in the replay system.



- **Replay Identity:** The unique ID value used to identify the recorder component in the replay system. This value is auto-generated when the component is added to a game object.
- **Replay Object:** The unique ID value of the managing ReplayObject component that is responsible for updating this recorder component.
- **Observed Renderer:** A reference to a Unity renderer component whose material properties should be recorded and replayed.
- **Material Index:** The index of the material that you want to record. This index value represents the material index into the observed renderers material collection. Use a value of '-1' if the main material of the renderer should be used. Note that you can use multiple ReplayMaterial components on the same renderer to record properties for multiple materials.
- **Shared Material:** Should the component record and replay using the 'sharedMaterial' property of the renderer. This is highly recommended because the 'material' property of the renderer will be used if this value is disabled. The 'material' property will allocate a new material instance on access.
- **Replay Color:** Should the color property of the target material be recorded and replayed.
- **Replay Main Texture Offset:** Should the main texture offset of the target material be recorded and replayed.
- **Replay Main Texture Scale:** Should the main texture scale of the target material be recorded and replayed.
- **Replay Double Sided GI:** Should the double-sided global illumination property of the target material be recorded and replayed.

- **Replay Global Illumination:** Should the global illumination flags of the target material be recorded and replayed.
- **Interpolate:** Should supported material properties be interpolated between frames to provide a smooth transition. For supported properties such as color, this will create a smooth blend effect over time between the last and target color.

Replay Line Renderer

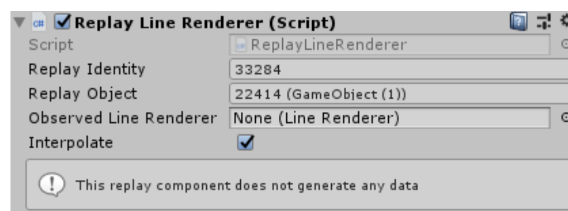
The ReplayLineRenderer component can be used to record and replay the Unity line renderer component. Note that this component can potentially use a lot of storage space so it is worth keeping an eye on the statistics information displayed in the inspector window.

Create menu

A ReplayLineRenderer component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Line Renderer. This will cause a ReplayLineRenderer component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayLineRenderer component has some inspector properties which can be used to control how the line renderer is recorded. There is also useful information displayed here such as the replay id used to identify the component in the replay system.



- **Replay Identity:** The unique ID value used to identify the recorder component in the replay system. This value is auto-generated when the component is added to a game object.
- **Replay Object:** The unique ID value of the managing ReplayObject component that is responsible for updating this recorder component.
- **Observed line Renderer:** A reference to a Unity line renderer component which will be recorded and replayed. It is recommended that the assigned line renderer exists on the same game object, although it is not a requirement.
- **Interpolate:** Should the line renderer positions be interpolated during playback. This will produce smoother results if low record rates are used and is highly recommended.

Replay Trail Renderer (Unity 2018.2 or newer)

The ReplayTrailRenderer component can be used to record and replay a Unity trail renderer component. Note that this component can potentially use a lot of storage space so it is worth keeping an eye on the statistics information displayed in the inspector window.

> **Note:** *This recorder component is only available in Unity version 2018.2 or newer, and is not available in the trial version of the due to the usage of pre-processor directives. The trail renderer component did not exist in previous versions of Unity.*

Create menu

A ReplayTrailRenderer component can be added via the menu 'Tools -> Ultimate Replay -> Make Selection Replayable -> Replay Trail Renderer. This will cause a ReplayTrailRenderer component to be added to the selected game object and may also attach a ReplayObject component if required.

Inspector

The ReplayTrailRenderer component has some inspector properties which can be used to control how the line renderer is recorded. There is also useful information displayed here such as the replay id used to identify the component in the replay system. The properties are much the same as the ReplayLineRenderer component as the same technique is used for both components.

- **Replay Identity:** The unique ID value used to identify the recorder component in the replay system. This value is auto-generated when the component is added to a game object.
- **Replay Object:** The unique ID value of the managing ReplayObject component that is responsible for updating this recorder component.
- **Observed line Renderer:** A reference to a Unity line renderer component which will be recorded and replayed. It is recommended that the assigned line renderer exists on the same game object, although it is not a requirement.
- **Interpolate:** Should the line renderer positions be interpolated during playback. This will produce smoother results if low record rates are used and is highly recommended.

Custom Recorder Components

Ultimate Replay 2.0 has a number of built in recorder components which can record and replay many frequently used Unity components. In some scenarios, it may be the case that a custom recorder component is required because one does not already exist, or you need to record a component from a third-party package. In Ultimate Replay 2.0, we have made it as easy as possible to create a custom recorder component and it can be as simple as implementing the ReplayRecordableBehaviour abstract class. This class has just 2 method which need to be overridden: OnReplaySerialize and OnReplayDeserialize.

