User Guide

Ultimate Replay

A simple and effective state based replay system for Unity

Trivial Interactive

Version 1.0.0

Ultimate Replay 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.
- Ghost vehicles in racing games.
- 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.
- 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 script.
- Built-in support for transform recording.
- Built-in support for audio recording.
- Easily create your own component recorders to expand the capabilities.
- Memory recording can be setup as continuous or as a rolling buffer configuration.
- 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.

Basic API

Ultimate Replay can be mostly setup via the Unity editor with very little scripting required to get up and running however you will need to inform the replay system when you want to perform basic replay tasks such as starting or ending recording. In order to do this you will need to use the static API that Ultimate Replay provides in the form of the 'ReplayManager' class. For basic usage, you will not need to use any other classes as all of the core functionality is encapsulated in the Replay Manager.

All types used by the Replay system are defined in the 'UltimateReplay' namespace and sub-namespaces which you will need to import in order to use the API.

C# Code

```
1  using UnityEngine;
2  using UltimateReplay; // Use the Ultimate Replay API
3
4  public class ReplayExample : MonoBehaviour
5  {
6      void Start()
7      {
8          // Call the Ultimate Replay API
9          ReplayManager.ForceAwake();
10     }
   }
```

Replay Manager

The Replay Manager uses a static interface for ease of use allowing you to access it from anywhere in your game code. The Replay Manager is also a component that can be attached to a game object in a scene which allows you to modify its settings via the property inspector in the Unity editor. Once the game starts, the replay system will attempt to find any Replay Manager components in the scene and if one is not found then it will be automatically created using the default settings. This means that the static API is safe at all times, even if you don't have a Replay Manager in the Scene. In most cases it will be adequate to create a Replay Manager scene component in the first loaded scene of the game and make the owning object persistent so that it survives scene changes. This allows the same Replay Manager to be used throughout the game.

There are a number of static methods in the Replay Manager class that control the state of the replay system. Some of the most useful methods are shown below:

- **BeginRecording(bool cleanRecording)**: The replay system will begin replay recording in the next frame using the current recording settings.
    - **cleanRecording**: When true, any existing replay data will be disposed of so that the new recording starts as fresh.
- **PauseRecording(void)**: The replay system will pause the current recording. If the replay system is not recording then this method will have no effect.
- **ResumeRecording(void)**: The replay system will resume the current recording. If the replay system is not in a paused state then this method will have no effect.

- **StopRecording(void)**: The replay system will finalize the recorded data and prevent further data from being recorded.
- **DiscardRecording(void)**: Causes all existing replay data to be disposed of so that a fresh recording can take place. This method is called automatically when 'true' is passed to 'BeginRecording'.
- **SetPlaybackFrame(float offset, PlaybackOrigin origin)**: Sets the replay playback to a specific frame in the recording as specified.
    - **Offset**: The time (in seconds) to seek to (based on 'origin'). If 'origin' is set to 'PlaybackOrigin.End' then the offset will be negative relative to the end time of the recording.
    - **Origin**: The location in the recording to offset from.
- **SetPlaybackFrameNormalized(float normalizedOffset, PlaybackOrigin origin)**: Sets the replay playback to a specified frame in the recording based on a normalized value between 0-1 representing the offset into the recording irrespective of the actual recording duration.
    - **Offset**: A normalized value between 0-1 representing the offset into the recording where 0 represents the first frame and '1' represents the end frame.
    - **Origin**: The location in the recording to offset from.
- **BeginPlaybackFrame(void)**: Use this method after a call to 'SetPlaybackFrame' or 'SetplaybackFrameNormalized' to cause the playback frame to be 'shown'. Only a single frame be replayed.
- **BeginPlayback(bool fromStart, PlaybackDirection direction):** The replay system will begin playback of the active recorded replay using the specified values.
    - **fromStart:** When true, the playback will begin from the start of the recording (offset 0).
    - **Direction:** The direction in which the recording should be replayed.
- **PausePlayback(void):** The replay system will pause on the current frame of playback causing all replay objects to freeze in their current state. If the replay system is not in playback mode then this method will have no effect.
- **ResumePlayback(void):** The replay system will continue playback using the current playback settings. If playback is not already paused, then this method will have no effect.
- **StopPlayback(void):** The replay system will exit playback mode and return to 'live' mode so that the gameplay can continue.

Replay Time

Another useful class that Ultimate Replay provides is the Replay Time class which provides information such as the current playback time and the delta between replay frames which can be used for interpolation purposes. You will note that this class is similar to the 'Time' class in unity which is intentional for ease of use.

- **Time (get):** The current replay playback time in seconds.
- **Delta (get):** A normalized value between 0-1 representing the delta between playback frames. This value can be used for interpolation purposes where a low record rate has been used.
- **TimeScale (get, set):** A scalar value indicating the speed at which playback will occur where 1 represents normal speed. You can modify this value to achieve slow motion replays or high speed replays as needed.

Quick Start

If you are just interested in getting the asset setup and integrated into your project as quickly as possible then skip ahead to the Minimal Setup section that provides a step by step guide of the process.

As an alternative, you can expand on one or more of the demo scenes provided using the attached scripts and prefabs as a foundation. This method requires almost no knowledge at all of scripting as the solution works out of the box and is easily modifiable to suit many different applications.

Examples

Included in the package are a number of demo scenes to show the functionality that is included with Ultimate Replay. Each demo scene will typically contain a few common game objects that contain the required scripts to get Ultimate Replay working. These shared objects are located under a root game object called 'ReplayCommon' in order to keep the scene organised.

All of the demo scenes included with ultimate Replay are located in the following folder: 'Assets/UltimateReplay/Demo'.

