User Guide

www.syntystore.com

Table of Contents

1. Introduction

Welcome to Synty Sidekick Characters, a flexible, modular character building system, designed to give game developers the ability to create detailed and diverse characters with an ever growing variety of themes and customisation tools.

From all of us at Synty,
We thank you for the support!

2. Installation and set up

Requirements

- Unity Version 2021 LTS+ or later
- This product is currently in early access and is only compatible with the Universal Render Pipeline (URP)

Installation

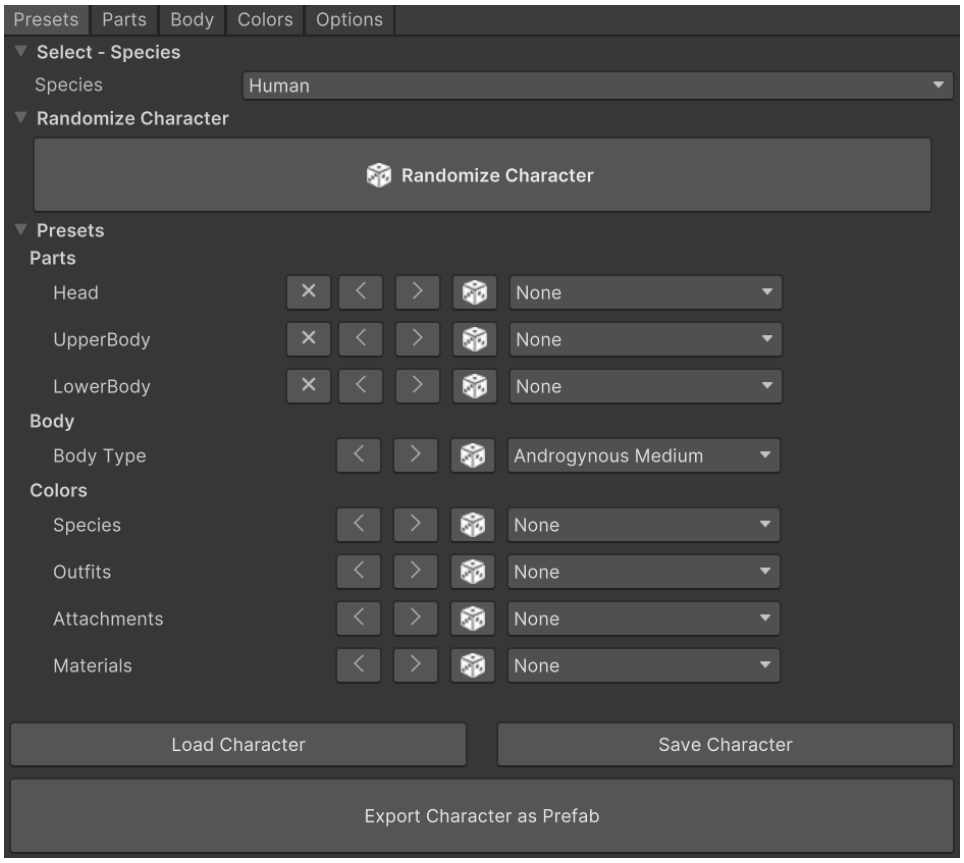
1. Open your Project window - '**Window**' > '**General**' > '**Project**' (from the top menu select)
2. Import Sidekick Characters - Right Click in the Assets Directory and Select '**Import Package**' > '**Custom Package...**' (from the right click menu)
3. Navigate to where you downloaded the .unitypackage file and click '**Open**'
4. You will be presented with a window to import the package, click '**Import**'.