In order to create a custom recorder component, a basic understanding of how the replay system works will prove very useful. Essentially, the replay system has a fixed record rate such as 16ms intervals for 60FPS. The replay system will wait for 16ms to pass and then begin a 'Sample Pass' which is where all replay objects collect and return data from their observed recorder components. The replay object will call OnReplaySerialize for all recorder components which is where per component data is serialized into a ReplayState object. The replay system will then verify, tag and compress the data ready for writing to the associated storage target and repeat until StopRecording is called.

Playback is much the same as recording except that the data is sent to the appropriate replay object which is then responsible for calling OnReplayDeseirlaize on the observed component. The OnReplayDeserialize can be used to restore the state of the component or to store the data for interpolation purposes. The basic principles are:

- OnReplaySerialize is used to record component data which is needed to restore the state at a later time. For example, the ReplayTransform component can record position, rotation and scale data so that the transform can be updated fully during playback. The method will be called multiple times during recording to create a sequence of state data.
- OnReplayDeserialize is called when the component should restore its state. The correct state date for the sequence position will be provided so it is just a case of deserializing the data and restoring the component state. It may also be desirable to store the state data between frames as fields so that interpolation can be performed via the update method.

Here is an example of a custom recorder component that records and replays the assigned material of a renderer component:

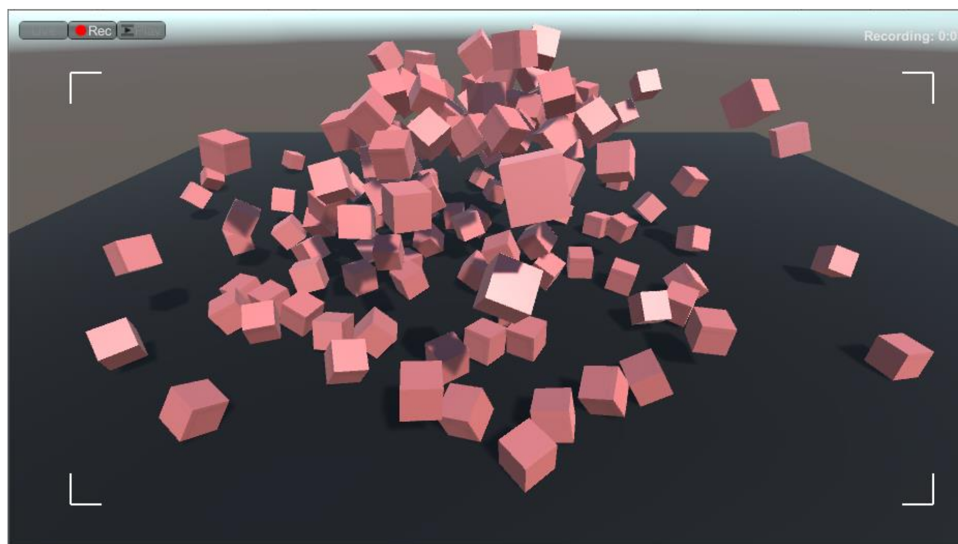
C# Code

```
class Example : ReplayRecordableBehaviour
{
    public Renderer renderer;
    public Material[]  materials;

    public override void OnReplaySerialize(ReplayState state)
    {
        bool matechedMaterial = false;

        for(int i = 0; i < materials.Length; i++)
        {
            if(renderer.material == materials[i])
            {
                state.Write(i);
                matchedMaterial = true;
                break;
            }
        }

        if(matchedMaterial == false)
            state.Write(-1);
    }

    public override void OnReplayDeserialize(ReplayState state)
    {
        int materialIndex = state.ReadInt32();

        if(materialIndex == -1)
        {
            renderer.material = null;
        }
        else if(materialIndex >= 0 && materialIndex <
materials.Length)
        {
            renderer.material = materials[materialIndex];
        }
    }
}
```

The material is checked against an array of possible materials and an index value is stored into the ReplayState object which represents the assigned material. If the current material does not exist in the possible materials array then an error value of '-1' is stored in the state object to reflect this. The deserialize method then attempts to restore the correct material to the renderer by reading back this index value. Note that the error case is properly handled and will set the renderer material to null.

Replay Controls

Ultimate Replay 2.0 includes a simple replay controls UI just like in the original asset. This UI is implemented using the legacy Unity immediate mode GUI and is intended for quick testing and demonstration purposes. If you need a similar in game UI then we recommend that you create your own using your favourite UI package or asset since the immediate mode GUI is now depreciated. Here you can see the replay controls UI in record mode:



The replay controls UI has 3 different modes which can be used to switch between different states in the replay system. Ultimate Replay 2.0 supports an unlimited number of simultaneous record and replay operations; however, the replay controls can only be used to control a single replay operation at any given time. This means that the controls can be used to either record, replay or remain idle. The replay state can be changed at any time using the controls in the upper left corner and is also indicated by the selected button:

.

