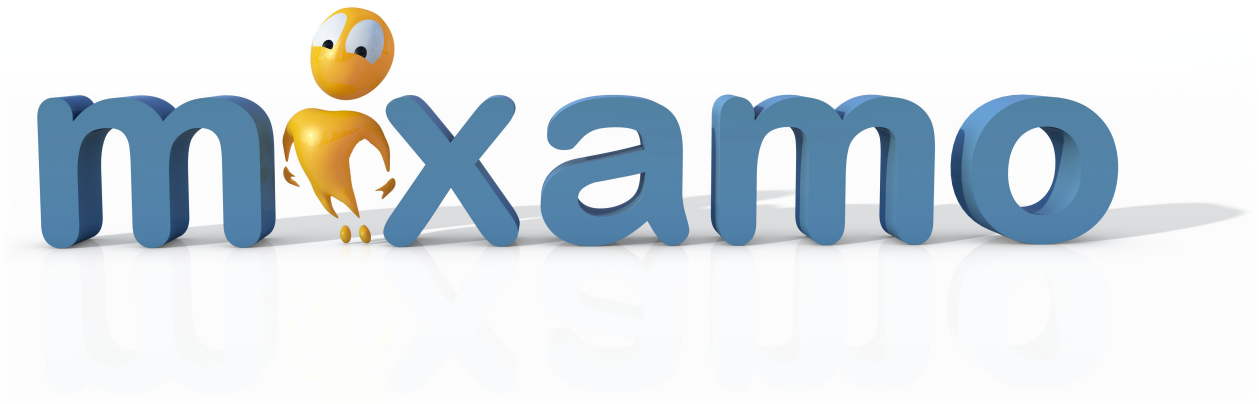

Root Motion Controller 2.0

Mixamo Animation State Machine

Documentation

v1.0

By

www.Mixamo.com

Basic Setup

The following steps illustrate how to set up the motion pack and controller on your character. These same steps can also be used if your just simply using the Mixamo Animation State Machine as your controller.

- Create an Empty GameObject that will act as the master node for the character or NPC
- Place the character (mesh with skeleton and animations applied) into the new Empty GameObject. (Thus making it a child of the Empty GameObject).
- Add the AnimationStateMachine.cs script to the Empty GameObject.
    - Set the Target to be the character place inside the Empty GameObject.
    - Set the Graph Text Asset to be the JSON .txt file containing all the state information.
- Add a CharacterController to the Empty GameObject.
    - Set it up to line up with the character's feet.
- Add the provided controller or your own custom controller.
- DONE

Overview

Overall, the Mixamo Animation State Machine system consists of the following pieces:

- Motion Pack definition JSON file
    - A JSON# encoded text file that specifies all of the motion clips that comprise the motion pack, and how they relate to each other.
- AnimationStateMachine Script
    - The main Unity Behaviour that drives the animation state + transition logic. This script uses definitions within the JSON to apply the motion onto the character and play corresponding animations.
- Game specific logic
    - The user then provides game specific logic that will drive the AnimationStateMachine via it’s public API.

A state machine is a system that utilizes a series of predefined states to interpret some kind of input. In the diagram above you can see that the keyboard input—depending on what keys are pressed—can activate different states in different orders.

JSON Documentation

The following document aims to explain the Mixamo Motion Pack JSON definition file in enough detail to allow others to create compatible JSON files for use with the Mixamo Animation State Machine set of scripts for character animation control.

JSON Definition - Root Object

Generally, JSON specifies nested structured data in a way that is realtively human readable and very parsable by code.

This is the root, main object that contains other data that actually describes the motion pack.

Root fields:

- name
    - The name or title of the motion pack
- root_path
    - The root node/joint in the target skeleton that motion data is applied to.
    - This node is special because depending on settings, the root motion can optionally be controlled by the motion data, or procedurally by the same developer / engine.
    - This is often “Hips”, “Root” or “Pelvis”
- clips
    - A list of clips that are used by this motion pack. Clips are described more below.
- layers
    - A layer is a concept in the Animation State Machine that defines a set of clips as being entirely separate from another. Typically all animations that control character moment should be placed on layer 0 the default layer.

