User Manual

Table of Contents

Table of Contents **2**

1. Introduction **3**

1.1 What Is Strider? 3

1.2 Standalone Demo **4**

1.3 Support **4**

2 Getting Started **5**

2.1 Prerequisites (IMPORTANT) 5

2.2 Installation 6

2.3 Demo Scenes **8**

2.4 Basic Setup 10

3 StriderBiped Component **11**

3.1 Settings 11

3.2 Stride Style **13**

3.3 Playback Speed Control 13

3.4 References 14

3.5 Callbacks **14**

4 Scripting with Strider **15**

4.1 Exposed Properties 15

4.2 Utility Functions 16

4.3 Custom IK Method **16**

5 Integrations **17**

5.1 Motion Matching for Unity 17

6 Execution Order **18**

1. Introduction

This is the user manual for ‘Strider’, an animation editor extension plugin for Unity game engine. The goal of this manual is to provide all information about strider from installation and general use to technical C# interfaces.

This written guide is a supplement to the tutorial video accessible from the Tutorial Playlist.

1.1 What Is Strider?

Strider is a simple yet very effective procedural animation tool which enables more believable locomotion by scaling the character’s stride instead of playrate to achieve different movement speeds. This can be used in various situations such as:

- Speed boosts / slow downs
- Analog movement speeds (e.g. with a gamepad joystick)
- Character size / scale compensation (i.e. different scale characters walking together)

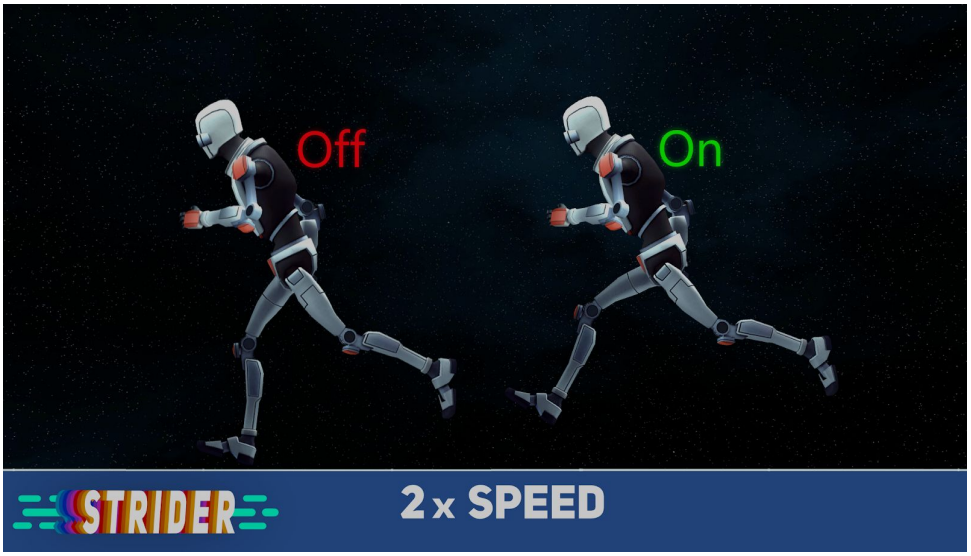

Figure 1.0 Stride Comparison

Traditionally you would either lock the potential character speed to digital steps based on your available animations (e.g. idle -> walk - > run) or you would scale the playback rate of your animations. The first method results in unresponsive gameplay while the second looks very silly (either slow motion or energizer bunny). Strider allows you to get the best of both worlds with responsive analog speeds that also look believable.

Within a particular movement form (e.g. running), humans don’t generally speed up much when going faster, instead we take longer strides with just a little bit of speed up. Strider models this behaviour. It uses a stride warping algorithm to scale the stride and adjust hip movement without hyper-extension. It then uses IK to procedurally modify the animation and achieve the desired result.

In Unity, ‘Strider’ manifests as a single component. This component is very flexible and has a number of settings to control how the stride is modified. This includes hip movement controls,

stride offsetting, animation playback limits and more. The component is very performant and uses Unity's job system and burst compiler for extreme optimisation.

1.2 Standalone Demo

A standalone demo is available to download and play for free without purchase. This demo shows and explains the features of strider, why you might want it and how it could be used. The demo scenes in the standalone demo are all available within the package open source.

The standalone demo can be downloaded for free here.

1.3 Support

Discord - send me a private message once you have joined with your Invoice Number so I can give you verified status and access to the private discord channels.
Tutorial Videos - In depth tutorial videos on using Strider
Issue-Tracker - Found a bug? Report it here.
Forum
Unity Connect
Asset Store Page

Note: For the best level of support please use the Discord. It is the easiest way for me to verify that you are a valid customer. I will only provide support for verified customers as an anti piracy measure.

