U-EAT

Documentation

Table of Contents

Events

What are Events?

In short, events are an abstract way for GameObjects to communicate with one another. GameObject 'A' can call a function on GameObject 'B' without either object necessarily needing to know about one-another.

How do they work?

In order to recieve an event, a GameObject must first connect via the EventSystem either using the static 'EventConnect' function or the extension method 'Connect':

```
public static void EventConnect(GameObject target, string eventName, Action<EventData
> func);

//Extension Method

public static void Connect(this GameObject target, string eventName, Action<EventData
> func);
```

target: The object that is expected to recieve the event.

eventName: The name of the event that is being listened for.

func: A function that returns nothing but takes in EventData as its first parameter.

Next, the event must be sent to the listening target using either the static 'EventSend' function or the 'DispatchEvent' extension method.

```
public static void EventSend(GameObject target, string eventName, EventData eventData
= null);

//Extension Method

public static void DispatchEvent(this GameObject target, string eventName, EventData
eventData = null);
```

eventData: An optional paramater which is used to pass data through the event. The user should store the data in a class which inherits from 'EventData'. By default, an empty EventData class is used.

Example

```
public class CustomEventData : EventData

{

    public int StoredInt;

    public CustomEventData(int val)

    {

        StoredInt = val;

    }

}


public class EventExample : MonoBehaviour

{
    void Start()
    {
        EventSystem.EventConnect(this.gameObject, "HelloEvent", SayHello);
        EventSystem.EventSend(this.gameObject, "HelloEvent", new CustomEventData(5));
    }

    void SayHello(EventData data)
    {
        CustomEventData customData = (CustomEventData)data;
        Debug.Log("Hello World");
        Debug.Log(customData.StoredInt);
    }
}
```

Output

```
Hello World
5
```

The 'Events' Class

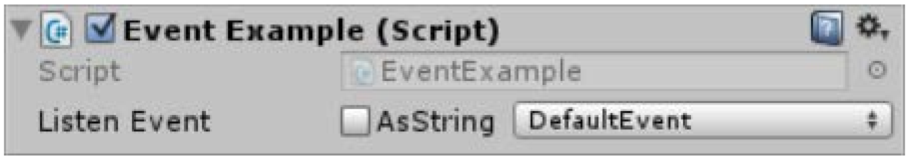
The 'Events' class is used to create a visual interface for strings to be used as events. Adding another public static readonly string to the class will add another event to the dropdown menu of events.

```
public class Events
{
    //All of the public static readonly strings in this class will appear in the Even
ts insepector's dropdown menu.
    //Leave DefaultEvent as the first event defined in this class.
    public static readonly String DefaultEvent = "DefaultEvent";
    public static readonly String KeyboardEvent = "KeyboardEvent";
    public static readonly String MouseUp = "MouseUp";
    public static readonly String MouseDown = "MouseDown";
    public static readonly String MouseEnter = "MouseEnter";
    public static readonly String MouseExit = "MouseExit";
...
```

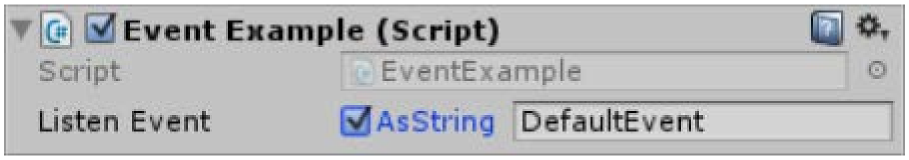
Creating a public member variable of type 'Events' in a monobehavior will make it be drawn in the inspector. The events class can be implicitly converted from (or to) a string.

```
public Events ListenEvent = Events.DefaultEvent;
```

Leads to:



Checking 'AsString' allows a custom string to be entered:



Actions

What are Actions?

Actions make it simple to set up groups or sequences of object interpolations. For example: Changing a color from red to blue over a specified duration.

How do they work?

All Actions inherit from the 'ActionBase' class, which allows them to be paused, resumed, restarted, and updated with a given Delta Time. The static 'Action' class is used to interface with the system.

The primary class for object interpolation is the 'ActionProperty' which can be created using the following static functions in the 'Action' class.