Clips

An ordered list of clip instances, specifying the individual clips of motion that comprise the motion pack.

Clip instance fields:

- name
    - (required)
    - A identifying name used elsewhere in the JSON to specify this specific clip
- anim_name
    - (required)
    - A Mixamo specific identifier specifying the raw motion data for this clip
- type
    - (required)
    - Values: <normal>

- root_motion_translation
    - (not required : Defaults to no root motion)
    - Specify the primary axis direction(s) the clip contains
    - Values: [[x][y][z]] (Must be in xyz order)
- root_motion_rotation
    - (not required : Defaults to no root motion)
    - Specify the primary rotation axis the clip contains
    - Values: [x][y][z]
- layer
    - (not required : Defaults to “0”)
    - Specify layer you want the animation set to. This is similar to: animation[“run”].layer = 5;
    - Layers left alone with be applied to layer (0). Only layers on (0) can effect root motion.
    - It's a general rule of thumb that all mixing animation should be set to layers above (0) so as to not interfere with the root motion of the animations on layer (0)
    - Values: int
- mixing_transform
    - (not required : Defaults to all curves)
    - Specify transform joint to begin mixing animations at. If nothing is specified all animation curves will be used.
    - E.G. “mixing_transform”: “Hips/Spine/Spine1/Spine2/LeftShoulder” will play only the curves starting at LeftShoulder and down the chain from there.
    - Values: Transform (use / to create hierarchies)

Layers

A list of layer instances within the JSON separate from unity animation[].layer. Like an Array each instance of a layer increments its int value.

Only animation on the initial layer (0) effect the root motion of the character. All other layers above (0) should be considered mixing layers.

All layers must have at least one active state. Since this can cause issue with mixing layers above layer (0) See “Type” “blank” below.

Layer instance fields:

- name
    - A user friendly name for the layer.
- priority
    - At this time Priority does nothing
- states
    - A list of animation states that define how an animation should be played, transitioned to and from, whether or not to blend and how fast to crossfade if at all.

States

A list of state instances (or modes) the character animation can be in.

State instance fields:

- name
    - (required)
    - an identifying name for the state (will be called by clients using the API
- is_looping
    - (not required : Defaults to true)
    - Set is_looping to false if you want the animation to play through only once uninterrupted. Useful for animations like; death, reloading or jump
    - Value: boolean
- tree
    - (required)
    - Specifies the clip (or clips) that combine to makeup the state's action animation
    - See tree specification below.
- Transitions
    - (not required : Defaults to crossfade(0.0))
    - Specifies transition specific settings (and optionally clips related for moving from one state to another).
    - See transitions specification below.

Tree

Contains the information that represents the clip (or clips) that animate in the parent state.

Tree instance fields:

- type
    - The blending logic to use for the clip(s) specified
    - Values: <clip|blend2d|blank>
        - clip
            - This type means that the state's animation is specified by 1 specific clip
        - blend2d
            - This type means that the state's animation is the blendeded product of two clips (or trees)
        - blank
            - This type means that state is completely empty and can be used as a null state on layers above (0). This should always be the first state in layers above (0).

Clip Tree Type

Clip tree type specific fields:

- name
    - The identifying name of the clip (as specified in the clips section of the JSON.

Blend2d Tree Type

Blend2d specific fields:

- control
    - A name displayed to the user representing the percentage to blend between the two specified motions
    - Note: actual values for the blend are specified at run-time via the API.
    - See API Documentation below
- blend1
    - The first of the two clips (or trees) that should be blended
    - Values inside blend1 are the same as the tree object above but instead of <blend2d> they are <clip>
        - A tree of type blend2d can blend two tree's also of blend2d
- blend2
    - The second of the two clips (or trees) that should be blended
    - Values inside blend2 are the same as the tree object above but instead of <blend2d> they are <clip>

Transitions

The transitions section is a list of transition instances that specify specific animation and other logic that should be performed when transitioning from *this* state to the other state specified in the list of target states.