Cube Demo

The cube demo scene shows how a single objects transform can be recorded by Ultimate Replay. The cube will fall to the ground plane upon starting the game using the physics system to provide gravitational force and collision response. You can then use the replay controls to playback the recording by switching to playback mode using the on screen GUI controls at which point you should see a replay of the cube falling to the ground. Take a look at the later 'Replay Controls' section for help using the replay controls.

Note that during playback the cube is no longer affected by the physics system and is only replaying recorded key frames to track its transform. If you select the 'ReplayCube' object in the editor you will see that it has 2 replay components associated with it. A 'ReplayObject' and a 'ReplayTransform'. The 'ReplayObject' component is required in order to inform the replay system that this object should be recorded and the 'ReplayTransform' component is specially tailored to recording and storing transform data for playback at a later time.

The 'ReplayObject' and 'ReplayComponents' are covered in more detail later in this document.

Minimal Setup

Ultimate Replay is designed to be as simple and user friendly as possible and as a result can be setup very quickly with very little coding skill required. The following steps show the minimal setup required to get a basic replay scene setup and working:

1. The first step you will need to take is to create a 'ReplayManager' in your scene which is used to manage the recording and playback of any replay objects. Due to the way Ultimate Replay is designed, this is not a required step because if a replay manager is not detected in the scene then Ultimate Replay will simply create one. It is however recommended to setup a replay manager in the scene as it allow you to control important settings such as record rate via the inspector window.

   To create a replay manager simply go to 'GameObject->Create Empty' which will place an empty game object in the scene. With this object selected, go to 'Component->Scripts->UltimateReplay->Replay Manager' which will attach a new replay manager component to the object.

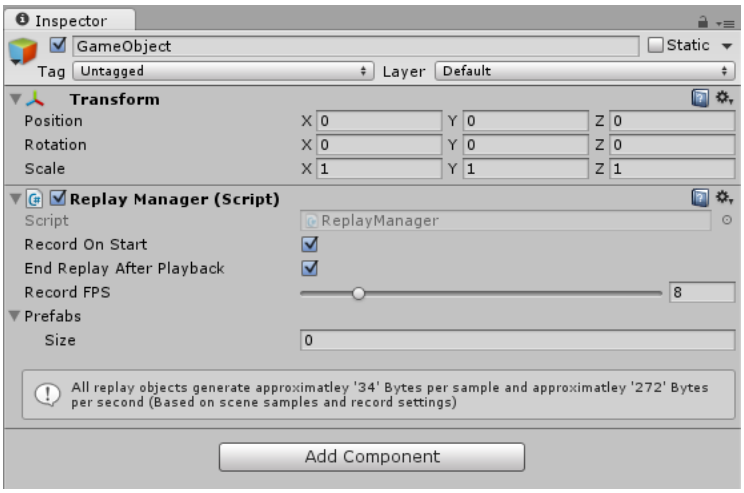

Figure 1

2. Next you will need to associate a 'ReplayTarget' with the replay manager which will be responsible for storing the replay data. Again this step is not necessary because Ultimate Replay will create a default replay target if none is found (ReplayMemoryTarget by default) but it is recommended that you explicitly create the script so you have control over its settings via the inspector window.

   In order to create a replay target ensure that the game object that you attached the replay manager script to is still selected and then go to 'Component->Scripts->UltimateReplay.Storage->Replay Memory Target'. The component will be added alongside the replay manager and will be automatically detected when the game starts.

   Note: *Currently only 'Replay Memory Targets' are supported which allows either buffered or rolling buffered memory storage meaning that very long recordings will be difficult to support. We will be adding a 'Replay File Target' in the near future that supports asynchronous streaming to and from file using compression to achieve small file sizes.*

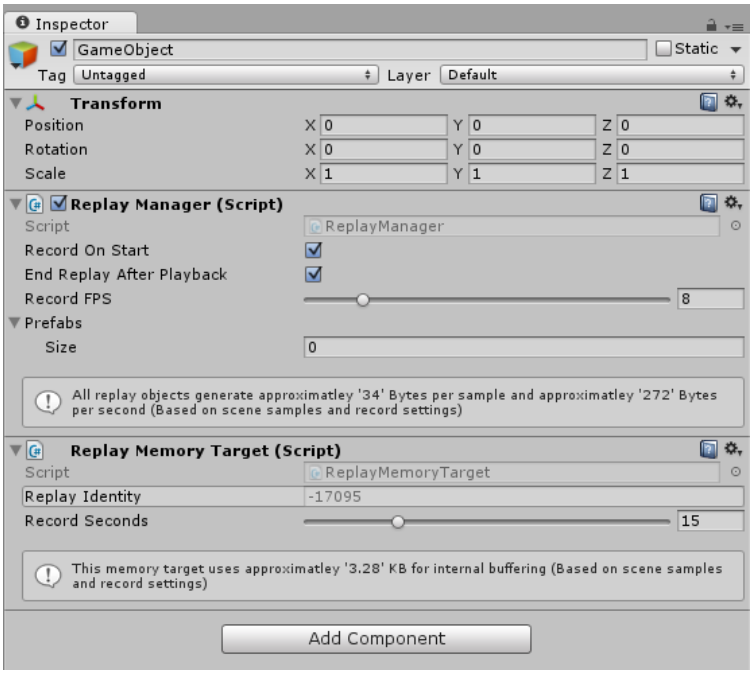

Figure 2

3. Once you have the replay manager setup you will need to start identifying any objects that should be recorded by the replay system. For this example we will assume that we have a single object and that we want to record its position over time.

   In order to record the transform of an object we need to attach 2 components to the object which will auto register the object with the replay system and associate it with a replay identity. The first component we need to add is a 'ReplayObject' component which marks the game object as recordable. The second component we need to add is a 'ReplayTransform' which is responsible for recording the position rotation and scale of our game object.