IMPORTANT: Import your animations correctly. Strider cannot fix your broken animations!

2 Getting Started

The following sections will assist with getting started with Strider. Please read this section carefully as the asset may not work if not setup properly.

2.1 Prerequisites **(IMPORTANT)**

'Strider' requires Unity 2019.1.14f1 as a minimum. It also requires Unity's new job system and a few additional packages. If you downloaded the package from the Unity Asset Store, you should be prompted to install these dependencies automatically. However, if you are installing it via a package from the Discord server, then you will need to manually install these dependencies.

- **Mathematics**
- **Collections** (Preview)
- **Burst**
- **Jobs** (Preview)

For the demo scenes you will need to download the **Cinemachine** package to get the camera to work. You will also need to add two input settings in *'Edit / Project Settings / Input'*, as shown in Figure 1, for the demo scene to work without errors.

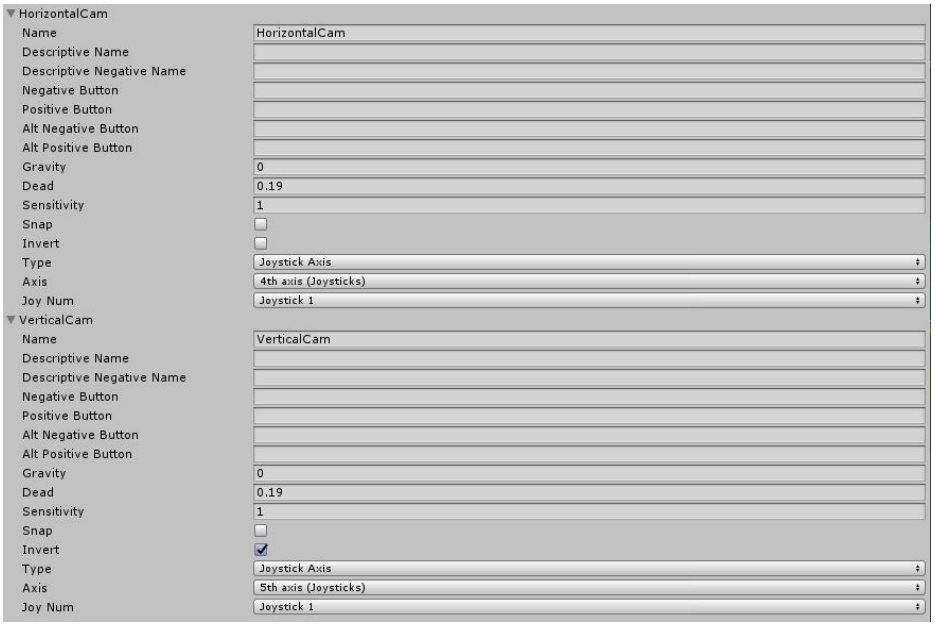

Figure 2.0 'Input settings required for demo scene''

To download packages navigate to Window -> Package Manager in the Unity Editor. Select the required package and choose install. For the 'Preview' packages you will need to click on the 'Advanced' drop down and make sure 'Show Preview Packages' is checked before it will show up in your list.