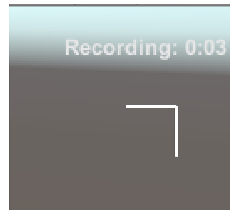
- **Live Mode:** Live mode allows gameplay to continue as expected. All replay objects have their components restored gameplay mode and physics and animation systems can control objects as normal.
- **Record Mode:** All replay objects are recorded at a fixed rate based upon the recording interval as specified via the settings window. All active ReplayObject s in the scene will be recorded to a memory storage target.
- **Playback Mode:** Playback mode allows you to view the recorded data and see the replay as it was recorded. All replay objects will be prepared for playback which involves disabling various game systems such as physics and scripts which could otherwise cause the object to move out of playback position. The replay system will then proceed to recreate the recording by restoring scene snapshots or key frames.

Record Mode

The replay controls include a record mode which is used to record replay objects in the scene over a period of time. The record state is indicated by the framing borders being displayed as part of the UI

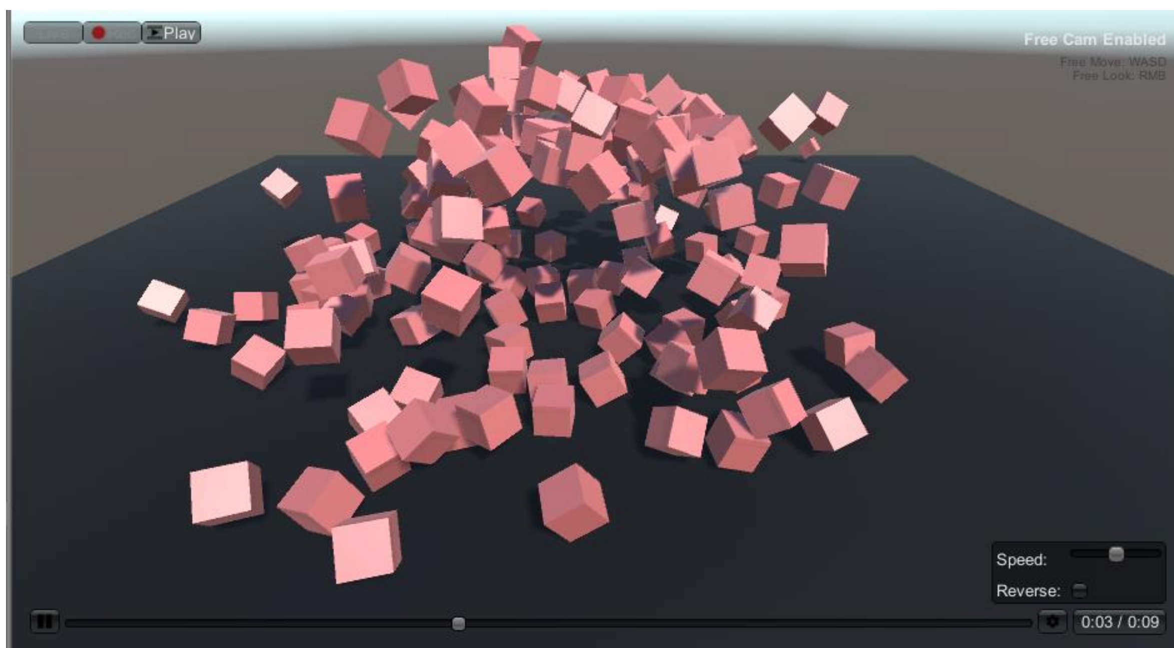
along with the current record duration in seconds. The replay controls will always store its recorded data in a memory target with an unlimited size meaning that long recordings are possible.

You can start recording with the replay controls UI by selecting the 'Rec' mode button which will begin capturing the scene.



Playback Mode

The replay controls also provide a convenient way to view the last recorded segment with full control via a playback seek control as well as speed and direction controls. The replay controls will also offer free cam perspective during playback meaning that you can move the camera around the scene using navigation keys to view the replay from any perspective.



- **State Controls:** As previously mentioned, the state controls allow you to switch between Live, Record and Playback modes offering full testing abilities.
- **Free Cam Hint:** The free cam hint is only displayed while in playback mode and indicates that you are able to move the camera around using the navigation controls. Keyboard and mouse control hits will also be displayed here.
- **Playback Speed:** The speed that the replay will be played at. The speed slider allows speeds between 0-2 to be specified where 1 is the default playback speed. A value of 0 would cause the playback to halt whereas a value of 2 would cause playback to run twice as fast. Note that the GUI slider is limited to 0-2 for ease of use but the replay system accepts much larger values via code.
- **Playback Direction:** Used to toggle between forward and reverse playback.

- **Playback Time:** Displays the current playback time value in seconds for the recording (Also indicated by the seek slider position) along with the total duration of the recording in seconds.
- **Playback Settings:** Used to show/hide the playback options popup containing the speed and direction controls.
- **Playback Slider:** The playback slider indicates the current playback position in relation to the overall recording. The slider can also be used to seek to different points in the replay by dragging or clicking along the slider bar. Note that interpolation is not available while seeking so snapping or jumping may occur when slowly scrubbing.
- **Play/Pause:** Allows playback to be paused or resumed at any point.

Free Cam Mode

One of the advantages of using a state-based replay system is that you are able to view the replay from any camera angle, or even multiple cameras in succession in order to create a highlights reel or similar. This is possible because the replay is rendered in real-time using the active camera.