   Since a 'ReplayObject' component is required by all types that derive from 'ReplayBehaviour' (including ReplayTransform), Ultimate Replay will automatically attach the 'ReplayObject' script if there is not one already on the object to make the process simpler.

You should now have a basic replay scene setup that is capable of recording and replaying an objects movement. In order to be able to control the recording and playback of the replay you can use the included replay controls GUI which provides most of the common functionality associated with recording and playback in a user interface that can be used in game. Take a look at the 'Replay Controls' section for help setting up and using the included GUI.

You can now enter game mode and begin recording the scene. Upon switching to playback mode you should observe that the gameplay behaviour of the object is reproduced. Now you can continue setting up other replay objects and fine tuning the replay settings to get the best recording / storage ratio.

Replay Considerations

Ultimate Replay uses a state based storage technique for recording gameplay data for playback at a later time. This approach means that any recorded objects need to be carefully managed during the recording and replaying processes. There are a few things you should consider when adding replay support to your game using Ultimate Replay:

- **Scripts don't run during playback:** In order for the replay system to be sure that replay objects stay where they are put, the replay system will disable any scripts attached to a replay object in order to prevent them from interfering. This will only occur during replay playback and the scripts will be re-enabled when switched back to game mode. If you want to ensure that your script continues to run during playback you can either make it derive from 'ReplayBehaviour' (See ReplayBehaviour' later in this document) or you can implement your own replay preparer (See Advanced Topics later in this document).
- **Physics components are disabled during playback:** Most physics components will also be disabled during playback to prevent the physics system from moving any replay objects out of their designated locations. Colliders will be disabled and rigid bodies will be set to kinematic to prevent forces being applied. If you need to keep some physics components alive during playback you will need to implement a custom replay preparer. See 'Advanced Topics' later in this document.
- **Other objects can affect replay objects:** Any other scripts running on non-replay objects will continue to execute during playback and as a result may also be able to manipulate replay objects causing them to move out of sync with the playback target snapshot. In order to prevent this you will need to carefully manage access to replay objects and ensure that they are not moved during playback. You can use the 'IsReplaying' property of the replay manager to conditionally move replay objects during gameplay. See the included .chm API documentation for more information.

Replay Prefabs

All of the examples you have seen up until now have involved recording scene objects which are nice and easy because they are always present in the scene during recording and are not likely to disappear unless they are specially scripted objects. So what would happen if these scene objects disappeared during recording? Ultimate Replay would simply look at the recorded data to determine which replay object should receive the deserialization call and would then notice that this object no longer exists in the scene and as a result would simply throw away the data.

This is no real problem until we try to playback the recording and find that the object that we destroyed in our scene is now no longer included in the playback meaning that the recording is inaccurate as it is missing objects. A similar problem can be observed when we instantiate a prefab during recording. Upon playback we will see that the object exists in the recording before it was actually created and as a result also causes inaccurate playback.

These problems can easily be fixed by making prefabs from objects that are likely to be spawned or respawned during recording, for example: bullets or explosion effects. It is likely that these objects will be stored as prefabs anyway due to the way Unity works. You are then able to register these prefabs with the active replay manager in the scene by adding to the 'prefabs' array of the replay manager component.

> **Note:** *Any prefab you register with the replay manager should have a unique name as it is used by the replay system to identify the object. Multiple prefabs with the same name may cause unexpected behaviour.*

It is important to note that all prefabs registered with the replay manager must have a 'ReplayObject' script attached to them even if there are no replay components that are recorded.

Once you have setup your prefab you can dynamically instantiate prefab assets or destroy prefab instances in the scene during recording and the replay system will then be able to accurately restore the scene state by creating or destroying objects as needed during playback. This dynamic spawning behaviour can be seen in the 'StressTest' demo that is included with Ultimate Replay where a large number of cubes are dynamically spawned during recording.

Dynamic Prefab Registration

As an alternative to registering replay prefabs via the inspector you can also register replay prefabs through code. This can be useful when you don't have a replay manager in the scene and are using an auto created replay manager. The following code shows how to register a prefab dynamically:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample: MonoBehaviour
{
    // Assigned via inspector window
    public GameObject bulletPrefab;

    void Start()
    {
        // Register the prefab with the replay system
        ReplayManager.RegisterReplayPrefab(bulletPrefab);