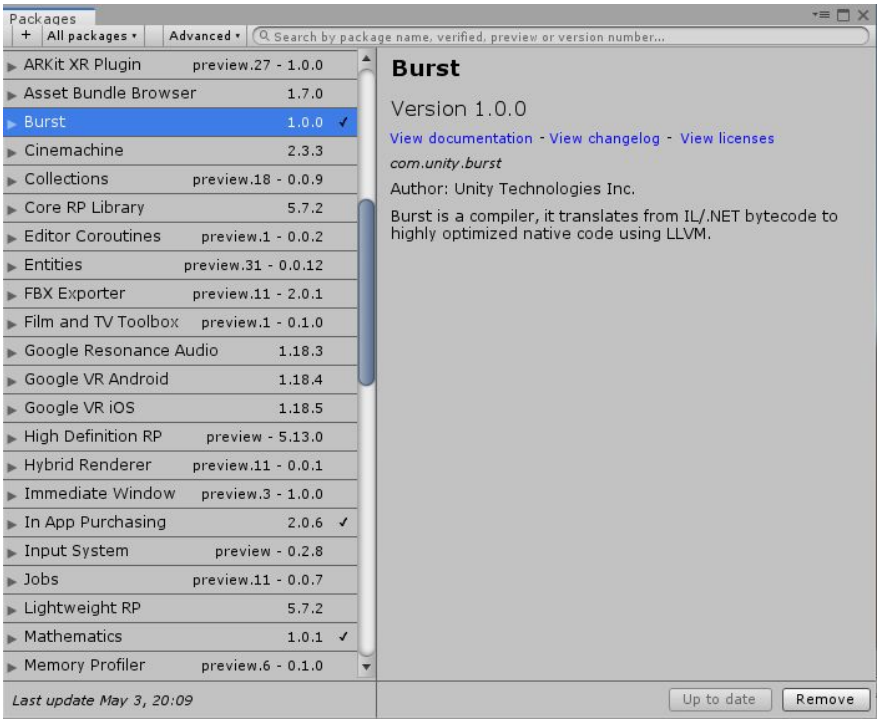

Figure 2.1 'Unity Editor package manager, install Burst, Mathematics & Jobs"

2.2 Installation

Once you have purchased Strider you will be able to download it through the Asset Store window within Unity itself. Choose 'Window' / 'Asset Store' and navigate to 'My Assets' (figure 3). Find the 'Strider' package and choose 'Download'.

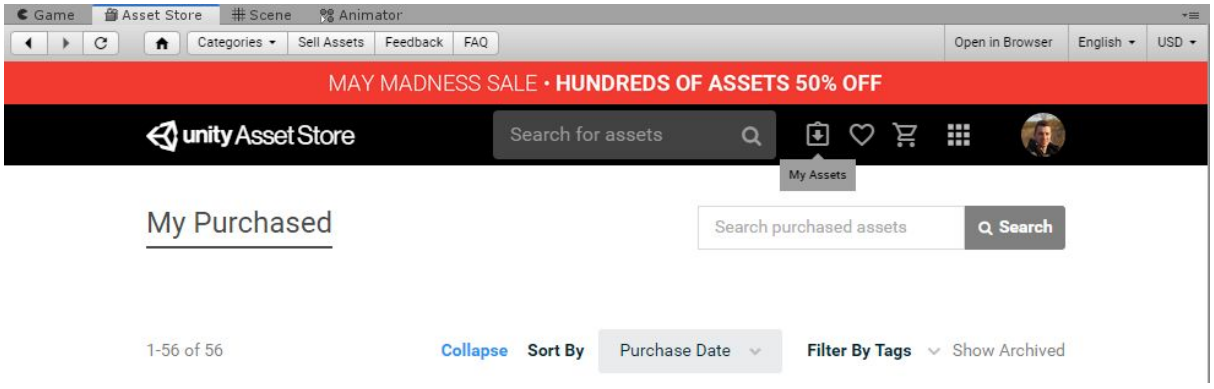

Figure 2.3 - 'Asset store embedded in the Unity Editor"

Once the download is complete choose import. You will be presented with the import window (figure 4). Leave everything checked and click the import button.