Note: The entire Action System is in the 'ActionSystem' namespace.

```
//Sequence Version

public static ActionProperty<T> Property<T>(ActionSequence seq, Property<T> startVal,
T endVal, double duration, Curve ease);

//Group Version

public static ActionProperty<T> Property<T>(ActionGroup grp, Property<T> startVal, T
endVal, double duration, Curve ease);
```

seq: The 'ActionSequence' for the property to be part of. Sequences are updated in first-in first-out order. Sequences and groups can even be in a sequence.

grp: The 'ActionGroup' for the property to be part of. Groups are updated all at the same time. Sequences and groups can even be in a group.

startVal: A 'Property' wrapper around the getter and setter functions of the value to be interpolated. Properties are created as follows:

```
//The property to interpolate must be public, gettable, and settable.

public float InterpolatedValue { get; set; }


//Inside of a function, the extension method 'GetProperty' can be called. This functi
on takes in a lambda of the variable to be interpolated.

Property<float> floatProp = this.GetProperty(val => val.InterpolatedValue);
```

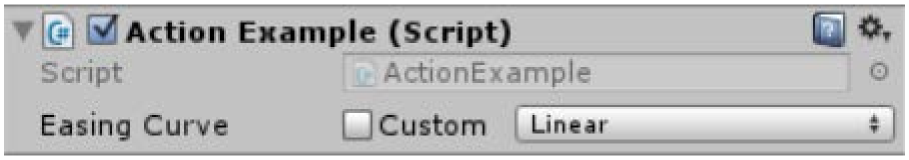
endVal: The value that the property is interpolating towards.

duration: The time in seconds of the interpolation.

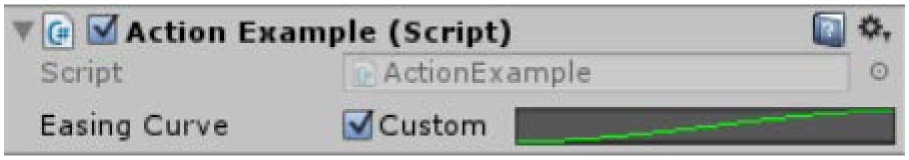
ease: The type of easing curve that the interpolation should use. All the eases listed here (A link to Gizma Easeing)are implemented. A 'Curve' is can be implicitly converted from either the 'Ease' enum or a Unity 'AnimationCurve'. Creating a public class variable of a 'Curve' like this:

```
public Curve EasingCurve = Ease.Linear;
```

Will show up in the inspector like so:



Checking 'Custom' allows a custom AnimationCurve to be used:



After the sequence or group is filled with the desired actions, the Action must have its Update function called every frame, with the desired Delta Time passed in.

Example

```
using UnityEngine;
using ActionSystem;

public class ActionExample : MonoBehaviour
{
    public Curve EasingCurve = Ease.QuarticIn;
    public float InterpolatedValue { get; set; }
    ActionGroup Grp = new ActionGroup();

    void Start ()
    {
        //Create an ActionSequence inside of the ActionGroup.
        ActionSequence seq = Action.Sequence(Grp);
        //Add an ActionProperty to the sequence which will interpolate from 0 to 5 ov
er 3 seconds.
```

```
        Action.Property(seq, this.GetProperty(val => val.InterpolatedValue), 5, 3, Ea
singCurve);

        Reset(seq);

    }


    void Reset(ActionSequence seq)

    {

        //Then, interpolate this objects position to the point [5, 3, 2] over 2 secon
ds.

        Action.Property(seq, transform.GetProperty(val => val.position), new Vector3(
5, 3, 2), 2, Ease.QuadInOut);

        //Then, interpolate this objects position to the point [-5, 3, 2] over 2 seco
nds.

        Action.Property(seq, transform.GetProperty(val => val.position), new Vector3(
-5, 3, 2), 2, Ease.QuadInOut);

        //Then call this function again with another sequence, causing a loop.

        Action.Call(seq, Reset, Action.Sequence(Grp));

    }


    void Update ()

    {

        Grp.Update(Time.smoothDeltaTime);

    }

}
```

Note: Instead of creating a new ActionGroup in every script that uses Actions, it is recommended that the extension method 'GetActions' be called inside of the MonoBehaviour.

```
ActionGroup grp = this.GetActions();
```

This function will get and return the ActionGroup inside of the 'ObjectActions' component. If there is no 'ObjectActions' component on the GameObject, the function will add an invisible one. Using this component will mean that all actions on the object will be updated automatically.

Output

First, the InterpolatedValue will interpolate to 5 over 3 seconds, then the object will interpolate back and for indefinitely.

Types of Actions

ActionProperty: Interpolates an object from one value to another. In order to be interpolated a type must be able to be added or subtracted from itself, and be multiplied and divided by either floats or doubles.

ActionSequence: A sequence of other Actions that happen one after the other. A looping sequence will repeat once the final Action has completed.

ActionGroup: A group of other Actions that happen all at the same time. A looping group will repeat once the longest Action has completed.

ActionDelay: Simply delays the sequence for the specified number of seconds.

ActionCall: Will call the given function that takes up to 4 paramaters.

ActionReturnCall: Will call the given function that takes up to 4 paramaters and stores the return type.

Types of Eases

All the mathematical eases on the following website have been implemented: http://gizma.com/easing/