        // We can now spawn the bullet during recording
        Instantiate(bulletPrefab);
    }
}
```

Only prefab assets should be passed to this method otherwise an error will be generated to inform that the object will not be registered because it is not a prefab.

Replay Controls

Ultimate Replay includes a simple replay control UI implemented using the legacy immediate mode GUI system which can be used to control most of the common tasks associated with replay recording and playback. You can easily add the replay controls to your scene by going to 'GameObject->UltimateReplay->Create Replay Controls'.

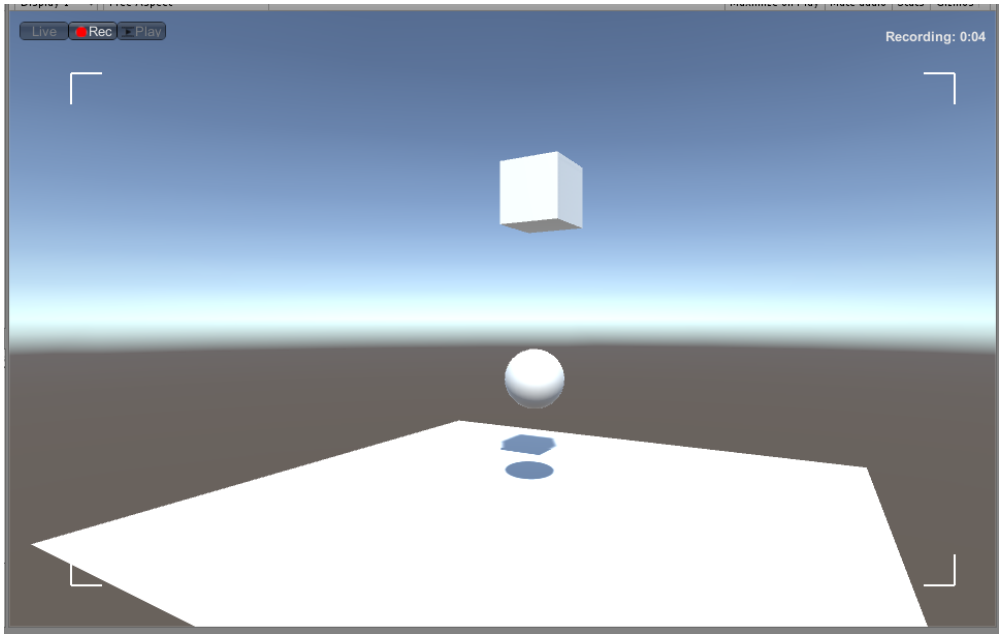

Figure 3 - Default replay controls in record mode

The replay controls include 3 different modes which determines the state the replay system is in. Depending upon the current state, replay objects may be manipulated by the replay system in order to restore previous scene states. The current state of the replay system can be changed at any time via the buttons in the top left corner of the GUI:

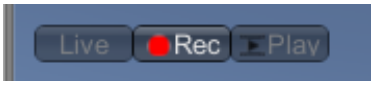

Figure 4 - Replay state controls

- **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 of the replay manager and the data is dispatched to the active replay target for storage.
- **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.

Playback Mode

The default replay controls include a playback mode which allows you to control the playback of the current recording by adjusting the time offset, playback speed and playback direction using a simply GUI interface.

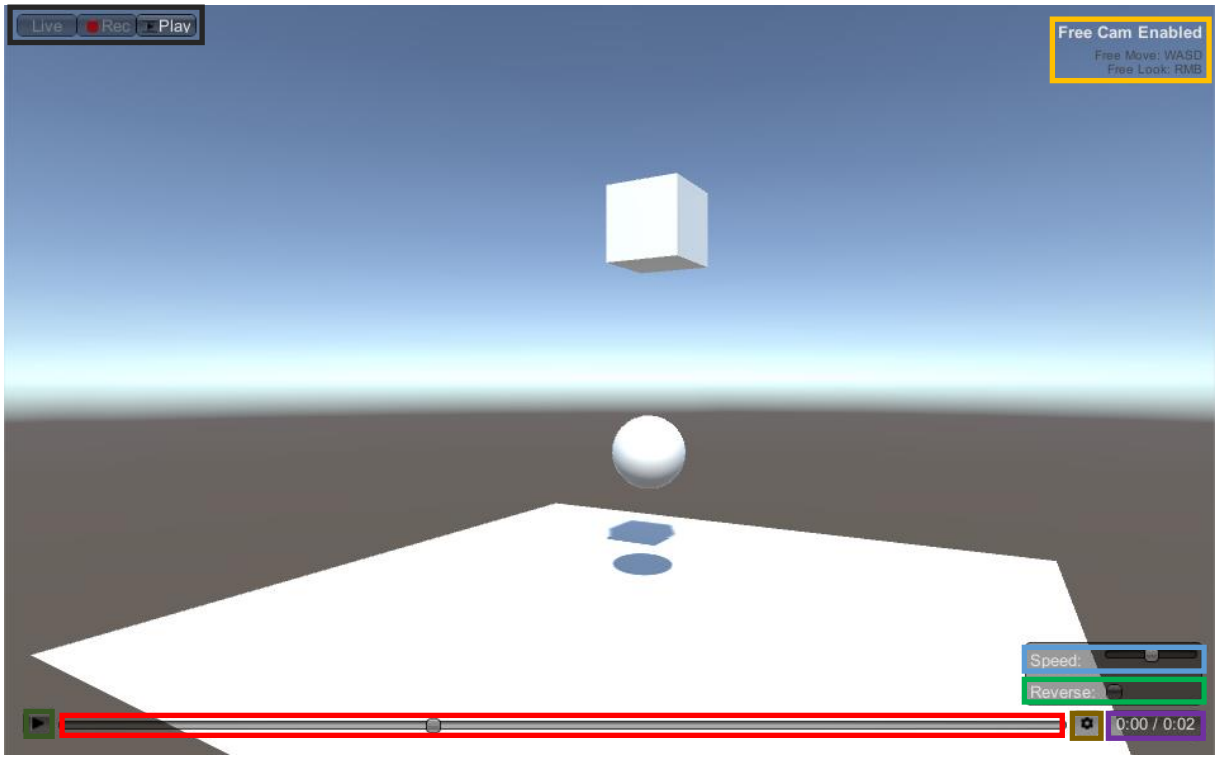

Figure 5

- **State Controls:** As described previously, control the current state of the replay system.
- **Free Cam Hints:** Control hints for the free cam mode. Take a look at the next section for more information.
- **Playback Speed:** The speed that the recording will be replayed at. The slider allows for float values between 0-2 where 1 is the default value. The replay system can handle much larger values than this but will need to be controlled via code (The GUI limits the value for ease of use).
- **Playback Direction:** The direction that the replay will be played in. when true, playback will be reversed.
- **Playback Time:** The current and maximum time values for the current recorded data. This is also displayed via the slider.
- **Playback Settings Toggle:** Shows / Hides the playback settings box.
- **Playback Slider:** The playback slider indicating the current playback frame for the recording. Using the slider you are able to seek to different locations in the recording. Note that seeking is achieved via frame offsets and as such will not be interpolated which may cause sudden frame jumps.
- **Play / Pause:** Pauses or resumes the playback depending upon the current pause state.

Free Cam Mode

One of the added advantages of using a state based replay system is that you are able to view the playback from any camera angle, even if it is not the camera that was used when the replay was recorded. To make the most of this feature the default controls include a simple free cam mode where you are able to break free of the current camera location and move freely around the scene to view the playback from different angles. Free cam mod is only enabled during playback mode and you can break free by moving via the WASD keys or by holding down the right mouse button and scrolling.

The controls for the free cam are described below and the speed values for movement and rotation can be changed via the replay controls inspector in the Unity editor.

- **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.

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 it will be left untouched.*

Replay Manager

The Replay Manager is the most important component and is responsible for all recording and playback operations. You will need a replay manager in your scene in order to use Ultimate Replay but if you call into the API at any point and a valid replay manager is not found, then Ultimate Replay will automatically create a replay manager object in order to prevent any errors. It is recommended that you manually create a replay manager in the scene as you will then have full control over its various settings:

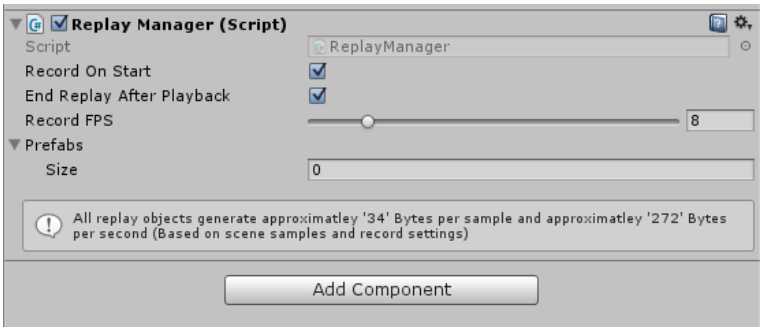

Figure 6

- **Record On Start**: Should the replay manager automatically begin recording the gameplay when the game is started. When this value is set to false, you will need to call 'BeginRecording' manually in order to record the gameplay.
- **End Replay After Playback**: Should the replay manager automatically switch back to live mode when a recording has finished playback. If this value is set to false then the playback will reset back to the first frame and continue playing in a loop.
- **Record FPS:** This is the main value that determines the interval at which scene snapshots will be recorded. This value directly affects how accurate the recording will be but comes at the cost of extra memory usage and CPU usage. You will need to find a suitable value to trade-off resource usage vs replay accuracy. The higher the record FPS the higher the replay accuracy will be but very low values can cause objects to be significantly out of place during playback as not enough state information is recorded. The best way to determine the correct value for your application is to decrease the FPS value slowly until you are dissatisfied with the replay and then increase it slightly. Due to the built in interpolation support, you may find that you can get away with relatively low record rates and still have the replay look natural and accurate.
- **Prefabs:** An array of game objects representing prefabs that can be used in game to spawn or despawn replay objects dynamically. You can also register prefabs dynamically via scripts if needed. Take a look at the 'Replay Prefabs' section earlier in this document for more information.

The replay manager is also able to provide some useful information about the amount of data generated by any active replay objects in the scene. This information is gathered by taking scene samples when required to determine roughly how much data is being generated which can then be used to calculate how much data is generated per second based upon the record rate. Obviously the lower you can get this value without compromising replay quality, the less memory and CPU resources you will use. This may be especially important for very long recordings or very complicated replay scenes where a large number of objects are being recorded.