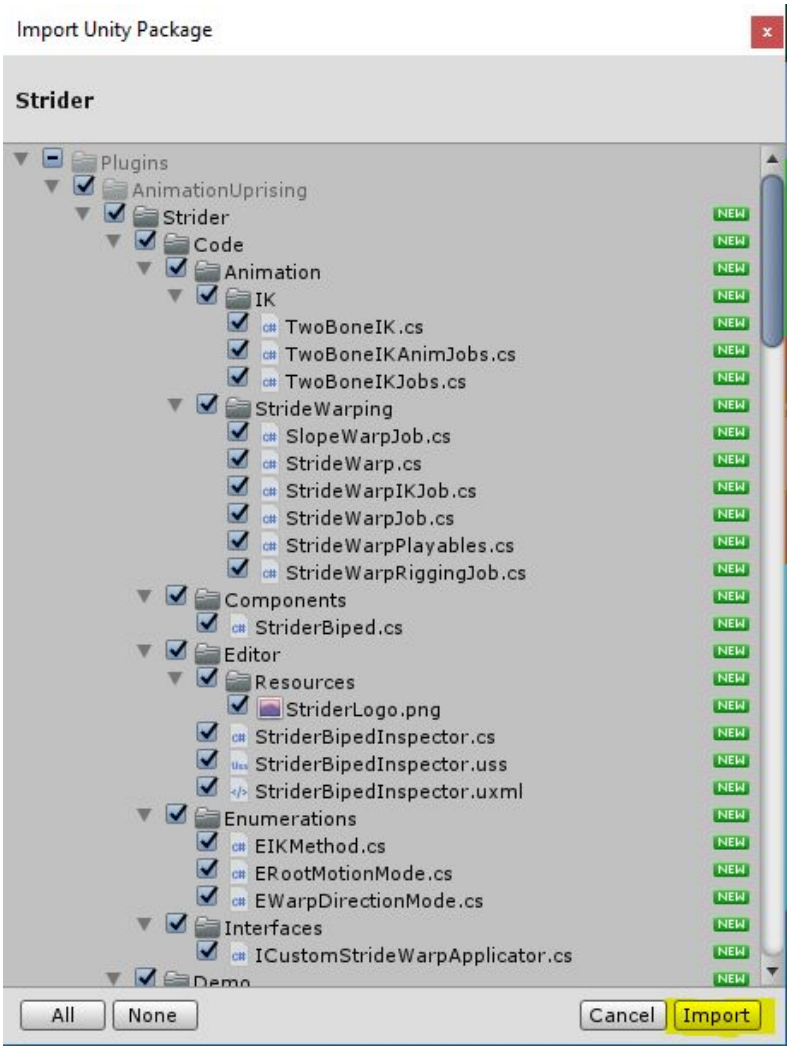

Figure 2.4 - 'Import window for Strider'

After a brief wait the package should be fully imported into your project.

Note: In some versions of 2019.3x +, the asset can be downloaded and imported from the package manager.

2.3 Demo Scenes

Strider comes with three (3) demo scenes which can all be accessed through the ‘StriderDemoBootstraper’. These are the exact same scenes as provided in the ‘Standalone Demo’.

In the project view navigate to ‘Plugins / Strider / Demo / Scenes’ and open the ‘StriderDemoBootstrapper’ scene (figure 5).

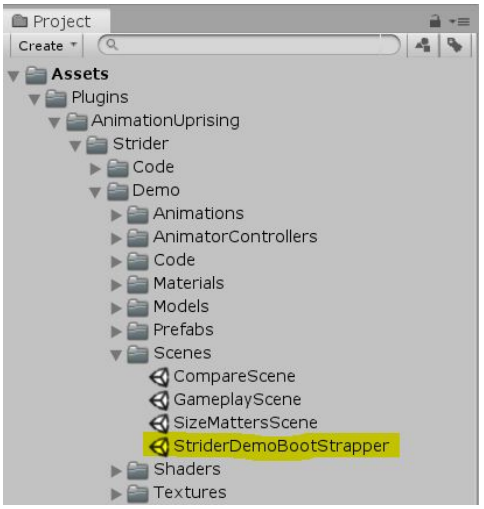

Figure 2.5 - ‘Demo scene location’

After entering playmode, you can navigate to the three scenes to see Strider in action. The scenes available include:

- **Comparison** - View a side on comparison of two characters moving at the same speed. The character on the left has ‘Strider’ off, while the one on the right has ‘Strider’ on. Change the character speed on the central slider to see how the characters react differently to speed changes. Change the animations in the top left panel and Strider settings in the top right panel.

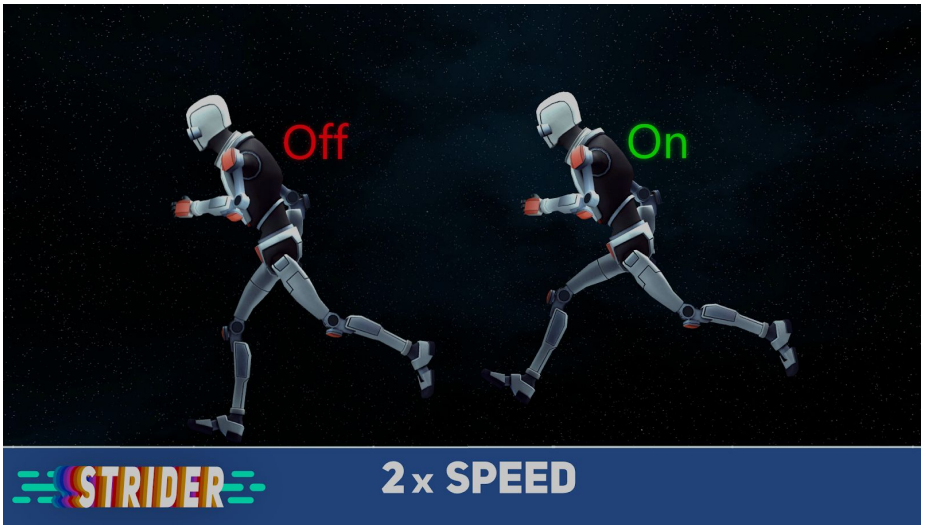

Figure 2.6 Stride Comparison Scene

- **Scale -** This demo shows how multiple characters of different size and scale can be made to move in sync at uniform speeds and pace, with the same animations and without footsliding. Large characters have their stride scaled down, while small characters have their stride scaled up. Click the spawn button in the top left to keep spawning randomly scaled groups. Turn strider on and off using the panel on the top right.