Transition instance fields:

- destination
    - The name of the destination state that this transition influences
    - The special value "*" can be specified which means use these settings for any transition that isn't specifically defined for this state.
- type
    - The behaviour type for this transition
    - Values: <clip|crossfade>
        - clip
            - Animate a specific clip as the transition from *this* state to the specified state
            - See clip transition type description below
        - crossfade
            - Crossfades from this animation to the next state's animation (in seconds)
            - See the crossfade transition type description below

Clip Transition Type

Animates a clip when transitioning for one state to another.

Clip transition type specific fields:

- clip
    - The clip name (as specified in the clips section of the JSON file) that should play when transitioning between the two states
- duration_in
    - The time (in seconds) to crossfade from the current state's animation to this transition specific clip
- duration_out
    - The time (in seconds) to crossfade from the transition's clip animation to the target state's animation

Crossfade Transition Type

- duration
    - The time (in seconds) to crossfade between the current state's animation to the destination state's animation

TIPS:

- JSON files are writen in ".txt" files.
- JSON doesn't recognize commenting such as "//" "///"or "/**/"
- Only declare an animation clip once.
- Only use an animation clip in a single state
- Never declare the same state in two different layers.
- Validate your JSON on jsonlint.com

API Documentation:

Contacting the API from your controller.

```
    // Define this variable
    private AnimationStateMachine asm;

    // Call this before the Update function to get and return the
    AnimationStateMachine from the GameObject.
    AnimationStateMachine GetASM() {
         return this.GetComponent<AnimationStateMachine>();
    }

    void Start () {
         // set AnimationStateMachine to equal "asm" for easy API
    calls.
         AnimationStateMachine asm = GetASM();
    }

    // Now you can make calls using “asm”
    asm.ChangeState(“run”);
```

//===\\

- GetRootMotion()
  - Returns the root motion that should be applied to the character's root node. (Read Only)
  - Always call this in LateUpdate.
  - If RootMotionMode is set to Automatic, this is done automatically and applied to this GameObject or character controller attached to it.
  - If RootMotionMode is set to Manual you will need to apply this to the GameObject or character controller manually.

```
    private AnimationStateMachine.RootMotionResult result; // Define
    Variable

    result = asm.GetRootMotion(); // call and store GetRootMotion
```

- GetRootMotion().GlobalTranslation : Vector3
  - Returns Root Motion based on worldspace (Read Only)
  - that can be used to drive the character controller or GameObject based on the root motion of the animation

```
    result.GlobalTranslation;
```

//===\\

- ChangeState(statename : string)
  - Call this function to change the current state. The default layer is 0. See the JSON section for more information on layers.

```
    if (Input.GetKey (KeyCode.W)){
        asm.ChangeState("run");
    }
```

//==========\\

- ChangeState(layer : int , statename : string)
  - Call this function and define the layer to change the current state within the defined layer. See the JSON section for more information on layers.

```
    if (Input.GetKey (KeyCode.W)){
        asm.ChangeState( 1, "reload");
    }
```

//==========\\

- GetCurrentState() : string
  - Returns current state playing on layer 0 and returns it. (Read Only)

```
    string currentState = asm.GetCurrentState();

    if (currentState == "run"){
        DoAwesomeFunction;
    }
```

//==========\\

- GetCurrentState(layer : int) : string
  - Returns current state playing on the specified layer and returns it. (Read Only)

```
    string currentState = asm.GetCurrentState(1);

    if (currentState == "reload"){
        DoReloadStuff;
    }
```

//==========\\

- GetCurrentDestinationState() : string
  - Returns the current state transitioning to layer 0. (Read Only)

```
string nextState = asm.GetCurrentDestinationState();

if (nextState == "prone"){
     SetUpProneStuff;
}
```

//==========\\

- GetCurrentDestinationState(layer : int) : string
  - Returns the current state transitioning to the specified layer. (Read Only)

```
string nextState = asm.GetCurrentDestinationState(1);

if (nextState == "stunned"){
     playerControl = false;
}
```

//==========\\

- ControlWeights[]
  - Control the input into different blend weights and state speeds using the parameters defined in the JSON. See the JSON section for more information.