Replay Target

Ultimate Replay uses components known as replay targets in order to store any replay data that is recorded. A replay target is essentially an interface that is able to store and fetch data upon request by the replay system. The actual implementation of the target is not known by the replay system nor is it important as the replay target abstracts the details of the storage method used, whether it be buffered memory storage or file storage. This means that you are able to introduce new storage components that store the replay data in another perhaps more efficient manner without requiring any changes to the core replay system.

A replay manager requires a replay target at all times and as a result will create a target if one is not assigned to avoid errors. By default the replay manager will create a 'ReplayMemoryTarget' as its assigned storage device using default values only. You can associate a replay target with a replay manager by simply attaching the desired replay target component onto the same object the replay manager component is attached to. The replay manager will then automatically detect the target when the game starts and use it for storage.

Replay Memory Target

A replay memory target is able to store replay data in memory for playback during the same game session. The target is volatile meaning that all replay data is lost when the game session is closed. Memory targets are idea when you want to record short periods of gameplay for replay at a later stage, For example: an action replay in a sports game or a kill-cam in an action shooter game.

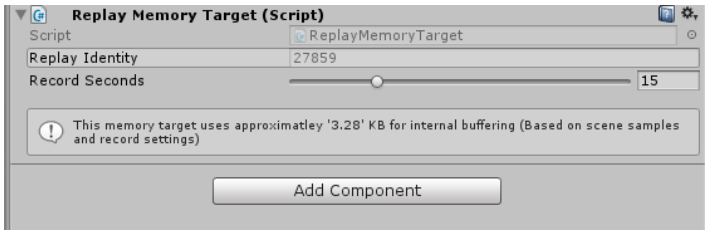

Figure 7

Replay memory targets use a rolling buffer by default which allows only the past 'X' seconds of gameplay to be recorded. You can adjust the slider to determine how many seconds of gameplay get recorded and you will also be able to see the average memory usage for the target in the information box.

- **Replay Identity:** A unique 16-bit identifier used by the replay system (Read only). This value is auto generated when the component is attached.
- **Record seconds:** The amount of time in seconds to keep buffered in memory. For example: when the default value of '15' is used, the past 15 seconds of gameplay will be stored in a rolling memory buffer. As the gameplay advances, old frames are disposed of to keep memory usage to a minimum. This is very useful when you know that you will only need to record the past X seconds of gameplay. By settings this value to '0' you will prevent the buffer from being clipped causing a continually expanding buffer to be used. You should take care with memory usage when using this approach.

Replay Object

A replay object is a vital component in the replay system as it determines which objects and components will have their data recorded. In order for any game object or prefab to be recorded, the replay object component should be added. You may also notice that a replay object is required by all replay components. When adding a replay component you may notice that a replay object component is also added to the object if there is not one already. This indicates the direct requirement of the replay object component.

Replay objects will auto register and unregister with the replay system when they are created or destroyed so you do not need to do it manually. If the component is attached to a prefab then you may notice a hint in the information box stating that the object can be used as a replay prefab if so desired. This may be useful if the object will be created or destroyed during recording. See the 'Replay Prefabs' section earlier in this document for more information.

Figure 8

A replay object has a few serialized variables which can be viewed via the inspector window:

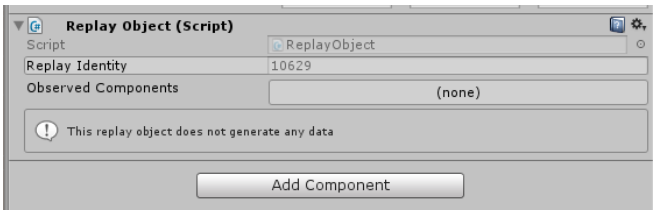

Figure 9

- **Replay Identity:** A unique 16-bit identity used to identify the replay object (Read Only). The identity is used to serialize the replay data for the object and ensures that all state data is dispatched to the correct object. This value is auto generated when the component is attached.
- **Observed Components:** An array of replay components deriving from 'ReplayBehaviour' that will be recorded by the replay object. The array will be auto filled as script components are added or removed from the game object or child objects. Any number of components should be observed but note that each component adds a certain amount of data overhead which must be serialized during recording. When components are added to the observed array, you will note that the information box will update accordingly to show approximately how much data will be generated per sample for the replay object.

Replay Behaviour

Replay behaviours are special scripts used by the replay system that are capable or recording and restoring replay data if necessary as well as begin able to receive a number of replay events such as 'OnReplayStart'. As a result all replay components must derive from replay behaviour and will be automatically detected by the managing replay object and marked for recording.

Replay behaviours also have the added benefit of surviving playback without being disabled by the replay system. This is because all replay components must be able to restore their data at any time so disabling the script may cause errors. If you have a game script that you do not want to be disabled during playback then you can make the script derive from replay behaviour (which in turn derives from mono behaviour) which will allow the script to survive. You may however not want the script to be marked for recording as there is a slight data overhead for even empty recorded replay behaviours.

Replay Ignore

You are able to specify that a script deriving from replay behaviour should not be recorded by using the 'ReplayIgnore' attribute on the class. This will cause the replay system to ignore the script and as a result it will not be observed by a replay object and it will not receive 'OnReplaySerialize' and 'OnReplayDeserialize' events during recording or playback. You will however still be able to use the various replay events. The following code shows how you can add the attribute to a custom replay component:

C# Code