The ReplayControls component makes use of this feature by allowing a free cam mode during playback which allows you to fly the camera around the scene as a replay is running. While in playback mode, you will see in the upper right corner that 'Free Cam' mode is enabled, meaning that you can manipulate the camera using the following controls:

- **W:** Move the camera forward relative to the current camera heading.
- **S:** Move the camera backwards relative to the current camera heading.
- **A:** Move the camera left relative to the current camera heading.
- **D:** Move the camera right relative to the current camera heading.
- **RMB + Drag:** Pan / tilt the camera angle based upon the mouse movement.

Exiting and re-entering playback mode will cause the free cam to be reset to its initial position which will be the position of the active rendering camera when entering playback mode.

> **Note:** *in order to preserve any gameplay cameras in the scene, the replay system will create its own camera that will be used during free cam mode which will adopt the position and rotation of the active scene camera. This will give the effect of moving the current scene camera but in actual fact, scene cameras will be left untouched.*

Replay Techniques

Replay Animation

Recording and replaying animated objects is a common use case of Ultimate Replay 2.0 and as a result we have added support for replaying Animator components, as well as support for IK animations via an alternative approach.

The ReplayAnimator component can be attached to a game object in order to record its animations. You will need to assign the ObservedAnimator property of the ReplayAnimator component to the Animator that you want to record and replay. After that, your animations should be recorded and replayed seamlessly by the replay system. Take a look at the ReplayAnimator section for more detail.

Some games may make use of IK animation to position bones via scripts to reach a target pose. Ultimate Replay 2.0 can also support IK animation, although a different approach needs to be used to setup the object. Essentially, each bone in the object skeleton that you wish to record should have a ReplayTransform component attached, and a single managing ReplayObject component at the root. If this sounds too complicated then don't worry, we have created a editor tools to help setup these replay component properly. The Replay Humanoid Configurator can be used to add the necessary replay components. Take a look at the Replay Humanoid Configurator section for more information.

Replay Ragdolls

Some games may make use of physics-based ragdolls for enemy deaths or similar which can also be recorded and replayed by Ultimate Replay 2.0. The process of recording a ragdoll character or similar is much the same as recording IK animation and requires that each bone in the skeleton has a ReplayTransform component attached. If your ragdoll has a humanoid structure, then setup is made quite simple using the Replay Humanoid Configurator which automates this process. For generic rigs, you will need to manually attach the recorder components which is a little tedious but worthwhile.

These simple rules will ensure that your replay components are placed on the correct objects in the ragdoll hierarchy:

1. A ReplayObject component is required on the very root of the ragdoll object. This will usually be the highest object in the hierarchy.
2. A ReplayTransform should be added to every bone in the hierarchy. An easy way to do this to setup your ragdoll using the Unity ragdoll window, and then add a ReplayTransform component to every object in the hierarchy that has a 'Character Joint' component attached.
3. Any ReplayTransform components that are not attached to the very root of the object should have their position and rotation options set to record in local space.

Here is an example setup to give you a better idea (This example assumes that you have already setup your ragdoll in Unity):

-Root	ReplayObject , ReplayTransform *(World Space)*
--Bone 1	*Character Joint, Replay Transform (Local Space)*
--Bone 2	*Character Joint, Replay Transform (Local Space)*
---Bone 3	*Character Joint, Replay Transform (Local Space)*
------Bone 4	*Character Joint, Replay Transform (Local Space)*

After attaching replay components following the structure above, you can immediately test the scene to ensure that everything is working correctly. It may also be worth taking a look at the included killcam demo scene which uses the ragdoll replay technique. The demo scene can be found at 'Assets/Ultimate Replay 2.0/Demo/Killcam.unity'

Killcams

Killcams are another use case that Ultimate Replay 2.0 fully supports. A killcam is usually used to replay the last few seconds of the game when a player is killed so that they can see the death from the point of view of the shooter. If we break down this problem, we can see that the following things are required in order to create a working killcam system:

- Continuous recording of the last (n)seconds of gameplay
- The ability to view the replay from different perspectives
- A scene to playback the recording without outside influences. For example, other players in a networked game.

Ultimate Replay 2.0 has support for endless continuous recording when using a ReplayMemoryTarget as the storage device. Generally, a killcam will only need to use memory storage as any old data is no longer relevant and can be discarded. A ReplayMemoryTarget has a time value in seconds that can be assigned to which represents the maximum length of recording that can be stored. This is not a normal limit though as the time value is a negative offset. Ie. This time value is used to constrain the recorded data to the last x amount of seconds so that the smallest amount of memory is used and endless recording is possible without running into memory usage issues.

Note that some games may like to use a killcam but also have the ability to record complete game sessions at the same time. Ultimate Replay 2.0 can now support an unlimited number of simultaneous replay operations so it is possible to both record in memory for the in-game killcam, and stream the entire game session to file or another storage device if required.

