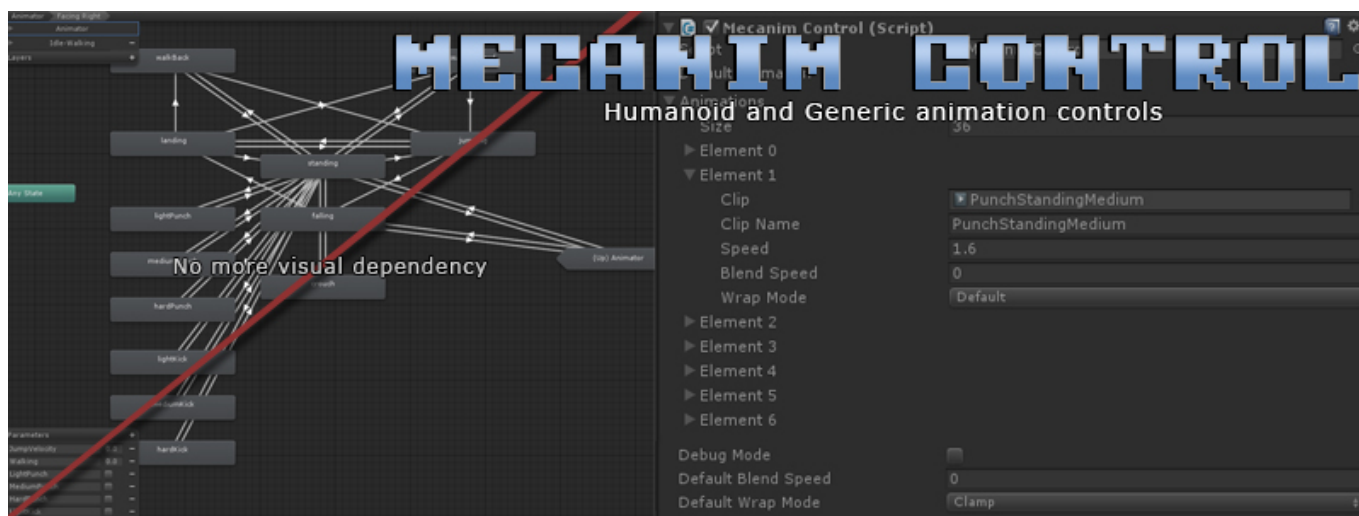


Mecanim Control

Your ultimate solution for Mecanim based games!



Mecanim Control is a coding tool made that allow for a wider variety of common methods used by the [Animation component](#) with Mecanim (Humanoid/Generic) animations. It allows you to not only dynamically load any animation clip during runtime, but also tap into several methods currently missing in this magnificent system.

[Mecanim Control](#) is a sub-tool of [Universal Fighting Engine](#). It's source code is available entirely free in the **Source** version of UFE.

Overview

You can use MecanimControl much like you would use the animation component.

To play a simple animation use *MecanimControl.Play*

To cross-fade between animations use *MecanimControl.CrossFade* -or- one of the *MecanimControl.Play* alternatives.

To change how animations wrap (Loop, Once, PingPong) change the WrapMode of the respective *AnimationClip* in their import settings, or use *MecanimControl.SetWrapMode* to change it at runtime. *AnimationData* can be used to modify the clip, playback speed, and direct control over blending.