```
1  using UnityEngine;
2  using UltimateReplay;
3
4  [ReplayIgnore]
5  public class ReplayExample : ReplayBehaviour
6  {
7      public override void OnReplaySerialize(ReplayState state)
8      {
9          // Abstract method must be implemented
10     }
11
12     public override void OnReplayDeserialize(ReplayState state)
13     {
14         // Abstract method must be implemented
15     }
}
```

Replay Variables

Ultimate Replay also allows you to mark any serialized field as a replay variable via an attribute which will automatically mark it for recording without you needing to do any custom data serialization. Serialized fields are defined as any public field or any private field with the 'SerializedField' attribute as per Unity. Many field types are supported for serialization by default. Take a look at the 'Replay State' section for information on supported types. The following code example shows how you can mark a field as a replay variable:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample : ReplayBehaviour
{
    [ReplayVar]
    public int recordedInt = 5;

    public override void OnReplaySerialize(ReplayState state)
    {
        base.OnReplaySerialize(state);
    }

    public override void OnReplayDeserialize(ReplayState state)
    {
        base.OnReplayDeserialize(state);
    }
}
```

As you can see, the variable is marked with the 'ReplayVar' attribute which will cause it to be automatically recorded. Another important thing to note is that the serialize and deserialize base methods are called. This is important as the base methods are responsible for handling both replay variables and replay events. If you do not call the base method then replay variables will simply be ignored.

Replay variables also support value interpolation meaning that primitive values that support interpolation such as float, int or Vector3 can be linearly interpolated during playback to achieve smoother transitions between values where low record rates are used. By default, variable interpolation is enabled provided that the variable type supports interpolation but you can disable interpolation by passing false as a parameter to the 'ReplayVar' attribute.

Replay variables can only be used in classes that derive from replay behaviour. If you use the attribute elsewhere, for example in a mono behaviour class then the variable will not be recorded.

Note: *Replay Variables need to be serialized with type information in order for correct deserialization during playback which causes additional Meta data to be stored causing increased storage sizes. For best optimization, you should use custom serialization to avoid this additional data from being stored.*

Replay Events

By deriving from replay behaviour, you will also be able to record replay events at any time which will be triggered at the correct time during playback. These events can be used to prompt certain behaviour during playback at a specific frame, for example: causing an audio effect to begin playing. In fact this is actually how the replay audio component is implemented and when the event is triggered during playback, the sound effect is played at the exact time it was recorded.