3. Quick Start

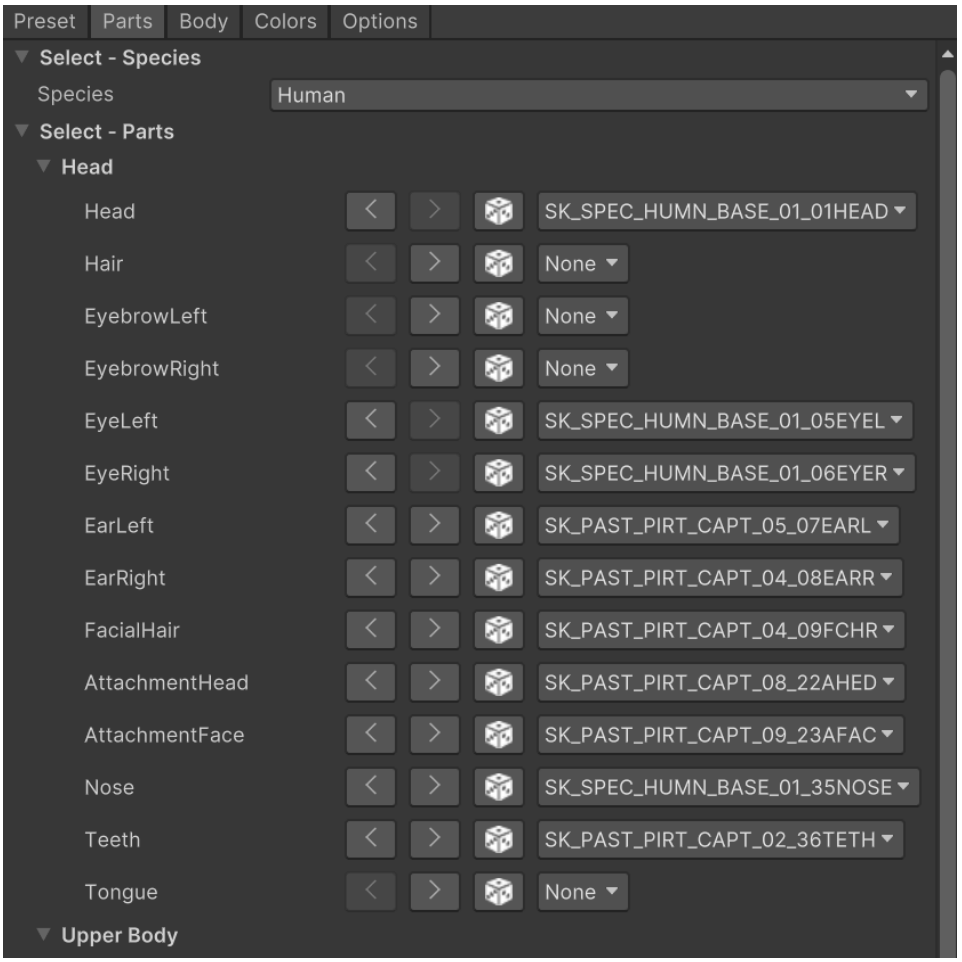
Build a Character

The following example will walk you through a typical use case for creating a character with the Sidekick Character Tool as a means to guide your experimentation further to find your own use cases. This assumes you have installed it correctly and starting from a new scene.

1. First open the '**Synty**' > '**Sidekick Character Tool**' window (from the top menu select)
2. Edit your character in the **Presets tab** (see below image) by changing the selected presets, **Part Presets** are split into Head, Upper body and Lower Body, these are collections of precompiled parts that can make up a character, **Body Type Presets** are combinations of genders and body mass, **Color Presets** are groups of colors for Species, Outfits, Attachments and Materials.



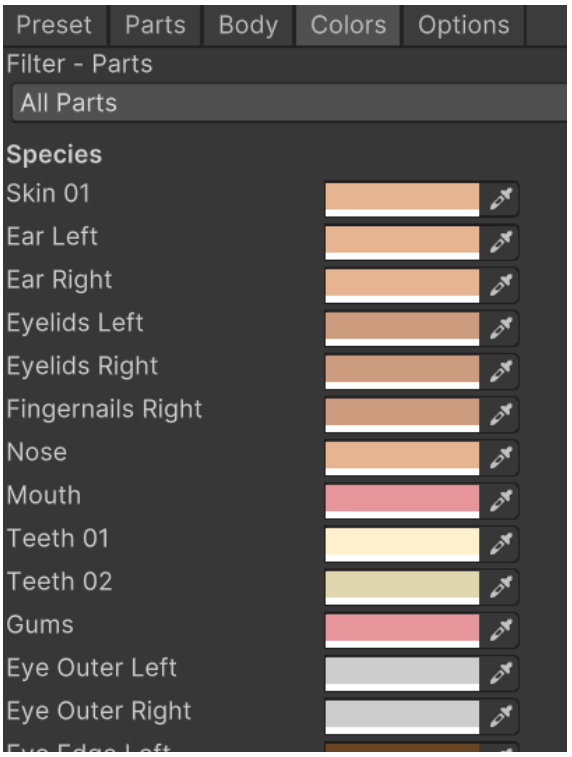
3. Refine your characters Parts via the **Parts tab** (see below image), by changing the individual selected parts, the character will update with your new selection.



4. Refine your characters Body Type via the **Body tab** (see below image)**,** by changing the **Body Type**, **Body Size** and **Musculature** sliders, the characters body type will change.

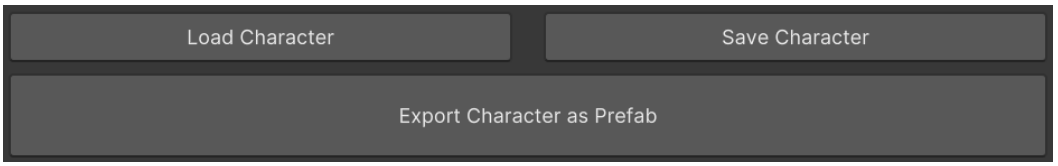
5. Refine your characters Colors via the **Color tab** (see below image), by changing the colors individually, the characters colors will change.

(The color tab will only show colors that are used by all of your characters parts. To see all possible colors, in the **Options tab**, deselect the '**Filter colors by currently selected parts**' option.)



6. Once you are happy with your character, you can create a prefab to use in your game via the '**Create Character Prefab**' button (at the bottom of the tool). Save your prefab in your project directory.