Figure 2.7 Size Compensation Scene

- **Gameplay -** This demo shows an example of strider in a third person setting like you might see in a game like Fortnite. In this case strider is being used to attain all the 'in-between' speeds required for an analogue gamepad stick. If you are using a keyboard and mouse, pick up the speed boosts (green boxes) and slow downs (red boxes) to see how the stride is affected by these 'buffs or debuffs'.

Figure 2.8 Gameplay Scene

2.4 Basic Setup

The core of ‘Strider’ is the ‘StriderBiped’ component. Simply add this component to your character and you should be able to start speed scaling right away. If you are using a ‘generic’ rig, first be sure to fill out the ‘references’ foldout in the inspector. This will be done automatically at runtime for humanoid rigs.

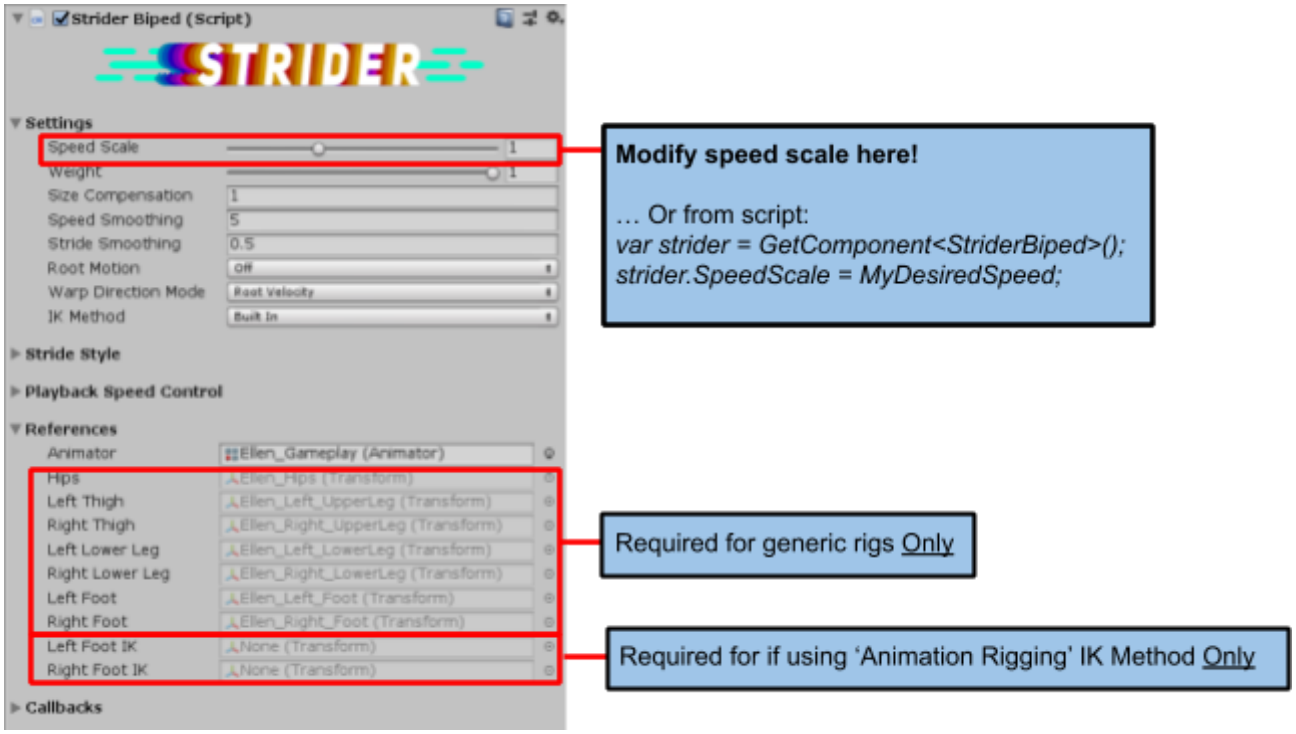

Figure 2.9 Basic Inspector Setup

Enter playmode and move the ‘SpeedScale’ slider in the inspector. Note your character’s stride getting longer when at higher speed scale and shorter at lower speed scale. This speed scale control, and all other settings, can be easily modified through code at runtime as described in Section 4.

Note: Don’t be concerned if it looks a bit silly at first, depending on your animations and character, you may need to modify some other settings. Please see Section 3 for details on all StriderBiped settings.