Blend Weights between animations

```
     if(Input.GetKey(KeyCode.LeftShift){
          asm.ControlWeights["blend"] = 1; // 1 full blend

     else
          asm.ControlWeights["blend"] = 0; // 0 blend
}
asm.ChangeState( "move" ); //call this in the end to make the changes.
```

- ControlWeights["state_speed"]
  - Passing in "<statename>_speed" will allow you to directly access the speed of the state at any time.

```
if(isDazed) {
     asm.ControlWeights["run_speed"] = 0.4; // slow speed
     asm.ControlWeights["walk_speed"] = 0.5; // slow speed
}
```

//==========\\

Sample JSON


```
{
  "name" : "Sample Graph",
  "root_path": "Root",
  "clips" : [
    {
      "name": "basic root motion clip sample",
      "anim_name": "basic root motion animation name",
      "type": "normal",
      "root_motion_translation": "z"
    },
    {
      "name": "turning root motion sample",
      "anim_name": "turning root motion animation name",
      "type": "normal",
      "root_motion_rotation": "y"
    },
    {
      "name": "procedural clip sample",
      "anim_name": " procedural clip animation name",
      "type": "normal"
    },
    {
      "name": "mixing sample",
      "anim_name": "mixing animation name",
      "type": "normal",
      "layer": "1",
      "mixing_transform": "Root/Spine/Spine1/Spine2"
    }
],
  "layers" : [
    {
      "name": "locomotion layer",
      "priority": 1,
      "states": [
      {
        "name": "Basic State",
        "tree": {
          "type": "clip",
          "name": "basic root motion clip sample"
        },
        "transitions": [
                                        {
            "destination": "*",
            "type": "crossfade",
            "duration": 0.3
          }
        ]
      },
      {
        "name": "Basic none looping State",
                  "is_looping": false,
        "tree": {
          "type": "clip",
          "name": "basic root motion clip sample"
        },
        "transitions": [
```

```
                        {
          "destination": "*",
          "type": "crossfade",
          "duration": 0.3
        }
      ]
    },
        {
            "name": "blend2d sample",
            "tree": {
                  "type": "blend2d",
                  "control": "blend_sample",
                  "blend1": {
                        "type": "clip",
                        "name": "blend_at_0"
                  },
                  "blend2": {
                        "type": "clip",
                        "name": "blend_at_1"
                  }
            }
        },
        {
                  "name": "State with Destination transition",
                  "tree": {
                        "type": "clip",
                        "name": "idle"
                  },
                  "transitions": [
                        {
                              "destination": "walk",
                              "type": "clip",
                              "clip": "start_walk",
                              "duration_in": 0.3,
                              "duration_out": 0.0
                        },
                        {
                              "destination": "*",
                              "type": "crossfade",
                              "duration": 0.1
                        }
                  ]
        }
    ]
},
{
      "name": "Mixing Layer 01",
      "priority": 2,
      "states": [
      {
                  "name": "blank state sample",
                  "tree": {
                        "type": "blank"
                  },
      "transitions": [
      {
                  "destination": "*",
                  "type": "crossfade",
                  "duration": 0.3
```

```
        }
        ]
        },
        {
        "name": "mixing sample",
        "tree": {
        "type": "clip",
        "name": " mixing animation name "
        },
        "transitions": [
        {
                "destination": "*",
                "type": "crossfade",
                "duration": 0.3
        }
        ]
        }
        ]
        }
  ]
}
```