Frame Recorder Design Overview & notes

# Brief

The Frame Recorder is a service that facilitates recording of Unity play time artifacts. The framework does not define what *can* be recorded, but does propose a standard way of how to record these artifacts and also attempts at taking care of aspects common to all recorders.

An important aspect is extendibility, where recorders are detected at run time and made available to the recording framework dynamically. This is accomplished through inheritance and class attributes.

Another key consideration is UX, by defining a standard pattern and basic classes, lets the service treat all recorders equally and display them consistently. This allows for a global “recorder window” that is part of the service and allows starting a “recording session” from edit mode. Two main considerations are a global Recorder Window and universal timeline UI integration.

Code reusability and easy of use for developers is also a prime consideration. As much as possible, modularization in a Lego mentality is promoted so that work done for one specific recorder, say MP4 recording, can be reused by an other type of recorder, say PNG or WAV recorders.

Of note is the control of flow of time: is variable frame rate requested or fixed? This impacts all recorders that record more than one frame’s worth of data and so is taken care of by the framework.

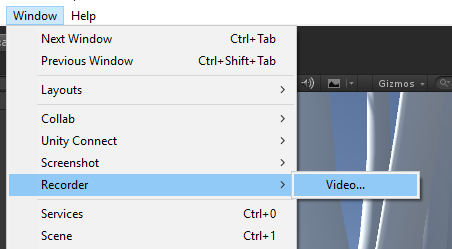
And last but not least, is the ensuring that at any point a custom recorder can override any standard behaviour of the service to allow full flexibility and not constrain what can be recorded or how.

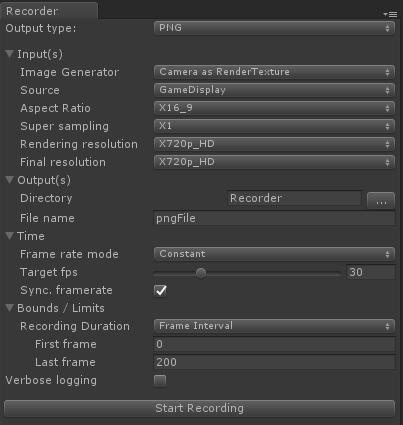
# Current obvious limitations

* Recorders are Player standalone friendly, but not the editors.
* Framerate is set at the Recorder level which makes for potential conflict when multiple recorders are active simultaneously.

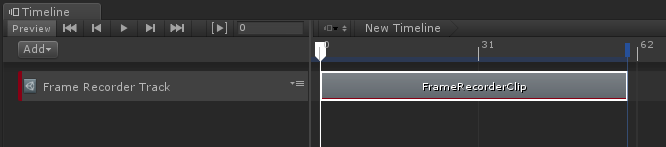
# Samples

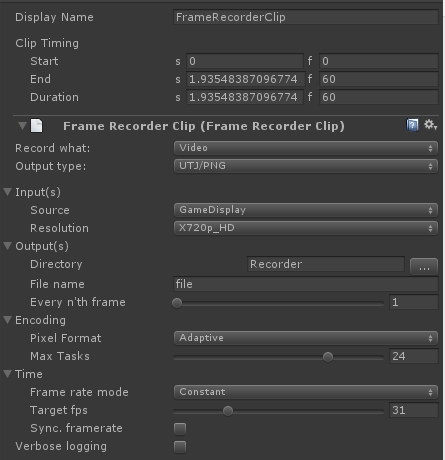
## The standalone Recording window





## A timeline recorder clip





# The “Core” classes



#### The engine classes:

* *Recorder*
  + Base/abstract class of all “recorders”. A recorder being the class that consumes the artifacts coming from the Unity Game and is responsible for transforming/encoding and “storing” them into the final output of a recording event. Examples would be: MP4, WAV, Alembic, Animation clips.
  + Instances of this class are temporary and live only for the duration of the actual recording. The data they record comes, normally, from *RecorderInput* objects (see below).
  + A recorder can have [0,n] inputs and is responsible for mixing/transcoding/processing the data coming from its inputs. In the context of audio recording, sources can be seen as audio tracks, where each track is a separate audio source.
  + Recorders get notified of when to start recording, when a new frame is being prepared, when a frame is ready to be recorded (at which point the recorder can read the frame’s data from it’s inputs) and when the recording ends.
  + Recorders are not responsible for handling/controlling time. That is the responsibility of the underlying recording service.
  + Recorders are responsible for creating their *inputs* and do so based on the specialized *RecorderSettings* objects that is passed to it on instantiation.
  + Recorders register themselves with the recorder framework by decorating a static method with the attribute *FrameRecorderRegister*. This is also how the framework determines what settings class is used by a given recorder class.
* *RecorderSettings*
  + Each type of recorder can specializes this class to add it’s own custom settings.
  + This base class comes with universal setting fields that apply to all recorders.
  + Instances of this class do persist in the Scene (the scene contains a auto-created GO that holds all the Recorder’s framework state data).
* *RecorderSource*
  + This is the base class for all data source that recorders use to gather/access the Unity game time artifacts to record.
  + The source classes have NO dependency on the recorders or their settings classes. This is important so that they can be reused between recorders.
  + A family of Sources is defined by their input type and output type and that is what determines if a recorder can use a source or not. (If a recorder can only record a RenderTexture but a given source outputs a packed pixel buffer/array, the recorder can’t use it)
  + Examples: RenderTextureSource, CameraSource, DisplaySource, GeometrySource, AudioSource.
  + Sources are responsible for gathering the data that the recorders end up recording. For example, say we have a recorder that wants to record Display 1 as a *RenderTexture*:
    - The renderer needs to select a source that takes as input a display ID and outputs a *RenderTexture*.
    - The source in question is responsible for figuring out how to do that.
  + Sources do not, by default, get Lifecyle event notifications like *OnUpdate*, *LateUpdate*, etc. It is the responsibility of the recorders to arrange them.
  + The system does NOT enforce a recorder to use a *RecorderSource*. It’s just the *standard* way of doing things.
* *RecordingSession*
  + This is a helper class that is used to hold and carry around the contextual state of a recording session.
  + Is the time reference used by the recorders: Recorders should, as a general rule, never access the unity Time class.
  + Contains:
    - Owns the recorder instance
    - Reference to the *GameoObject* that hosts the recorder
    - A list of the unity objects that the recorder is interested in. (cameras, game obj, etc.)
    - Current frame count (rendered, not recorded)
    - Current frame’s start time stamp relative to the start of the recording session.
    - Timestamp of when the recording session started.