3 StriderBiped Component

The StriderBiped component is the core of Strider. This section will go into detail on all of the settings available from the component and how they should be used. The full StriderBiped component is shown in Figure ##.

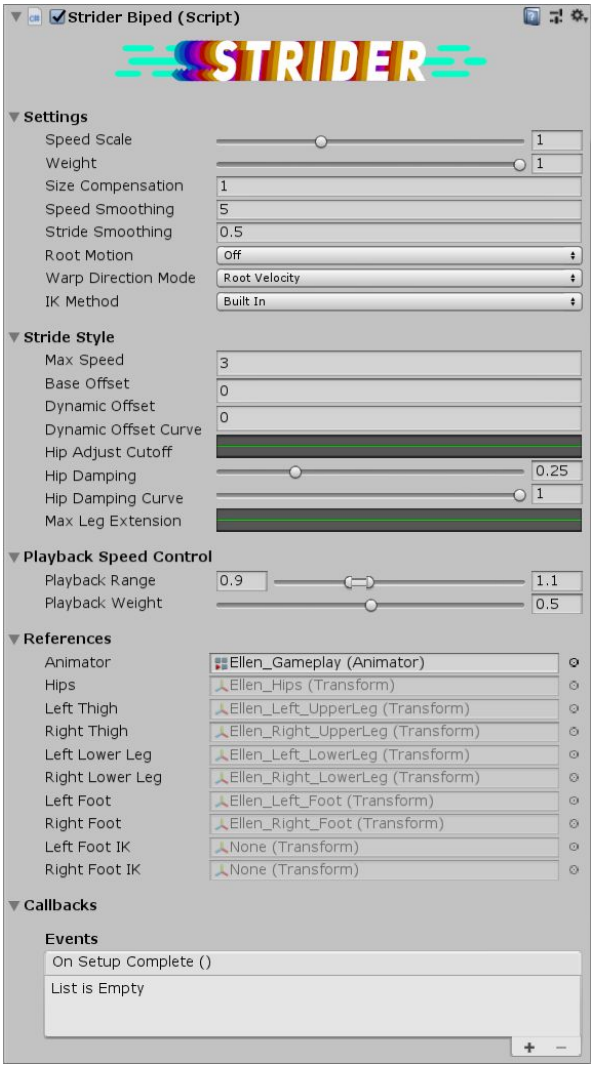

Figure 3.0 'The StriderBiped component'

3.1 Settings

This foldout contains various general settings for strider. These settings are as follows

Speed Scale (%)**:** This is the overall speed scale for the character combining both stride scale and playback speed scale. This is the primary setting of strider and it is intended that this value will be modified at runtime.

Weight (%)**:** The 'weight' of strider is how much it should influence the underlying animation. Most of the time this should be set to 1 (full influence). Generally the weight is used at runtime to smoothly (see Section 4) turn strider on and off but it is exposed here for flexibility.

Size Compensation (%): This value can be used to compensate for character size so that it's movement can match other characters of different size. If the character model has simply been scaled, the size compensation should equal the scale. However, sometimes characters are larger or smaller simply because they use a different model. In this case you will need to figure out the exact value yourself.

Note that size compensation is different to speed scale as it doesn't affect playback rate so that characters can remain in sync. Do not use SpeedScale for size compensation or vice versa.

Speed Smoothing (units/s)**:** Speed smoothing is a value used for smoothly changing the Speed Scale at runtime instead of instantly. Instant speed scale changes result in jittery animation. Generally at runtime, a 'desired' speed scale is set and the 'actual' speed scale is moved towards the desired scale via this smooth rate.

Stride Smoothing (units/s): Stride smoothing is how smoothly the stride pivot point changes position. This helps prevent animation jitter. The lower this value is, the less likely there will be jitter but more foot sliding will be introduced. Higher values eliminate foot sliding but introduce jitter. The default value of 0.5 is recommended in most cases.

RootMotion: Root motion can be set to either one of the following options:

- On - Root motion will be automatically applied
- Off - root motion will not be automatically applied

WarpDirectionMode: For proper stride scaling, a stride warp direction is required. If your animations have root motion (even if you aren't using root motion) The best option is to use the 'Root Velocity' setting. Other options are as follows:

- Root Velocity - Stride direction will automatically be calculated by the root velocity of your animations.
- Actual Velocity - Strider will continually calculate the actual velocity of your character in global space and use this as the stride direction.
- Character Facing - Stride will always use your character's forward facing direction as the stride direction
- Manual - With this setting the user must specify the stride direction any time it changes. See Section 4.1 for more details.