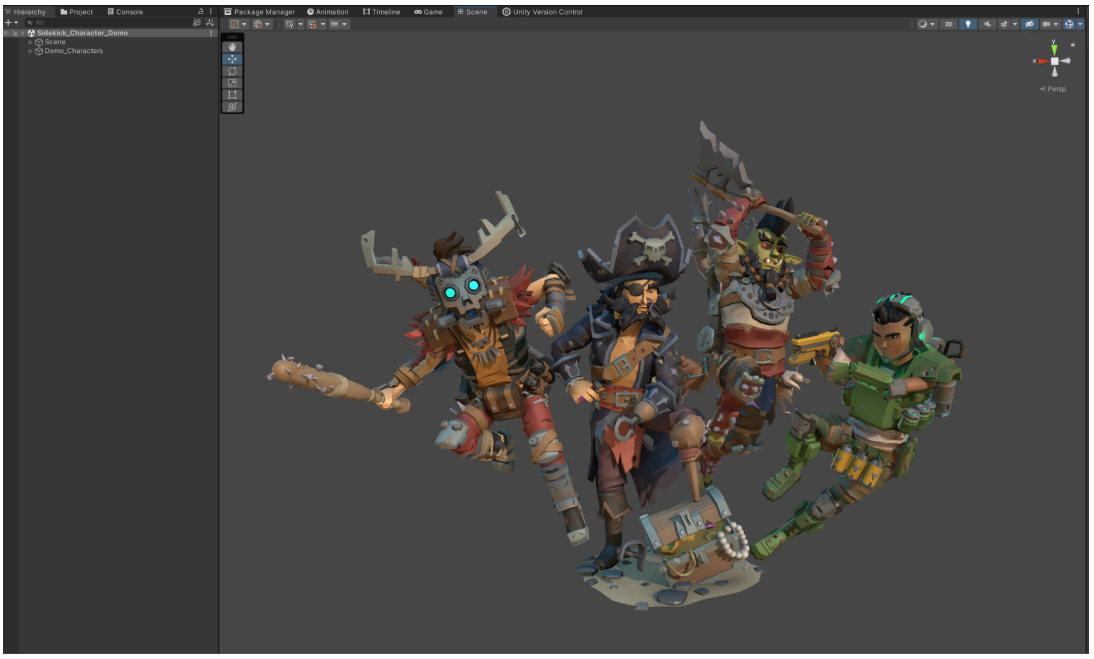
7. If you would like to save your characters settings so you can edit them later, save the character to file via the '**Save Character**' button. (at the bottom of the tool). Load character settings files via the '**Load Character**' button (at the bottom of the tool) to continue editing.



Demo Scenes

Sidekick Character Demo

Project location - Assets/Synty/SidekickCharacters/_Scenes/Sidekick_Character_Demo
This scene can be used to set up a new character with some preset lighting to help see the character clearly in a potential lighting set up a user might have in-game. (hide the Demo_Characters)



Sidekick Facial Demo

Project location - Assets/Synty/SidekickCharacters/_Scenes/Sidekick_Facial_Demo

This scene demonstrates the use of the various animations in combination with a layered animator to create an animated moving character with facial animation. The scene demonstrates:

- The sample prefab character made with the Character Creator, driven by a sample animator
- UI driven swapper to cycle through facial expressions and poses
- Animator sample for how a user could set up a system for their own project

4. Using your Characters

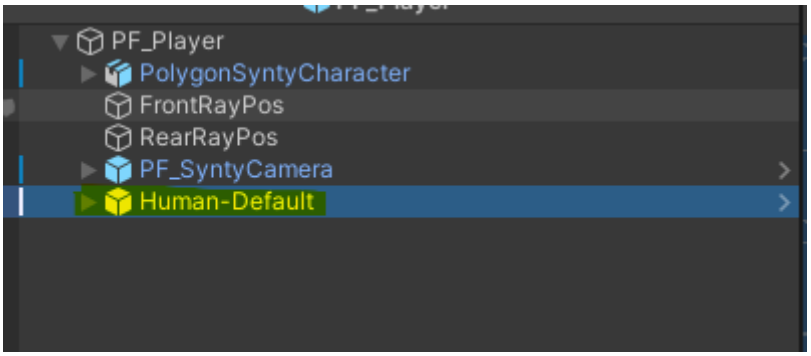
Add an animator with animation
Note : Characters are setup as humanoid and can utilized most animations out there

ANIMATION | Base Locomotion Asset Pack

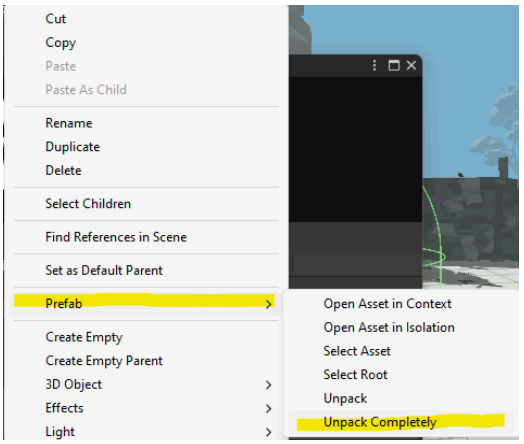
Our Animation package allows you to get your character moving and playable in an instant.

Setting up in Unity (Version 2021 and older)

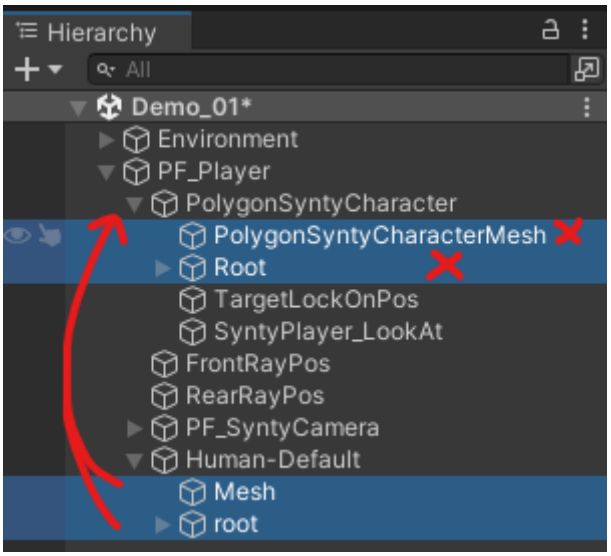
1. Open the PF_Player asset in **Synty \ AnimationBaseLocomotion \ Samples \ Prefabs**
2. Drag your prefab asset into the player prefab under PF_Player