MecanimControl also supports enumerators so you can loop through all AnimationData like this:

```
using UnityEngine;
using System.Collections;

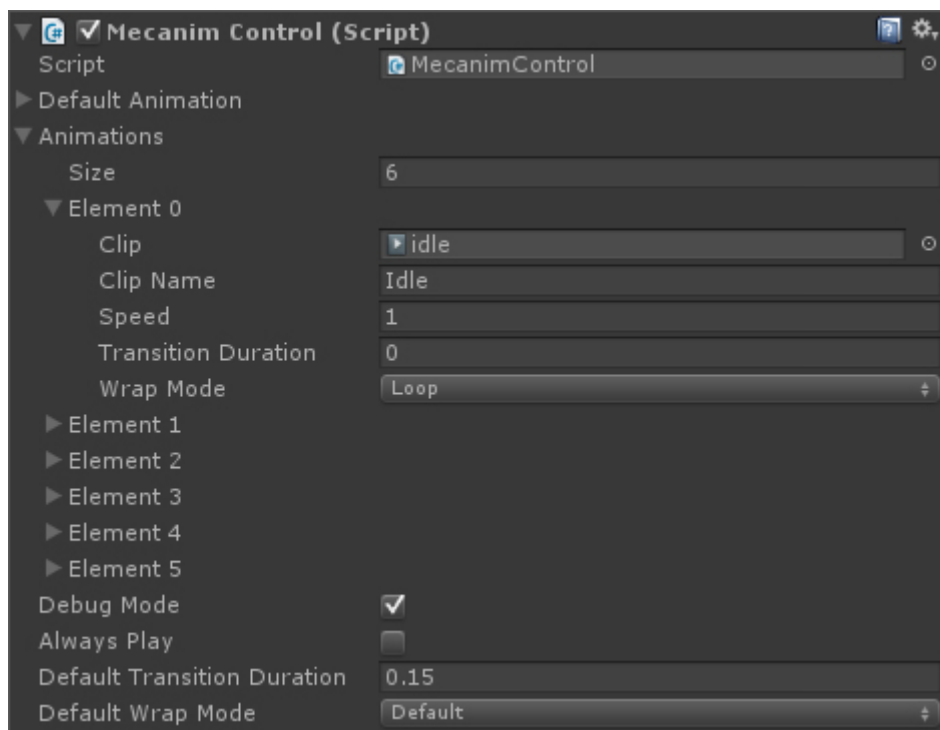
public class AnimationControlDemo : MonoBehaviour {

    private MecanimControl mecanimControl;

    void Start () {
        mecanimControl = gameObject.GetComponent<MecanimControl>();
    }
}
```

```
}  
  
void OnGUI(){  
    foreach(AnimationData animationData in mecanimControl.animations){  
        if (GUILayout.Button(animationData.clipName)){  
            mecanimControl.Play(animationData, mirror);  
        }  
    }  
}  
}
```

Public Variables



Index:

- [Default Animation](#)
- [Animations](#)
- [Debug Mode](#)
- [Always Play](#)
- [Default Transition Duration](#)
- [Default Wrap Mode](#)

[defaultanimation](#) **Default Animation**

AnimationData *defaultAnimation*;

By default, if no order is given, the animator will play the animation stored in this AnimationData.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.defaultAnimationData.speed = .5f;  
}
```

animations Animations

AnimationData[] *animations*;

The array containing all the AnimationData stored by either the UI or by using AddClip.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    foreach(AnimationData animationData in mecanimControl.animations) {  
        animationData.speed = .5f;  
    }  
}
```

debugmode Debug Mode

bool *debugMode*;

Toggles a GUI box containing all the information about the current clip playing as well as its blending weight.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.debugMode = true;  
}
```

alwaysplay Always Play

bool *alwaysPlay*;

If an animation is set to *WrapMode.Once* and *alwaysPlay* is toggled on, after the clip ends it will immediately play the *default animation*.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.alwaysPlay = true;  
}
```

defaulttransitionduration Default Transition Duration

float *defaultTransitionDuration*;

If an animation has its blending speed set to 0, it will use this value instead.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.defaultTransitionDuration = .2f;  
}
```

```
}
```

defaultwrapmode **Default Wrap Mode**

float *defaultWrapMode*;

If an animation has its *wrapmode* set to *default*, it will use this value instead.