Another key aspect of a killcam is that the replay viewpoint is from the perspective of the shooter. This means that you get to view the actions of another enemy/player leading up to the death as if you were in their shoes. Luckily, this is something that is very easy to achieve using Ultimate Replay 2.0 due to the state-based approach used for replays. Ultimate Replay 2.0 renders all replays in realtime using the active camera. This camera can be positioned at any location and moved as required in order to view the replay from any location.

The easiest way to achieve this would be to attach a secondary camera to the enemy character model assuming a first person killcam is required. This camera should be setup as a first-person view for the enemy but will not be activated until you need to view the replay. You will also need to record the transform of this camera so that the players movements are captured. Then when entering playback mode, it is simply a case of switching cameras to view the replay from the shooter perspective. This step can be repeated for each enemy/player in the game so that all potential shooters can be used as the viewing perspective.

One final thing to consider when implementing a killcam is where a replay will be constructed. Ultimate Replay 2.0 uses the recorded scene objects to reconstruct the scene which could potentially cause issues. For example: If you have a networked multiplayer game where you send player updates across the network, when you switch to replay mode, other clients may receive the results of the replay rather than the gameplay. In this scenario, you would need to suspend network

updates while the replay is running so that the local scene is not synced to other clients, and also that other clients do not affect the local scene used for the replay which could cause inaccurate playback. It is not a major issue, but something to take into account when designing a killcam system.

Ultimate Replay 2.0 can be used to create ghost vehicles for a racing game and it is now much easier and better supported than in the original asset. A ghost vehicle is used in racing games to show the player their previous best racing line/time, usually via a semi-transparent non-collidable car. To setup a ghost vehicle system there are a few things to consider:

- The player vehicle is used for recording
- Usually a different ghost vehicle object is used for playback
- The player could beat their previous time and the ghost vehicle should display the fastest lap only

Recording the player vehicle is straight forward and can be achieved with the built-in recorder components, namely the ReplayTransform component. This would record the vehicles transform as it drives around the track. The problem comes when we need to replay the recording. Usually with Ultimate Replay 2.0 you could just call BeginPlayback and the replay would run just fine. The issue though is that the replay would be played back on the player car instead of the ghost vehicle which is not desirable. This can be resolved quite easily though using identity transfer to allow the ghost vehicle object to take on the identity of the player car for playback purposes. Take a look at the identity transfer section for more information.

Using the identity transfer technique, we can allow the player car to record the information as it drives around the track, and then replay that information onto a different ghost vehicle car. This means that the player car is not taken over by the replay and is free to drive another lap. It also means that a completely different game object usually with a different visual appearance can be used for the ghost vehicle which is an ideal solution.

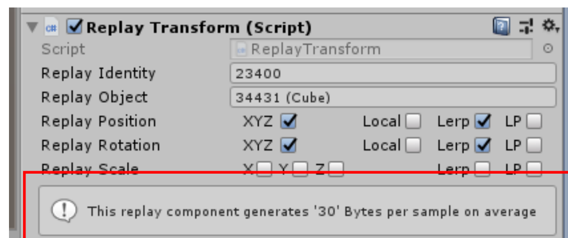
Another thing to consider when implementing a ghost vehicle system is that the player could beat their previous lap and the ghost vehicle would need to use this latest replay to reflect this. This means that we need to record every lap that the player car completes, just in case they posted a faster time. This is quite simple to do using the following rules:

1. Start recording the player car when they cross the start/finish line and wait for the lap to complete.
2. Check if we have any previous times posted.
    a. If yes, then we check the newly posted lap time to see if it was faster and store the recording if so. The recording can safely be discarded if the time was not a new record as a previous recording will exist.
    b. If no, then we keep a reference to the storage target that contains the recording and start recording the player car again using a new recording target.
3. Create a ghost vehicle using the identify transfer process and replay the saved storage target which contains the fastest lap time.
4. Loop back to step 2 or until the player quits the game.

Replay Statistics

Ultimate Replay 2.0 uses a state-based approach and needs to store data for each recorder component of every ReplayObject in the scene in order for replays to be captured. On top of this, snapshot frames are captured in quick succession, often at over 16 frames per second which can result in quite a bit of data to store. Ultimate Replay 2.0 features some highly effective lossless compression techniques to keep that figure low but you can also save storage space on a per component basis. Many recorder components such as ReplayTransform have inspector properties which control which data is recorded and how accurately. By tuning these components to only record what is needed, you will be able to reduce the overall storage requirements for your replays.

It is important to note that saving a couple of bytes per component may not seem like much, but at over 16FPS and many recorder components in the scene, it does in fact add up to make quite a difference to the overall size. You will notice that any component deriving from ReplayBehaviour will display useful stats in the inspector window indicating how much data is generated per average sample. This figure is essentially the amount of data you can expect to be produced by the component for every recording sample prior to compression techniques.