IK Method: Strider has its own built in IK methods which are recommended for most cases. However, it is possible to use other IK methods including Unity's mecanim IK and even third party options like FinalIK. Options are as follows:

- Unity IK - This uses Unity's built in mecanim IK. For this to work, the character must be using mecanim and 'IK Pass' must be checked on the mecanim layer.
- Built In - This is the built in IK method packaged with strider. It is performant and uses the job system. With this method, the IK is processed in late update. This is the recommended default setting.
- Built In Anim Jobs - This is an alternative built in IK method that runs within the animation update using Animation Jobs. This method is very useful if using Strider with Unity's 'Animation Rigging' package and need the stride to be applied before animation rigging constraints are applied.

- Custom - Strider allows the use of external IK systems like FinalIK that run in the 'Late Update' loop. However, results and ease of use may vary depending on the custom IK implementation. See section 4.2 for more details.

Manual Root Motion: This option is for a workaround for a bug in Unity where animation jobs fail to work properly on certain rigs / animation clips. If you encounter this problem, please ensure to use either BuiltIn or UnityIK Ik methods and check this box.

Warning: This bug within Unity Animation Jobs has been reported. If you encounter it you will have to use the ManualRootMotion workaround. If you are handling your own root motion, you will have to multiply base movement by StriderBiped.StrideScale (not SpeedScale!) in your OnAnimatorMove method. See the gameplay demo for an example or contact me on the Discord and I can help you through it.

3.2 Stride Style

The stride style foldout contains settings relating directly to the style of the character stride. Options are as follows:

Max Speed (units/s)**:** This value needs to be set manually from the fastest speed that your character can move (e.g. the running speed). While it doesn't control your maximum speed, it is used to calculate a speed ratio which is used to sample curves (see below) that allow for dynamic stride style changes based on your movement speed. For example, you may want more hip damping (less bounce) while running fast when compared to walking.

Base Offset (units)**:** Typically stride is scaled from a central point at the ground directly below the character's hips. This scale point can be shifted forward and backward along the stride direction using stride offset. The base offset is the default offset from this central point.

Dynamic Offset (%)**:** In addition to base offset, stride can be dynamically offset based on the speed of your character. This value is multiplied by the Dynamic Offset Curve (below) sample with a speed ratio which is calculated from your defined 'MaxSpeed'.

Hip Adjustment Cutoff (units)**:** The maximum amount of hip adjustment allowable. Any hip adjustment calculated by strider will be clamped so that it does not exceed this value.

Hip Damping (%)**:** The amount of damping to be applied to hip movement. This should be a value between 0 and 1. Lower values provide more damping while higher values provide less damping. This value will be multiplied by the Hip Damping Curve which is evaluated via a speed ratio calculated from 'Max Speed'.

Hip Damping Curve (%): This curve can be used to modify the amount of hip damping at different speeds. For example, it is usually desirable to have more hip damping at run speeds (less bobbing up and down) as the character is essentially jumping at full stride, while at walking speeds, or slower speeds, there should be more vertical movement as one foot is always in contact with the ground.

3.3 Playback Speed Control

Strider does allow some amount of animation playback speed scaling. This playback speed scaling is mixed with the stride scaling to achieve the total desired speed scale. These settings allow you complete control over the limits of the playback speed scaling as well as the weight / rate that it is blended with stride scaling.

Playback Range: The upper and lower limits of allowable speed playback scaling. Keep the variation low. Typically playback speed scaling starts to degrade quality beyond 10% scale.

Playback Weight:. Until the upper or lower limit of playback scale is reached, playback scale and stride scale will be blended via this weighting. Higher values mean that playback ratio. Higher values means more playback scale while lower values result in higher stride scale. It is recommended to keep this value around 0.5 at least to start with.

3.4 References

Strider requires a number of references to other components and animation joints. Most of the time these references will be fetched automatically.

Animator: A reference to the animator that this instance of strider should operate on. Most of the time this is fetched automatically if Strider is placed on the same game object as the animator component.

Joint References: The following joint references are needed for Strider to work. If you are using a humanoid rig, then these will be filled automatically at runtime. However, for generic rigs you need to fill them manually.

- Hips
- Left Thigh
- Right Thigh
- Left Lower Leg
- Right Lower Leg
- Left Foot
- Right Foot

3.5 Callbacks

This foldout contains Striders event callbacks. In its current state, Strider only has one event which is called once all back end setup is complete. Any other component or system that depends on strider should only initialize after strider setup is complete using this event callback.