```
void Start () {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.defaultWrapMode = WrapMode.Once;  
}
```

Public Functions

```
void OnGUI() {  
    if (GUILayout.Button("Invert Speed")) mecanimControl.SetSpeed(-mecanimControl.GetSpeed());  
    if (GUILayout.Button("Mirror "+ mirror)) {  
        mirror = !mirror;  
        mecanimControl.SetMirror(mirror);  
    }  
  
    GUILayout.Space(10);  
  
    foreach(AnimationData animationData in mecanimControl.animations){  
        if (GUILayout.Button(animationData.clipName)){  
            mecanimControl.Play(animationData, mirror);  
        }  
    }  
}
```

Index:

- [AddClip](#)
- [CrossFade](#)
- [GetAnimationData](#)
- [GetCurrentAnimationData](#)
- [GetCurrentClipName](#)
- [GetCurrentClipPlayCount](#)
- [GetCurrentClipPosition](#)
- [GetMirror](#)
- [GetSpeed](#)
- [IsPlaying](#)
- [Pause](#)
- [Play](#)
- [Remove Clip](#)
- [Restore Speed](#)
- [Rewind](#)
- [SetCurrentClipPosition](#)
- [SetDefaultClip](#)
- [SetMirror](#)

- [SetSpeed](#)
 - [SetWrapMode](#)
 - [Stop](#)
-

[addclip](#) **AddClip**

`void AddClip(AnimationClip clip, string name);`

`void AddClip(AnimationClip clip, string name, float speed, WrapMode wrapMode);`

Parameters

clip - The AnimationClip file.

name - Animation name.

speed - Animation speed.

wrapMode - The animation's default [WrapMode](#).

Description: Adds a clip to *animations* with the name *newName*.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public AnimationClip walkClip;
    void Start () {
        mecanimControl = gameObject.GetComponent<MecanimControl>();
        mecanimControl.AddClip(walkClip, "walk");
    }
}
```

[crossfade](#) **CrossFade**

`void CrossFade(string clipName, float blendingTime);`

`void CrossFade(string clipName, float blendingTime, float normalizedTime, bool mirror);`

`void CrossFade(AnimationData animationData, float blendingTime, float normalizedTime, bool mirror);`

Parameters

clipName - Animation name.

animationData - The correspondent animation data.

blendingTime - The blending duration between the 2 animations.

normalizedTime - The timeline's position of the animation to be played (0-1)

mirror - Should the animation be mirrored?

Description: Fades the animation with name *clipName* in over a period of *blendingTime* seconds as it fades other animations out.

You can also set *normalizedTime* to set where, in its timeline, you want the animation to start (0-1) as well as toggle [mirror](#).

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
```

```
public AnimationClip walkClip;
void Start () {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    mecanimControl.CrossFade("walk", .2f);
}
}
```

[getanimationdata](#) **GetAnimationData**

AnimationData *GetAnimationData*(AnimationClip clip);

AnimationData *GetAnimationData*(string clipName);

Parameters

clip - Animation clip.

clipName - Clip name.

Description: Returns the AnimationData related to that animation name or clip.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public AnimationClip walkClip;
    void Start () {
        mecanimControl = gameObject.GetComponent<MecanimControl>();
        mecanimControl.AddClip(walkClip, "walk");
        Debug("Animation Name:" +
mecanimControl.GetAnimationData(walkClip).clipName);
    }
}
```

[getcurrentanimationdata](#) **GetCurrentAnimationData**

AnimationData *GetCurrentAnimationData*();

Description: Get the AnimationData currently running.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public AnimationClip walkClip;
    void Start () {
        mecanimControl = gameObject.GetComponent<MecanimControl>();
        Debug("Animation Name:" +
mecanimControl.GetCurrentAnimationData().clipName);
    }
}
```

getcurrentclipname **GetCurrentClipName**

string *GetCurrentClipName()*;

Description: Get the name of the current running clip.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Start () {
        mecanimControl = gameObject.GetComponent<MecanimControl>();
        Debug("Animation Name:" + mecanimControl.GetCurrentClipName());
    }
}
```

getcurrentclipposition **GetCurrentClipPosition**

float *GetCurrentClipPosition()*;

Description: Get the *normalized time* of the current running clip. (0-1)

```
void CheckProgress() {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    Debug("Animation Progress (%):" + mecanimControl.GetCurrentClipPosition()
* 100);
}
```

getcurrentclipplaycount **GetCurrentClipPlayCount**

int *GetCurrentClipPlayCount()*;