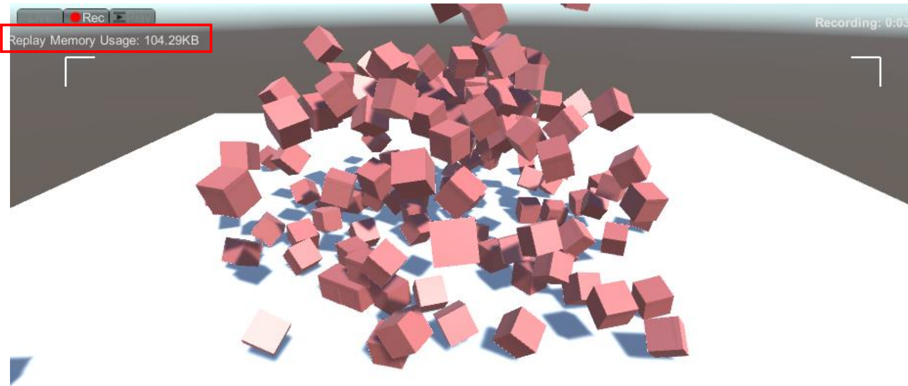
Note: *Some recorder components may only be able to provide accurate data statistics while in play mode. An example would be the ReplayAnimator component which displays a sample size of '0' while the game is not running, but displays accurate statistics in play mode.*

When you change the properties of the recorder components, you will see that the statistics data updates in real time to give you immediate feedback on the storage requirements for the component. This is highly useful to fine tune a recorder component for playback accuracy vs storage size.

Storage Statistics

Per component statistics are useful to fine tune recorder components, but it does not give an overall idea of the amount of actual storage space required by all replay objects in the scene. Ultimate Replay 2.0 addresses this issue but gathering statistics from all active storage targets to give a single total usage value in real time. There is a dedicated component to display this information as UI text but an API is also available so that you can use that data in any way you need.

The ReplayStatistics component can be attached to any game object in the scene and will display the realtime total storage value used by all storage targets. You will see that value change as recording operations are updated and new data is written to a storage device. The component can be added to a game object by going to 'Add Component -> Scripts -> Ultimate Replay -> Replay Statistics'.



As mentioned previously, an API is also available to access this data via script if required. The 'UltimateReplay.Statistics' namespace contains methods to access this information such as:

C# Code

```
1  ReplayStorageTargetStatistics.CalculateReplayMemoryUsage();
```

This method is used by the ReplayStatistics component and will return an integer value representing the total amount of bytes used by all active storage targets combined. Any storage target that exists in memory will be included in the calculation as the constructor is used to register for statistics. Dead storage targets are not included and by setting all references to a storage target to 'null', the storage target will no longer affect the calculation. In the same namespace, there is also the useful ReplayStatisticsUtil type which contains many helper methods to convert a byte value to the largest possible unit such as 'KB', as well as get the string representation of the unit.

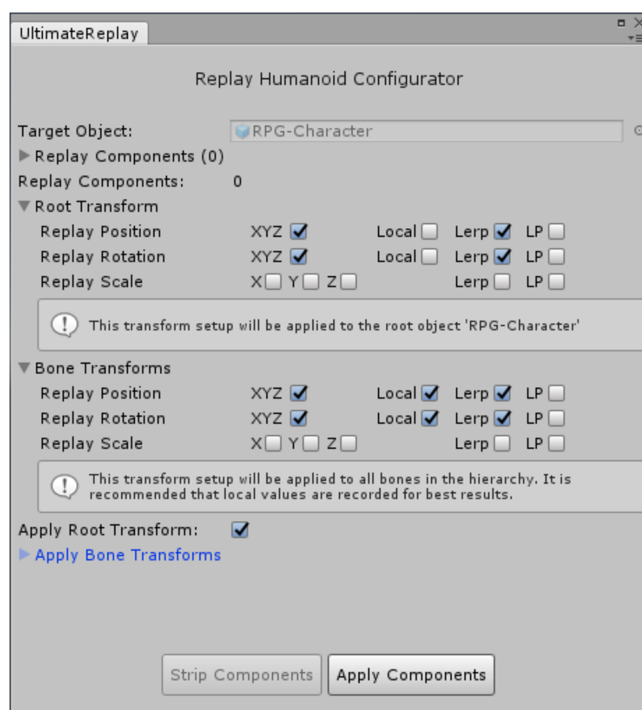
Replay Humanoid Configurator

The replay humanoid configuration is used to setup humanoid characters for bone recording. Bone recording may be required I order to record non-standard animation techniques like inverse kinematics which are not supported by the Animator. You can also use this approach to record ragdolls or as an alternative to the ReplayAnimator component.

Essentially in order for this technique to work correctly, a number of ReplayTransform components need to be added to each bone transform in the object hierarchy along with a single managing ReplayObject component on the root. Doing this manually would be time consuming and possibly error prone so we have created a simple editor tool to automate the process to make things easier.

The Replay Humanoid configurator window can be opened by going to 'Tools -> Ultimate Replay -> Setup -> Replay Humanoid'. This will open the setup window which operates on the currently selected game object.