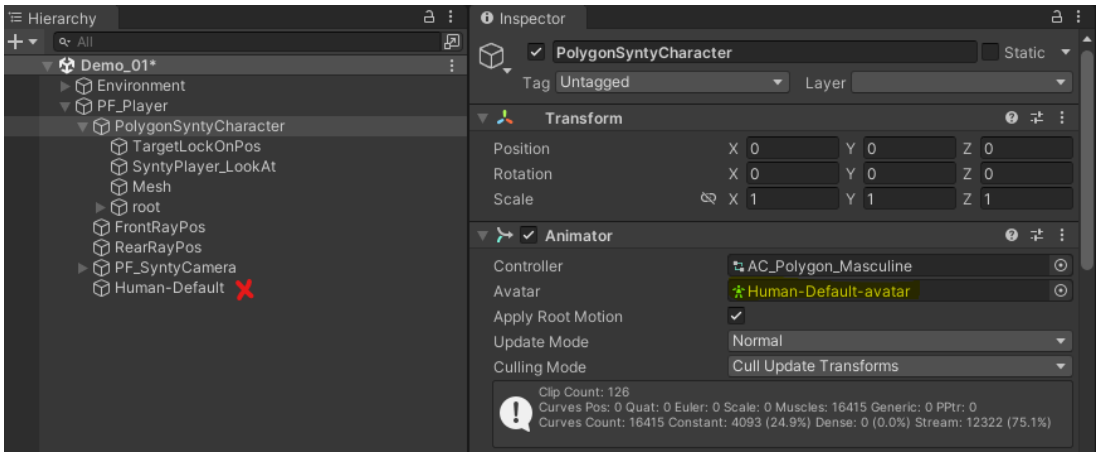
3. In the scene, right click on your prefab and select unpack your character Prefab, and then do the same to the PF_Player Prefab



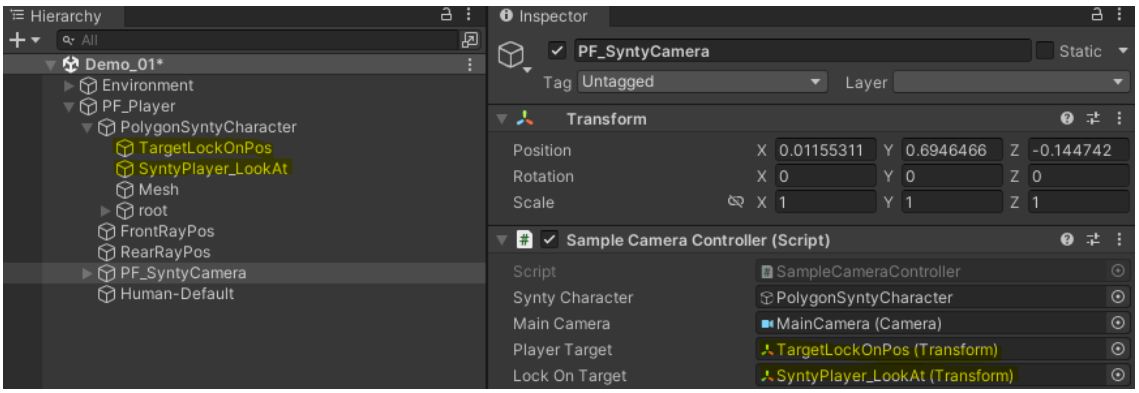
4. Delete the PolygonSyntyCharacterMesh and the Root under the PolygonSyntyCharacter node, and replace them with the Mesh and root from your Sidekick prefab



5. Assign the avatar of your prefab to the Avatar tab in the Animator on the **PolygonSyntyCharacter**, and delete the other prefab node from your character that is now empty



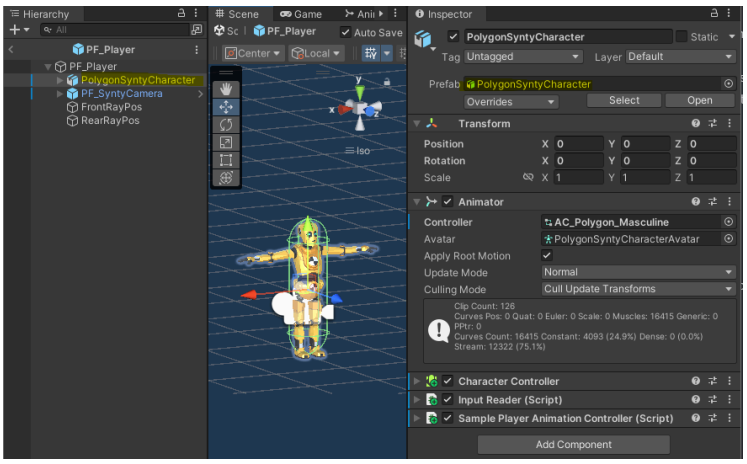
6. Check that your **PF_SyntyCamera** settings point to your Prefab, hit the small circle on the right of any that say missing to find the correct asset to aim at.
7. Swap the new meshes layer from **Default** to **Character**, and apply that to children as well