Description: Get the number of times the current clip has played. Only works if the animation's *WrapMode* is set to either *WrapMode.Loop* or *WrapMode.PingPong*

```
void CheckProgress() {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    Debug("Times Played:" + mecanimControl.GetCurrentClipPlayCount());
}
```

getmirror **GetMirror**

bool *GetMirror()*;

Description: Get the current *mirror* state of the *emulated runtime animator*.

```
void FaceLeft () {
```

```
mecanimControl = gameObject.GetComponent<MecanimControl>();  
if (!mecanimControl.GetMirror()) mecanimControl.setMirror(true);  
}
```

getspeed **GetSpeed**

float *GetSpeed*();
float *GetSpeed*(AnimationClip clip);
float *GetSpeed*(string clipName);

Parameters

clip - Animation clip.
clipName - Clip name.

Description: Get the speed value set for *animationClip/clipName*.

no parameters - Get the speed the animator is running based on the current running animation.

```
void SlowDown() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    if (mecanimControl.GetSpeed() > 1) mecanimControl.SetSpeed(1);  
}
```

isplaying **IsPlaying**

bool *IsPlaying*(string clipName);
bool *IsPlaying*(AnimationClip clip);
bool *IsPlaying*(AnimationData animationData);

Description: Returns true if *clipName*, *clip* or *animationData* is playing.

```
void Example() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    if (mecanimControl.IsPlaying("walk")) Debug.Log("Walk is playing");  
}
```

pause **Pause**

void *Pause*();

Description: Pauses the animator component.

```
void Example() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.Pause();  
}
```

play Play

```
void Play();  
void Play(string clipName);  
void Play(AnimationClip clip);  
void Play(AnimationData animationData);  
void Play(string clipName, bool mirror);  
void Play(AnimationClip clip, bool mirror);  
void Play(AnimationData animationData, bool mirror);  
void Play(string clipName, float blendingTime, float normalizedTime, bool mirror);  
void Play(AnimationClip clip, float blendingTime, float normalizedTime, bool mirror);
```

Parameters

clip - Animation clip.

clipName - Animation name.

animationData - The correspondent animation data.

blendingTime - The blending duration between the 2 animations.

normalizedTime - The timeline's position of the animation to be played (0-1)

mirror - Should the animation be mirrored?

Description: Plays animation. *Play* can be used in several ways, including blending. If no blending is set, *Play* will try using the default blending value. If blending is set to -1, the animation will be played abruptly without any blending.

If the animation is not set to be looping and *alwaysPlay* is toggled off it will be stopped after playing.

If no parameters are used, *Play* can be used as a follow up to *Pause*. It restores the speed of the Animator to the current animation speed value.

Normalized Time lets you start the animation from a predefined position in the animation timeline (0-1).

```
void Example() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.Play();  
}
```

```
void Example() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.Play("walk", .2f, 0, true);  
}
```

removeclip RemoveClip

```
void RemoveClip(string clipName);  
void RemoveClip(AnimationClip clip);
```

Description: Removes the *AnimationData* from *animations* related to *clipName/clip*.

```
void RemoveAnimation(string animation) {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.RemoveClip(animation);  
}
```

```
}
```

[restorespeed](#) **RestoreSpeed**

`void RestoreSpeed();`

Description: Restores the speed of the animator component to the original value from the current animation being played.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

    private MecanimControl mecanimControl;

    void SlowMo(string animation) {
        mecanimControl.SetSpeed(.01f);
        Invoke("Restore", 2);
    }

    void Restore() {
        mecanimControl.RestoreSpeed();
    }
}
```

[rewind](#) **Rewind**

`void Rewind();`

Description: Inverts the speed of the animator component.

```
void Example() {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    mecanimControl.Rewind();
}
```

[setcurrentclipposition](#) **SetCurrentClipPosition**

`void SetCurrentClipPosition(float normalizedTime);`

`void SetCurrentClipPosition(float normalizedTime, bool pause);`

Description: Set the position in the timeline of the current playing clip (0-1). If pause is toggled on, the animation will be paused afterwards.