In order to record an event you simply need to call the method 'ReplayRecordEvent' on the replay behaviour base class. The method accepts an 8-byte id value to uniquely identity the event during playback along with a replay state containing any data that should be serialized along with the events. The following code shows how a replay event can be recorded:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample : ReplayBehaviour
{
    public void Update()
    {
        if(IsRecording == true)
            ReplayRecordEvent(5, null);
    }

    public override void OnReplaySerialize(ReplayState state)
    {
        base.OnReplaySerialize(state);
    }

    public override void OnReplayDeserialize(ReplayState state)
    {
        base.OnReplayDeserialize(state);
    }
}
```

As you can see, we first check whether we are currently recording because it only makes sense to record events if we are actually recording. We then record an event every update with an id of '5' and no data.

Note: *It is not recommended to record events in Update or Fixed Update. The record rate may be less that the call frequency which would cause multiple stacked events with identical data to be recorded in the same replay frame which is obviously very memory inefficient. The above code is used for demonstration*

Once you have recorded an event, the next thing you will want to do is listen for that event to occur during playback so you can act accordingly. The following code builds upon the above example and adds an event listener to handle the event:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample : ReplayBehaviour
{
    public void Update()
    {
        if(IsRecording == true)
            ReplayRecordEvent(5, null);
    }

    public override void OnReplayEvent(ReplayEvent evt)
    {
        switch(evt.eventID)
        {
            case 5: /* Handle event number '5' */ break;

            default:
                Debug.LogWarning("Unknown replay event triggered: "
+ evt.eventID);
                break;
        }
    }

    public override void OnReplaySerialize(ReplayState state)
    {
        base.OnReplaySerialize(state);
    }

    public override void OnReplayDeserialize(ReplayState state)
    {
        base.OnReplayDeserialize(state);
    }
}
```

As you can see, the 'OnReplayEvent' method is called for each replay event that occurs and you must identify the type of event via its event ID. Typically a switch statement will be the most convenient way to handle multiple events as shown above. You can then perform any replay code you want when the event is triggered.

For more information on replay events take a look at the source code for the replay audio component which can be found at 'Assets/UltimateReplay/Scripts/ReplayAudio.cs'.

Built-in Components

Ultimate Replay includes a number of built-in replay components that can be used to record specific data associated with a game object. You can easily create your own replay components in order to expand the functionality of Ultimate Replay or you could even request a replay component to be added if it does not exist. The built-in components are listed below:

- **ReplayTransform:** Responsible for recording a game objects transform component. The component allows you to record position, rotation and scale in any combination and fully supports interpolation for smooth playback at low frame rates. Local transform values are recorded meaning that you can record transform hierarchies for game objects with children.
- **ReplayAudio:** Allows an AudioSource to be recorded by the replay system. In order to record the audio, you will need to use the 'Play' method of the replay audio component instead of the audio source which will cause the audio to play in game and mark an audio event in the recording. Audio playback is not supported when a recording is being replayed in reverse mode. The audio pitch is adjusted to the replay speed to match the audio effect to the speed of the replay.

More replay components may be added in the future.

Replay State

A replay state is a lightweight storage container used throughout the Ultimate Replay system to serialize and deserialize data. You will likely never need to use the Replay State class unless you need to implement some advanced behaviour in order to record components that are not supported or similar. The Replay State is responsible for taking higher level types such as integers and Vector3's and converting them into a byte stream that can be easily serialized to the active replay target.

The following types can be serialized into a replay state by default using the appropriate write / read method.

- **Byte:** Write(byte), ReadByte()
- **Byte[]:** Write(byte[]), ReadBytes(int)
- **Short:** Write(short), Read16()
- **Int:** Write(int), Read32()
- **Float:** Write(float), ReadFloat()
- **Bool:** Write(bool), ReadBool()
- **String:** Write(string), ReadString()
- **ReplayIdentity:** Write(ReplayIdentity), ReadIdentity()
- **ReplayState:** Write(ReplayState), ReadState(int)
- **Vector2:** Write(Vector2), ReadVec2()
- **Vector3:** Write(Vector3), ReadVec3()
- **Vector4:** Write(Vector4), ReadVec4()
- **Quaternion:** Write(Quaternion), ReadQuat()
- **Color:** Write(Color), ReadColor()
- **Color32:** Write(Color32), ReadColor32()

Special write methods:

- **object:** TryWriteObject(object), TryReadObject()

The above methods will attempt to write a System.object to the replay state. These methods should be avoided where possible as they need to store type information which uses a large amount of bytes in order to read the object back during deserialization. If the type is known at compile time then these methods should not be used.

Advanced Topics

Most of the topics covered so far have been mainly basic and common tasks that you will usually perform in Ultimate Replay but there are some more advanced features which the more experienced user may want to use.

Custom Replay Components

Since ultimate Replay uses a state based serialization system you are able to record almost any type of data you want by creating your own replay component. This custom component could be used to record animation data or any other data you like which can be restored during playback. In order to create a replay component you will need to make use of the 'ReplayBehaviour' class. If you haven't yet read the section for replay behaviours you should take a look at it before continuing so that you understand how they work.

You will need to create a class that derivers from replay behaviour and as a result will need to implement its abstract members as shown below:

C# Code

```
1  using UnityEngine;
2  using UltimateReplay;
3
4  public class ReplayExample : ReplayBehaviour
5  {
6      public override void OnReplaySerialize(ReplayState state)
7      {
8          base.OnReplaySerialize(state);
9      }
10
11     public override void OnReplayDeserialize(ReplayState state)
12     {
13         base.OnReplayDeserialize(state);
14     }
}
```

You will note that there are 2 methods which must be implemented, OnReplaySerialize and OnReplayDeserialize. The serialize method is called during recording when a replay component should be recorded and the deserialize method is called during playback when a replay component should be restored. Each method accepts a single argument of a 'ReplyState' which is what you will use to write or read any data to and from. Take a look at the 'ReplayState' section to find out what types cane be serialized automatically. You will also note that the base methods are called in our example. The base method will serialize all replay events and replay variables that are associated with the behaviour so if you know that you will not be using any replay variables or replay events in your custom components then you can safely remove the calls to the base method and replace it with your own functionality.