8. Hit Play to run around the scene as your newly created character

Setting up in Unity (Version 2022 onwards)

1. Once the package is installed, open the Demo_01 Scene in **Synty\AnimationBaseLocomotion\Samples\Scenes**
2. Select the PF_Player asset and in the inspector select Open in the Prefab section and drag in the prefab of the character you have made with the Character Creator Tool into the Prefab PF_Player and select Replace and Keep Overrides
3. Swap the new meshes layer from Default to Character, and apply that to children as well



Facial Animator setup

Within **Assets \ Synty \ SidekickCharacters \ Samples \ Animations** there is a **AC_FacialAnim** asset with some layers applied that house two types of facial animation, the **Emotion_Additive** and the **FaceAnim_NeutralCycle**.

- **Emotion_Additive** are poses that can be used with the overrides to add a base pose or drive facial expressions
- **FaceAnim_NeutralCycle** are Loop override animations for movement on the face

Grumpy	Angry	Enraged	Smug
			
Content	Happy	Joyous	Surprised
			
Disgusted	Terrified	Scared	Worried
			
Sad	Distraught	Suspicious	In-Pain
			

5. Character Properties

Parts

Character Parts are divided into three sections, **Head**, **Upper Body** and **Lower Body**. Initially, character parts are set via the **Parts** presets, they can be further edited individually in the **Parts Tab.**

Part Sections

Head section parts consist of - Head, Hair, Eyebrows, Eyes, Ears, Facial Hair, Nose, Teeth, Tongue, Head Attachment and Face Attachment.
Upper Body section parts consist of - Torso, Arms Upper, Arms Lower, Hands, Back Attachment, Shoulder Attachments, Elbow Attachments and a wrap.
Lower Body section parts consist of - Hips, Legs, Feet, Hip Attachments (Front, Left/Right/Back) and Knee Attachments.

Multiple parts

In some situations, some parts can be replaced by one part i.e. if the character has a pirate leg it will not need a foot or if a character had an apocalypse weapon on its arm, it would not need a hand. In the current version of the tool, you will need to control these parts yourself.

Above - An example of a Leg and a Foot (two parts) being replaced by a peg leg (one part)

Custom Parts

If you wish to modify parts or create your own, do it at your own risk, there is a lot of complexity to this product and any custom content that is not setup correctly could break the tool, we cannot and will not debug custom content.

Body Types / Blendshapes

Body Types are achieved via blendshapes on parts and are divided into three sections, **Gender, Body Type** and **Musculature**. Initially, body types are set via the **Body Type** presets, they can be further edited in the **Body Tab.**

Body Type Sections

Gender - Blends between the **Masculine** Base and the **Feminine** Blendshape
Body Type - Blends between the **Slim** and **Heavy** Blendshape
Musculature - Blends between the **Lean** and **Muscular** Blendshape

Note : Body Blendshapes are baked down to the skinned mesh renderer when the character prefab is created for optimization

Colors

Colors are divided into four sections, **Species**, **Outfit**, **Attachments** and **Materials**. Initially, colors are set via the color presets for the above sections, they can be further edited individually in the **Color Tab.**

Color Sections

Species section colors consist of - Skin, Scar, Lips, Ears, Eyelids, Fingernails, Toenails, Nose, Mouth, Teeth, Gums, Tongue, Eye Outer, Eye Inner, Eye Color, Eye Highlight, Eyebrows, Hair, Head Stubble, Facial Hair, Facial Stubble, Flesh and Horns.
Outfit section colors consist of - Torso Outfit, Arm Outfit, Hand Outfit, Hips Outfit, Leg Outfit, Feet Outfit, Hair Accessory, Facial Hair Accessory and Teeth Accessory
Attachments section colors consist of - Head Attachment, Face Attachment, Shoulder Attachments, Elbow Attachments, Knee Attachments, Back Attachment and Hip Attachments
Materials section colors consist of - Metals, Jewellery Metals, Jewellery Gems, Glass, Glows, Transparency FXs, Digital Screens, Rubber, Plastic, Bone, Fur, Rope, Straps, Stitches, Bandages, Papers, Wood, Feathers and Concrete.

6. Runtime Character Creation

- Shared editor/runtime functionality moved into a single API like class.
- Demo scenes added to give examples on how to interact with the code.

Core Components

`SidekickRuntime.cs` contains all of the runtime functionality (Assets/Synty/SidekickCharacters/Scripts/Runtime/API). This is an instance class that tracks variables needed to create the Sidekick character.
To instantiate this class, you need 2 required parameters, and can include 2 optional parameters. The 4 parameters are:

- `model` - The base model to use for the rig and allow parts to be added and removed from.
- `material` - The material to use for the completed character. This material will be updated with the colors (if desired) and applied to the character created.
- `animationController` - (Optional) If you wish to apply an animation controller to the created character on creation, you can add it in here.
- `dbManager` - (Optional) The database Manager to use to access the database, if none is supplied then a new one will be created.

Sidekick DTOs (Data Transfer Objects) are used to get information out of the database. This information is used in supplying presets, color sets and (in a coming update) parts. These allow for fast and easy setup of a character creator at runtime.

Sidekick Runtime Demo

There are 3 demo scenes that contain sample code on the core components of the character creator and how they may function in a runtime character creator. There is a demo for presets, for parts, and for colors.