Using Animation Rigging with Strider
Whether you are using the ‘Animation Rigging’ IKMethod or are using Unity’s Animation Rigging package over the top of strider, it needs to be enabled after strider has completed setup to do this, add a callback to enable the Animation Rigging ‘Rig Builder’ component from the

OnSetupComplete event as shown below:

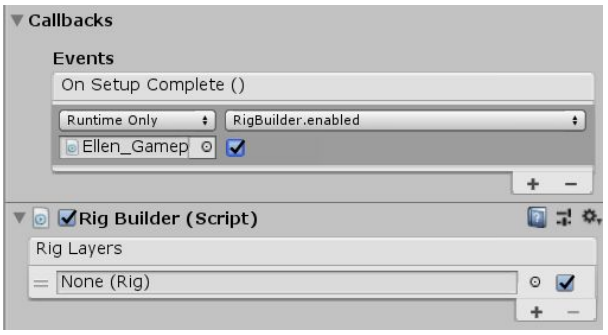

Figure 3.1 'OnSetupComplete event to enable RigBuilder'

4 Scripting with Strider

Strider doesn't know how your game works or how it should modify speed unless you tell it. Every game is uniquely different and therefore, you need to convey your desired speed scale to strider, just like you would have to for the animator. Fortunately, 'Strider' has a very easy to use scripting API which allows you to change almost every setting on the component at run time as well as functions for smooth transitions.

4.1 Exposed Properties

The following exposed properties allow manipulating of settings exposed in the inspector through code.

```
//Use this property to change the speed scale at runtime.
float SpeedScale;

//General Settings
float SizeCompensation;
float SpeedSmooth;
float StrideSmooth;
ERootMotionMode RootMotion;
EWarpDirectionMode WarpDirectionMode;

//Stride Style
float MaxSpeed;
float HipAdjustCutoff;
float HipDamping;
float BaseOffset;
float DynamicOffset;

//Playback speed control
float MinPlaybackSpeed;
float MaxPlaybackSpeed;
Vector2 PlaybackSpeedRange;
float PlaybackSpeedWeight;

//Internal
float StrideScale;  //This is the final stride scale following calculations
```

Note: IK Method cannot be changed at runtime

4.2 Utility Functions

Instead of directly disabling and enabling the strider component instantly, use these functions to smoothly blend strider in and out. Enabling and disabling of the component will occur automatically.

```
public void DisableSmooth(float a_fadeRate);
public void EnableSmooth(float a_fadeRate);
```

4.3 Custom IK Method

While it is recommended to use one of the other available IKMethods built into Strider, it is possible to use an external custom IK method. To achieve this, a custom script must be made which communicates between strider and your custom IKMethod via the ICustomStrideWarpApplicaor interface:

```
public interface ICustomStrideWarpApplicator
{
    void StrideWarp(float a_hipAdjustY, float3 a_leftFootPos, float3
        a_rightFootPos);
}
```

The ‘StrideWarp’ function must be implemented to apply the foot positions and hip adjustment calculated by Strider.

a_hipAdjustY - Required hip adjustment on the Y axis
a_leftFootPos - Target position for the left foot IK
a_rightFootPos - Target position for the right foot IK

5 Integrations

Most of the time strider will be able to integrate with other systems without any special scripts. However, there are exceptions to this. The following is a list of integrations and how to get them working.

Note: If there is an asset that you think should have an integration with strider please notify me through the ‘feature-requests’ channel on the Discord server.

5.1 Motion Matching for Unity

Strider is best used with Motion Matching for Unity via ‘longitudinal error warping’. The goal of this is to modify the speed of the character to match the desired trajectory speed even if the current animation doesn’t match perfectly.

To install this integration, import the ‘MotionMatchingforUnity’ package located in the ‘Integrations’ folder. This will provide an additional script called ‘StriderBipedMxM’, which extends ‘StriderBiped’ so that it can be used seamlessly and automatically with MxMs warping systems.

6 Execution Order

As strider is a procedural, ‘post process’, animation tool, it runs during and immediately after the animation update. The exact update point is dependent on the method of IK chosen as follows:

UnityIK - Updated within the OnAnimatorIK update during the animation loop
BuiltIn - Updated immediately after the animation Update (in LateUpdate)
BuitIn_AnimationJobs - Updated during the playable graph evaluation, immediately after the underlying pose has been determined (via mecanim or other)
Custom - Updated in LateUpdate immediately after the animation Update.

It is always vital that Strider be updated after the animation update but before any other IK or procedural animation tools are updated (e.g. a grounder from FinalIK). Your gameplay code that affects the movement of the character needs to come before the Animation Update for best results.

The default execution order for strider is set to -50 in the project settings pane and can be modified if necessary.