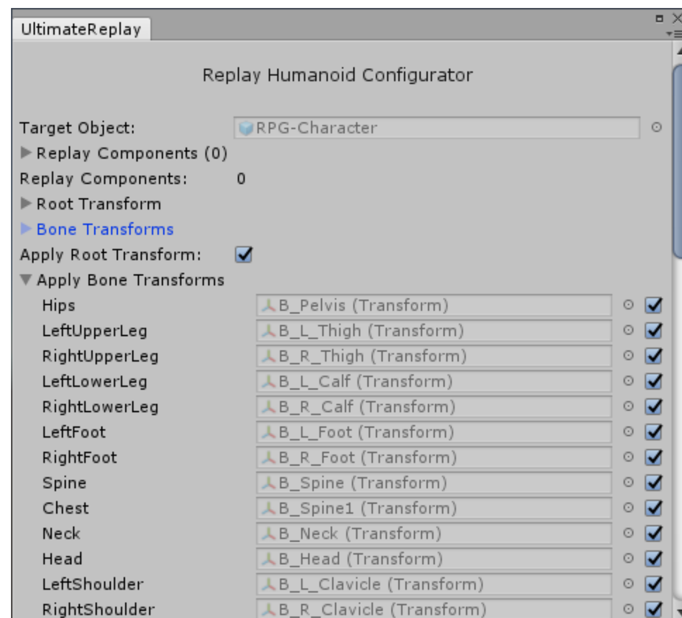
> **Note:** *The selected game object must have an Animator component attached with a suitable humanoid avatar assigned. If the selected object is not suitable, then the setup window will display a warning to indicate this.*



- **Target Object:** This field displays information about the current selected object so that you know which object will be configured by the setup tool.
- **Replay Components:** This section will display a collection of replay components which already exist on the selected object. Note that components attached to child objects will also be listed here as well as ReplayObject components. If one or more components exist, then you can use the 'Strip Components' button to remove them so that you start the setup with a clean object.
- **Root Transform:** This section displays ReplayTransform information which will be applied to the root object. In this case, the 'RPG-Character' object will receive a ReplayTransform component with identical properties when the 'Apply Compontents' button is clicked. The

root transform should almost always use world space for recording position and rotation values.

- **Bone Transform:** This section displays ReplayTransform information which will be applied to every selected bone of the humanoid skeleton. This will allow the movement of each bone to be recorded and replayed, allowing for animation and physics movements to be recorded. Bone transforms should almost always use local space for recording position and rotation elements of the transform.
- **Apply Root Transform:** When enabled, the root transform information specified in the 'Root Transform' section will be applied. If this option is unchecked, the root object will not receive a ReplayTransform component.
- **Apply Bone Transforms:** This section contains a list of bones that should be recorded by the replay system. Each bone is named as per the avatar description and can be selected or deselected to specify whether replay components should be applied to that particular bone. By default, all bones are enabled and will receive replay components.



Once you are happy with your root and bone transforms, and have selected the bones that you want to be recorded, you can then hit the 'Apply Components' button. This will apply the necessary replay components to the appropriate bones in the hierarchy using the properties specified. You should then have a fully recordable humanoid character that supports standard animation, IK animation, ragdolls and other physics-based bone manipulation.

Integration

This section will cover common integration techniques that can be used to better incorporate Ultimate Replay 2.0 into your game project.

Pooling Support

Ultimate Replay 2.0 is able to support dynamic prefab instantiation and destruction during recording meaning that game objects can be added and removed during recording and playback will replicate this behaviour. In order to do this, Ultimate Replay 2.0 will Instantiate and Destroy prefab instances as required using the Unity API. This may not be desirable if your game implements a pooling solution as you may like to have full control over the creation and destruction of objects. The good news is that Ultimate Replay 2.0 was designed with pooling in mind and it is possible to completely take over this process.

Ultimate Replay 2.0 has 2 static delegates which can be used to add Instantiate and Destroy listeners. These listeners will then be invoked every time Ultimate Replay 2.0 needs to instantiate or destroy a game object in order to achieve an accurate playback state. These delegates are OnReplayInstantiate and OnReplayDestroy and can be found in the UltimateReplay type. You can add listeners as shown below:

C# Code

```
1  class Example : Monobehaviour
2  {
3      void Start()
4      {
5          UltimateReplay.OnReplayInstantiate = CreateObject;
6          UltimateReplay.OnReplayDestroy = DestroyObject;
7      }
8  
9      GameObject CreateObject(GameObject prefab, Vector3 pos,
   Quaternion rot)
10     {
11         // Create instance from pool...
12     }
13 
14     void DestroyObject(GameObject target)
15     {
16         // Remove instance from pool...
17     }
   }
```

Note: *If an instantiate or destroy handler fails by returning null or throwing an exception, Ultimate Replay 2.0 will default to the standard approach of using Instantiate and Destroy to avoid playback issues.*

How do I...

Get the replay duration?

Once you have recorded a replay to a storage device, you can access the overall duration of the recording using the Duration property of the storage target. The duration will be set to zero if no data has been recorded.

C# Code

```
1  ReplayStorageTarget target = new ReplayMemoryTarget();
2  
3  // Get the recording duration is seconds
4  float duration = target.Duration;
```

Set playback time?