The code in the demo scripts provides examples that can be expanded on to create all that is needed for a runtime character creator.

The scripts are intended as examples only, and are not fully exhaustive in their implementations.

Basic Character Creation Workflow

First a list of parts to combine needs to be obtained. The list is made up of Skinned Meshes from the selected parts.
Once a list of parts is obtained, they can then be passed through the Sidekick runtime API.

This is done by calling `CreateCharacter`. If you want a model with a single mesh, call `CreateCharacter` with the parameter `combineMesh` set to `true`, otherwise to keep the parts

separate set it to `false`. The fourth parameter should almost always be set to `true` as it adjusts the positioning of attachment joints based on the body blend shape settings.

The GameObject that is returned is a fully complete skinned character that is ready for use.

The combining process is fairly heavy, so it is not recommended to use this during gameplay as it may cause noticeable hangs.

7. Naming conventions

A_ Animation
AC_ Animation Controller
M_ Material
PF_ Prefab
PM_ Physics Material
SK_ Skeletal Mesh
SM_ Static Mesh
T_ Texture

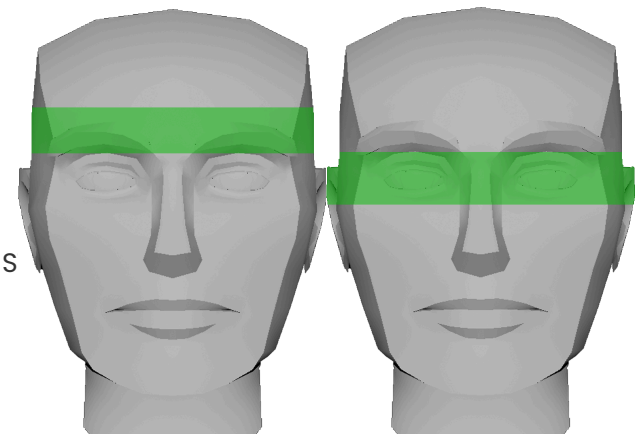
Scripts are contained in the **Synty.SidekickCharacters** namespace.

8. Animation

Facial Blendshapes

Sidekick characters have the option of facial blendshapes to drive facial expressions, lip sync, or even use facial expressions to further customize a character.
The facial system follows the ARKit Blendshapes system so that users can then take their characters and set them up for AR with facial capture.
Because our system uses so many parts we have multiple meshes with blends with the same name, because they are part of the same blendshape pose but are applied to different meshes. This can make it difficult to hand modify blends one at a time since you potentially need to adjust up to 6 meshes blendshapes for a single pose.
The comprehensive list of facial blendshapes targets are as follows:

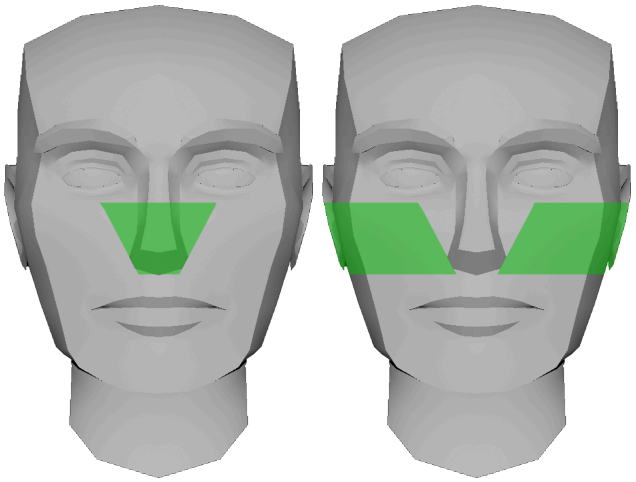
browFrownLeft
browFrownRight
browInnerDownLeft
browInnerDownRight
browInnerUpLeft



eyeBlinkLowerLeft
eyeBlinkLowerRight
eyeBlinkUpperLeft
eyeBlinkUpperRight
eyeSquintLeft

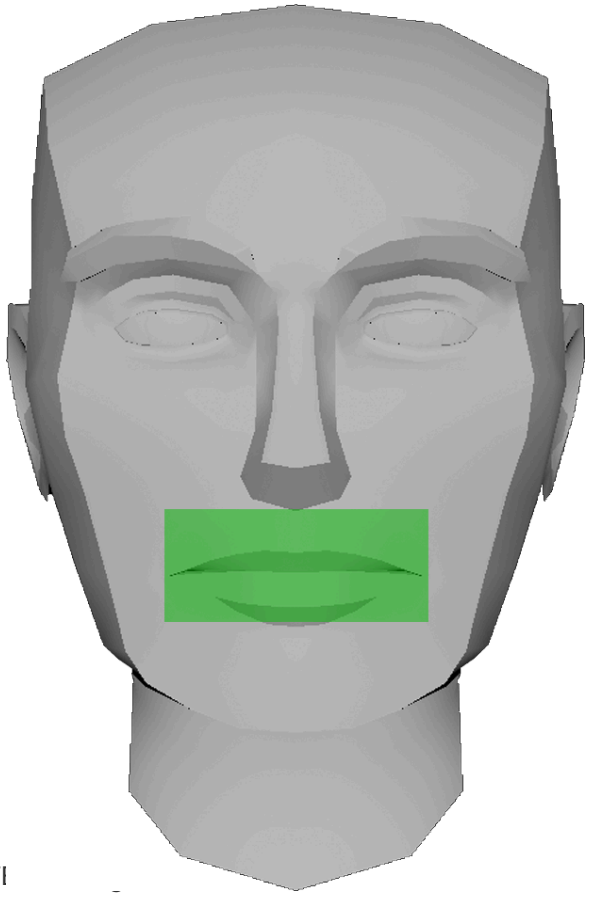
browInnerUpRight eyeSquintRight
browOuterDownLeft eyeWideLowerLeft
browOuterDownRight eyeWideLowerRight
browOuterUpLeft eyeWideUpperLeft
browOuterUpRight eyeWideUpperRight
browRaiseLeft browRaiseRight