#### The editor classes:

* *RecorderSettingsEditor*
  + Each recorder settings class must have an associated specialization of this class for the framework to be able to display and edit the settings.
  + The recorder settings editors get embedded inside the *RecorderWindow* and the *RecorderClipEditor* and are also the inspectors for the record settings types.
  + The settings are displayed in five main groups and each get a separate virtual handler method:
    - *OnInputGui*: input related settings, like display id, camera selection, etc.
    - *OnOutputGui*: output related settings, like file name, ftp address, etc.
    - *OnEncodingGui*: encoding specific parameters like bit rate, width/height of image, etc.
    - *OnTimeGui*: how is time processed. Fixe frame rate? Variable? Etc.
    - *OnBounds*: this only appears in the Recorder Window and indicates when to start and stop the recording. (from frame x to frame y, from t0 to t1, 1 frame)
* *RecorderWindow*:
  + This window can be displayed in edit mode to trigger a recording session.
  + Is invoked through the “Window” menu of Unity.
  + It lists the available recorders by category/capabilities.
  + Once a specific recorder has been selected, the associated recorder settings editor is embedded inside this window to let user select the particulars of the recorder.
  + Enters play mode automatically when “record” button is pressed and will exit play mode when record bounds (see recorder settings) are met.
  + Since this window can be visible in edit mode, the settings that are stored on a game object that lives in the scene. Which means that the settings are remembered per Scene IF the scene is saved.

#### Timeline integration:

* *RecorderClipEditor*
  + This is similar to the *RecorderWindow* only for timeline: when a recorder clip is added to a track, this was what is used to edit the clip.
  + The editor allows user to select a recorder from all the available recorders and edit that recorders settings by embedding the specialized *RecorderSettingsEditor* inside itself.
* *RecorderClip*
  + Timeline clip class that holds a recording session and its settings.
  + The settings actually live in the scene…
* *RecorderPlayable*
  + The playable class that drives the recorder.

#### The support classes:

* *RecorderComponent*
  + Used when *RecorderWindow* triggers a recording.
  + Might be used if a recorder is manually added at runtime, through scripts.
  + Is responsible for creating the recorder instance and owns the lifetime of the recorder objects.
  + Pipes the recorder instance with lifecycle events: *OnUpdate*, etc.
  + Based on the settings object, determines when to stop the recording.
* *RecorderTimelineComponent*
  + Similar to the RecorderComponent bit when the recording is started from a timeline.
  + Looking to get rid of this class and move its logic to the *RecorderPlayable* class.
* RecorderFactory:
  + Since recorders are discovered at runtime by the framework, this class is used to create them.
  + Each recorder is responsible for registering itself with this class (using a *FrameRecorderRegister* attribute) and also informing what class to use for the settings of the recorder.
  + Responsible for instantiating Settings objects and registering them with the current Scene’s *GameObject* that holds them.

# Example MP4 recorder

This is an example use case of a hypothetical MP4 recorder that takes an image feed and a sound feed (inputs).

* *Frame Recorder* package are Unity provided classes.
* *AdamBeautyInput* is an input that “binds” to a camera and generates a *RenderTexture* that is consumed by the MP4Recorder class. Which camera that is bound to is determined by *AdamBeautyInputSettings*.
* CameraAudioSource is an input that “binds” to a camera and generates an Audio byte stream that is consumed by the MP4Recorder class. Which camera that is bound to is determined by *AdamBeautyInputSettings*.
* BaseImageRecroder is a partial recorder implementation that deals with recording Rendertextures.
* *MP4Recorder* is a custom class that takes the audio and video inputs and encodes them in the MP4.
* *MP4RecorderSettings* recorder specific settings, for example: output path, encoding options, etc. Does not Include settings for the Inputs. But can select which Input’s to use.