You will now need to decide what data you want to record and how it should be stored. This will be entirely dependent upon what you are trying to record. For example: if you want to record a character animation, you may want to serialize animation key frame times along with animation names. It may be worth looking at the source code for the built in components at this time such as

'ReplayTransform' so you can see how they are implemented to get a better understanding of how the replay system works.

When you have your component recording and restoring data correctly you may want to further improve it by adding support for interpolation. Again this will depend upon the type of data you are recording as interpolation may not be needed or may even cause undesirable behaviour. If you do decide to add interpolation then a common method is to store information about the previous replay frame (during deserialization) so that you have a previous and current replay snapshot. You can then interpolate between these 2 frames in an update method. Usually 'OnReplayUpdate' will be suitable for this purpose. The following code example shows a common interpolation pattern used to smoothly transition between 2 values:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample : ReplayBehaviour
{
    private float lastValue;
    private float currentValue;

    public override void OnReplaySerialize(ReplayState state)
    {
    }

    public override void OnReplayDeserialize(ReplayState state)
    {
        // Store the current value as the last value
        lastValue = currentValue;

        // Get the current value
        currentValue = state.ReadFloat();
    }

    public override void OnReplayUpdate()
    {
        // Interpolate our value
        float interpolatedValue = Mathf.Lerp(lastValue,
currentValue, ReplayTime.Delta);

        // Use the interpolated value as required
        //??? = interpolatedValue;
    }
}
```

Pooling Support

Ultimate Replay support instantiating and destroying objects during recording however if this is something you will be doing often then you may want to add pooling support to achieve better performance. Ultimate Replay does not include any form of pooling support but it does make it easy to implement on your own by simply subscribing to 2 replay events. These events are:

- **OnReplayInstantiate:** Called by the replay system when a prefab needs to be instantiated during playback in order to maintain scene state.
- **OnReplayDestroy:** Called by the replay system when a game object should be destroyed during playback in order to maintain scene state.

These replay events allow you to implement your own instantiate or destroy code which allows you to easily support pooling. The events are implements as static C# events which you can subscribe to as shown below:

C# Code

```
1  using UnityEngine;
2  using UltimateReplay;
3
4  public class ReplayExample : Monobehaviour
5  {
6      void Start()
7      {
8          // Subscribe to replay events
9          ReplayManager.OnReplayInstantiate += InstantiateHandler;
10         ReplayManager.OnReplayDestroy += DestroyHandler;
11     }
12
13     void InstantiateHandler(GameObject prefab, Vector3 pos,
14 Quaternion rot)
15     {
16         // Add custom instantiate code here
17     }
18
19     void DestroyHandler(GameObject go)
20     {
21         // Add custom destroy code here
22     }
23 }
```

When you add a listener to these events you will override the default instantiate of destroy method used by Ultimate Replay.

Custom Object Preparers

Replay preparers are used by the replay system to make a game object ready for playback mode or live mode. In order for an object to be replayed properly without having game systems such as physics or animation affect them they need to be prepared. This preparation stage usually involves disabling, modifying or in some cases destroying the components that are attached to the replay object. This allows the object to be frozen in the scene and manipulated as required by the replay system to produce an accurate playback sequence.

By default, ultimate Replay uses a preparer that disables colliders and scripts and makes any rigid body components kinematic. If this is undesirable then you are able to implement your own preparer that will be used when switching between live mode and playback mode. You can then handle the object preparation in any way you like, but bear in mind that failing to disable certain components could cause the playback to be inaccurate or even glitch if colliders are involved.

In order to create your own preparer you will need to implement the 'IReplayPreparer' interface which is located under the 'UltimateReplay.Core' namespace:

C# Code

```
1  using UnityEngine;
2  using UltimateReplay.Core;
3
4  public class ReplayPreparerExample : IReplayPreparer
5  {
6      public void PrepareForPlayback(ReplayObject obj)
7      {
8          // Custom preparation code here
9      }
10
11     public void PrepareForGameplay(ReplayObject obj)
12     {
13         // Custom preparation code here
14     }
}
```

You can take a look at the default implementation in order to see how the preparer works which can be found at the following location: 'Assets/UltimateReplay/Scripts/Core/DefaultReplayPreparer.cs'.

Once you have completed the implementation of your custom preparer you will then need to register it when the game starts so that the replay system will know to use it instead of the default preparer. This can be done by simply assigning an instance of your custom preparer class to the global preparer property of the replay manager:

C# Code

```
using UnityEngine;
using UltimateReplay;

public class ReplayExample : MonoBehaviour
{
    void Start()
    {
        // Assign our custom preparer
        ReplayManager.Preparer = new ReplayPreparerExample();
    }
}
```

You should now have a working custom replay preparer which will be called when required by the replay system.

Experimental

At the time of release we have already began adding additional features that we expect will be highly anticipated and will include these features in a future update. Some of these experimental features are:

- **File support:** Support for streaming to and from file which would allow recordings to be persistent as well as compressed to reduce the storage requirement. This will also allow for much larger recordings to be captured.
- **Particles support:** Support for recording and replaying the Unity particle system. This will allow particles effects such as explosions or similar to be recorded.
- **Animation support:** Since a lot of games rely on animation of various characters and props, it would make sense to add recording support for the animator component so that these animations could be accurately replayed.

Let us know if you want to see any more features. See the last page of this document for information on requesting a feature to be added.

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, this is a pathfinding asset and requested features should fall under this category. We will make no attempt to add features that are off topic such as enemy AI behaviours.

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/