noseSneerLeft
noseSneerRight



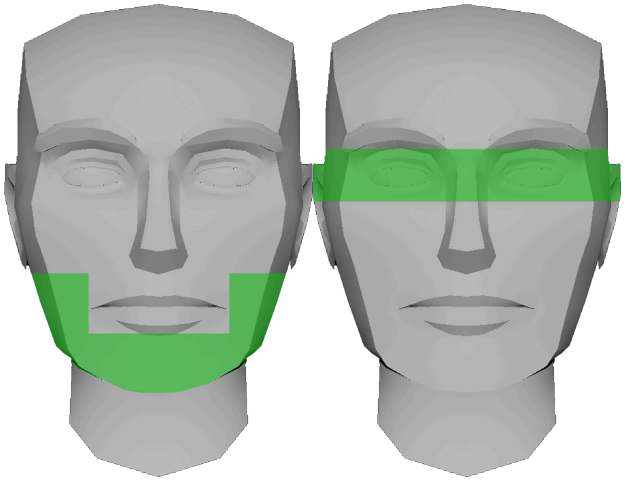
cheekHollowLeft
cheekHollowRight
cheekPuffLeft
cheekPuffRight
cheekSquintLeft
cheekSquintRight

mouthClose
mouthDimpleLeft
mouthDimpleRight
mouthFrownLeft
mouthFrownRight
mouthFunnel
mouthLowerDownLeft
mouthLowerDownRight
mouthPressLeft
mouthPressRight
mouthPucker
mouthRight
mouthRollLower
mouthRollOutLower
mouthRollOutUpper
mouthRollUpper
mouthShrugLower
mouthShrugUpper
mouthSmileLeft



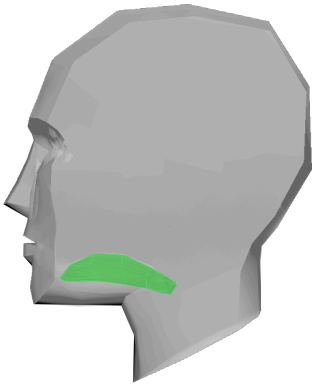
mouthSmileRight
mouthStretchLeft
mouthStretchRight
mouthUpperUpLeft
mouthUpperUpRight

jawBackward
jawForward
jawLeft
jawOpen
jawRight
mouthLeft



eyeLookDownLeft
eyeLookDownRight
eyeLookInLeft
eyeLookInRight
eyeLookOutLeft
eyeLookOutRight
eyeLookUpLeft
eyeLookUpRight

tongueCurlDown
tongueSideCurlDown
tongueCurlLeft
tongueCurlRight
tongueCurlUp
tongueDown
tongueLower
tongueOut



tongueSideCurlUp
tongueTwistLeft
tongueTwistRight
tongueUp
tongueIn
tongueRaise

Blendshape naming across meshes is consistent

- ***Ease of use**: This will allow developers to access facial shapes through code alot easier as each shape is named the same, the main difference when the meshes are uncombined is the prefix names of the Blendshape node on each mesh. The potential meshes with blendshapes and the node names are as follows*
    - *Head - **HEADBlends***
    - *Eyebrow Left - **EBRLBlends***
    - *Eyebrow Right - **EBRRBlends***
    - *Nose - **NOSEBlends***
    - *Teeth - **TETHBlends***

- *Tongue - **TONGBlends***
  - *Facial Hair - **FCHRBlends***
  - *Head Attach - **AHEDBlends***
  - *Face Attach - **AFACBlends***

Example: On a combined mesh the jaw is opened.
Jaw open may be needed across the head, teeth, tongue, facial hair and head attachment to hit the pose with no clipping parts. The attributes needed changing would be
HEADBlends.jawOpen
TETHBlends.jawOpen
TONGBlends.jawOpen
FCHRBlends.jawOpen
AHEDBlends.jawOpen
In each case the '**Blends.jawOpen**' is an identical string, and the 4 letter prefix is the separate identifier.

Mecanim

By default the Sidekick characters are using Unity's Mecanim humanoid system to keep broad usability with animations. When a prefab character is generated the avatar it uses is stored with it and named to match what the prefab's name is. However all the parts are also mecanim ready before being combined through the tool, and are using the **SK_BaseModelAvatar** so that the avatar that is generated off that will be correct whenever a prefab is made.