You can seek to particular a particular time stamp of a recording during playback. There are 2 methods of the ReplayManager which can be used to achieve this.

C# Code

```
1  ReplayHandle handle = ReplayManager.BeginPlayback();
2  
3  // Seek to the 2 second mark
4  ReplayManager.SetPlaybackTime(handle, 2f, PlaybackOrigin.Start);
```

This method accepts a replay handle which should be a valid playback handle returned by BeginPlayback. The second parameter is the time stamp value that you want to seek to in seconds. Note that this value is relative to the specified PlaybackOrigin. The final parameter is the seek origin which indicates where the specified time value should be offset from. This works in a similar way to file system seeking and allows you to seek relative to the start of the recording, the current recording position and the end or the recording. Negative time values are accepted since seeking relative to the end of the recording requires a negative time offset.

C# Code

```
1  ReplayHandle handle = ReplayManager.BeginPlayback();
2  
3  // Seek to the middle of the recording
4  ReplayManager.SetPlaybackTimeNormalized(handle, 0.5f,
5  PlaybackOrigin.Start);
```

This method also allows you to seek though a recording during playback but only accepts normalized values. The first and last parameters are the same as above but the second parameter is now a normalized value between 0 and 1. When a playback origin of Start is specified, an offset value of 0 would indicate the start of the replay and a value of 1 would represent the end of the recording.

Replay in reverse?

The replay manager has a method called SetPlaybackDirection which can be used to change set the direction that a replay will play.

C# Code

```
ReplayHandle handle = ReplayManager.BeginPlayback();

// Change playback direction to reverse
ReplayManager.SetPlaybackDirection(handle,
PlaybackDirection.Backward);
```

The method accepts a playback handle which should be a valid replay handle returned by BeginPlayback and a PlaybackDirection value which can either be Forward or Backward.

Replay in slow motion?

The playback speed is determined by the time scale value for the replay. By default, this time scale is set to a value of 1 which represents normal playback speed. A value of 0.5 would cause playback to run at half the speed. You can set the time scale value for a replay playback operation using the replay manager method called SetPlaybackTimeScale. Note that you can also enabled reverse playback by passing a negative value.

C# Code

```
ReplayHandle handle = ReplayManager.BeginPlayback();

// Change playback speed to 1/2
ReplayManager.SetPlaybackTimeScale(handle, 0.5f);
```

The method accepts a playback handle which should be a valid replay handle returned by BeginPlayback and a float value which represents the time scale value.

Quickly test my scene?

You can use the built in ReplayControls component in order to quickly test recording and playback to make sure it is working as expected. The replay controls can be added to any game object in the scene and when in play mode will display a UI that can be used to control the recording and playback of replays. All of the ReplayManager interaction is handled by the replay controls so it is a quick and easy way to test out your scene. Replay controls can be easily added to the scene by going to 'Tools -> Ultimate Replay 2.0 -> Replay Controls'.

Create a killcam?

A killcam is can be implemented quite easily using Ultimate Replay 2.0, especially with the Rolling memory target storage capabilities. We have created a dedicated section of the user guide which covers the various techniques used by killcam replay systems and how they can be implemented. We have also included a simple demo scene which shows how a basic killcam can be implemented. You can find the demo scene at 'Assets/Ultimate Replay 2.0/Demo/Killcam.unity'.

Create a ghost vehicle?

A ghost vehicle is a common use case for Ultimate Replay 2.0 and we have created a dedicated section of the user guide to cover the necessary techniques to create a successful ghost vehicle replay system. We have also included a simple ghost vehicle demo scene with the asset which can be found at 'Assets/Ultimate Replay 2.0/Demo/GhostVehicle.unity'.

Having difficulty finding information about a particular aspect of Ultimate Replay 2.0? Contact us via the forum or by email and we can add your question to this section to help other users. Contact details are also included at the end of this document.

Report a Bug

At Trivial Interactive we test our assets thoroughly to ensure that they are fit for purpose and ready for use in games but it is often inevitable that a bug may sneak into a release version and only expose its self under a strict set of conditions.

If you feel you have exposed a bug within the asset and want to get it fixed then please let us know and we will do our best to resolve it. We would ask that you describe the scenario in which the bug occurs along with instructions on how to reproduce the bug so that we have the best possible chance of resolving the issue and releasing a patch update.

http://trivialinteractive.co.uk/bug-report/

Request a feature

Ultimate Replay was designed as a complete replay system, however if you feel that it should contain a feature that is not currently incorporated then you can request to have it added into the next release. If there is enough demand for a specific feature then we will do our best to add it into a future version. Please note, requested features should fall within the scope of the asset and unrelated or overreaching features will not be added.

http://trivialinteractive.co.uk/feature-request/

Contact Us

Feel free to contact us if you are having trouble with the asset and need assistance. Contact can either be made by the contact options on the asset store or via the link below.

Please attempt to describe the problem as best you can so we can fully understand the issue you are facing and help you come to a resolution. Help us to help you :-)

http://trivialinteractive.co.uk/contact-us/