```
void Example() {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    mecanimControl.SetCurrentClipPosition(.3f, true);
}
```

```
}
```

setdefaultclip SetDefaultClip

`void SetDefaultClip(AnimationClip clip, string name, float speed, WrapMode wrapMode);`

Description: Sets the *defaultclip* through code (instead of the UI).

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {

    private MecanimControl mecanimControl;
    private AnimationClip idle;

    void Start() {
        mecanimControl.SetDefaultClip(idle, "Idle", 1, WrapMode.Loop);
    }
}
```

setmirror SetMirror

`void SetMirror(bool mirror);`

`void SetMirror(bool mirror, float blendingTime);`

`void SetMirror(bool mirror, float blendingTime, bool forceMirror);`

Description: When toggled on, every animation will be played with the *mirror* tag toggled on.

```
void FaceLeft () {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    if (!mecanimControl.GetMirror()) mecanimControl.setMirror(true);
}
```

setspeed SetSpeed

`void SetSpeed(float speed);`

`void SetSpeed(string clipName, float speed);`

`void SetSpeed(AnimationClip clip, float speed);`

Description: Change the speed value of the Animator component or AnimationData based on *clipName/clip*.

If no parameters are used, SetSpeed will change the global speed from the Animator component.

```
void SlowDown() {
    mecanimControl = gameObject.GetComponent<MecanimControl>();
    if (mecanimControl.GetSpeed() > 1) mecanimControl.SetSpeed(1);
}
```

setwrapmode **SetWrapMode**

```
void SetWrapMode(WrapMode wrapMode);  
void SetWrapMode(AnimationData animationData, WrapMode wrapMode);  
void SetWrapMode(AnimationClip clip, WrapMode wrapMode);  
void SetWrapMode(string clipName, WrapMode wrapMode);
```

Description: Sets the Wrap Mode of an AnimationData based on *clipName/clip*.
If no parameters are used, SetWrapMode will change *defaultWrapMode*.

```
void ClampCurrentClip() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.SetWrapMode(mecanimControl.GetCurrentAnimationData,  
WrapMode.Clamp);  
}
```

stop **Stop**

```
void Stop();
```

Description: Stops any animation from playing and starts playing the default animation.

```
void PlayDefaultAnimation() {  
    mecanimControl = gameObject.GetComponent<MecanimControl>();  
    mecanimControl.Stop();  
}
```

Public Events

```
Animation Data  
Current Clip Name: Default  
Wrap Mode: Loop  
Normalized Time: 0.1442214  
Blend Weight: 1  
Current Speed: 1  
Times Played: 1
```

Index:

- [OnAnimationBegin](#)
- [OnAnimationEnd](#)
- [OnAnimationLoop](#)

onanimationbegin OnAnimationBegin

`void AnimEvent(AnimationData animationData);`

Description: Fires when an animation begins.

```
void OnAnimationBegin(AnimationData animData) {  
    if (animData.clipName == "walk") Debug.Log("character is walking");  
}
```

onanimationend OnAnimationEnd

`void AnimEvent(AnimationData animationData);`

Description: Fires when an animation ends.

```
void OnAnimationEnd(AnimationData animData) {  
    if (animData.clipName == "walk") Debug.Log("character has stopped  
walking");  
}
```

onanimationloop OnAnimationLoop

`void AnimEvent(AnimationData animationData);`

Description: Fires when an animation loops. This is only triggered if the animation WrapMode is set to either *WrapMode.Loop* or *WrapMode.PingPong*

```
void OnAnimationLoop(AnimationData animData) {  
    if (animData.clipName == "walk")  
        Debug.Log("walking animation has looped "+ animData.timesPlayed + "  
times.");  
}
```

From:

<http://www.ufe3d.com/> - **Universal Fighting Engine**

Permanent link:

<http://www.ufe3d.com/doku.php/mecanimcontrol?rev=1426391876>

Last update: **2015/03/14 23:57**