Facial Animation

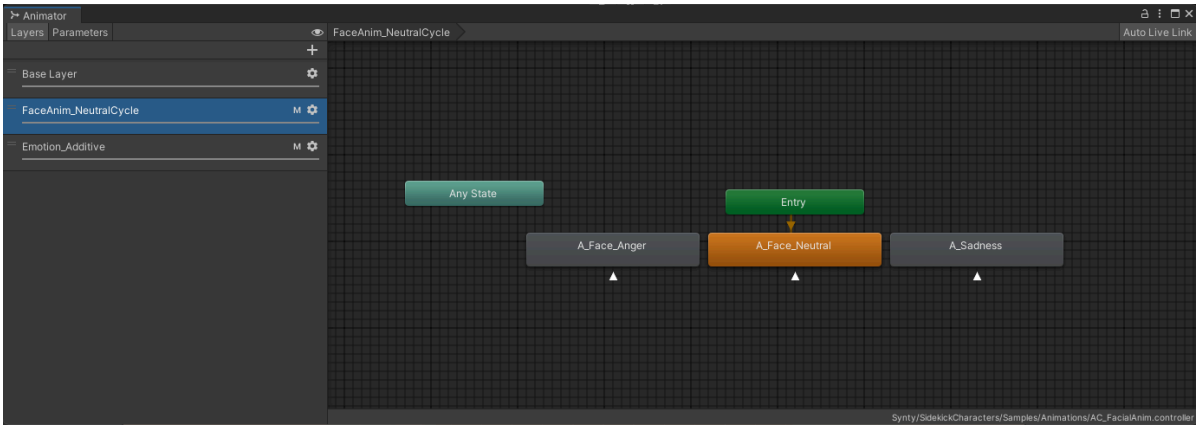
The facial animation system is primarily using blendshapes to add motion to the facial features. Users can use the **eye_l**, the **eye_r** and jaw joints to do simple facial movements if their designs call for that instead. Otherwise the system implemented works off similar principles to the FACS or facial action coding system, bit is more directly based on the ARKit (https://arkit-face-blendshapes.com/)

The sample provided has a prefab character made with the Character Creator, driven by a sample animator called **AC_FacialAnim**, with 3 layers, one for the body and two for the face. These are the **FaceAnim_NeutralCycle** and **Emotion_Additive**.

FaceAnim_NeutralCycle Layer

This Layer is an Override layer with a mask so that it only affects the blendshapes on a character. It comes with three animations that can be swapped for exaggerating poses but by default the Neutral is often what you would use to drive the default pose of the face

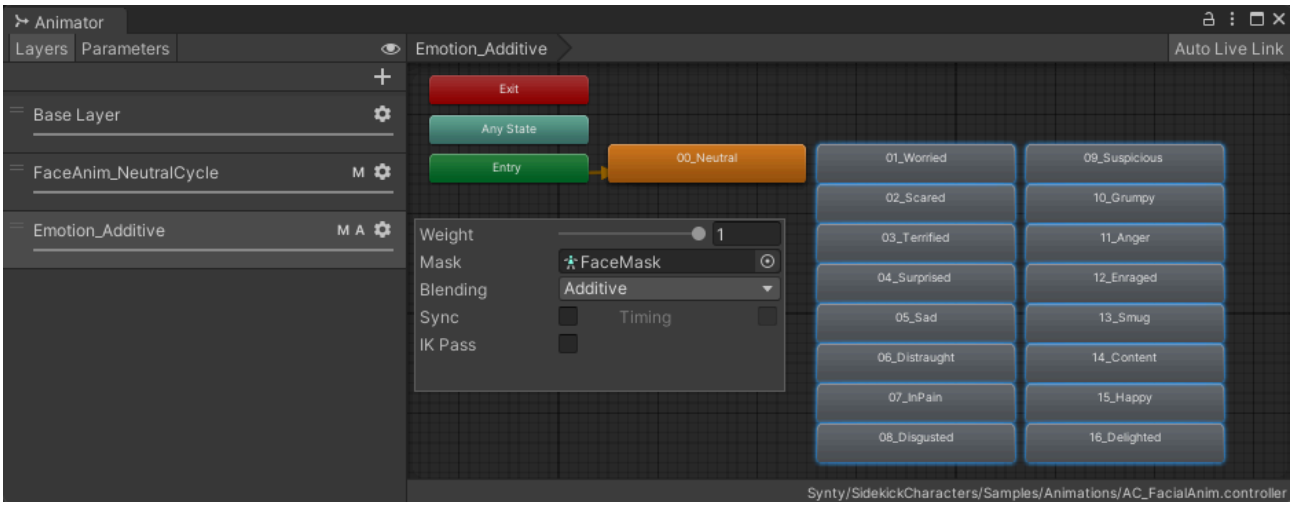
- The animation is a loop of a neutral movement, a small amount of jaw and blinking.
- A User could create a new facial pose to modify the shape, for example puffing the cheeks and closing one eye with the blinks could add personality to the character as a facial expression layered onto the character in this layer.



Emotion_Additive Layer

This Layer is an Additive layer with a mask so that it only affects the blendshapes on a character. It comes with 16 animations that can be iterated through when a User plays the scene to see the different poses on the face

- Each animation is a static pose that is applied on top of the Neutral Cycle
- If a User presses **Next Expression** the facial pose will blend to the next in the list of animations as an example



Users Expand on this system to implement their own facial system for reactions, poses etc, and implement this alongside body motion in a single animator, rather than as a separate animator like this sample.

Constraints - Attachment Meshes

These characters have attachment joints on the elbows, back, hips, and knees for attaching meshes or equipment in the creator tool. Users can customize joint behavior to fit their needs. For instance, as a child of the lower arm joint the elbow joint typically follows the elbow at 100% influence, but this can be adjusted by adding a constraint component to modify its movement.

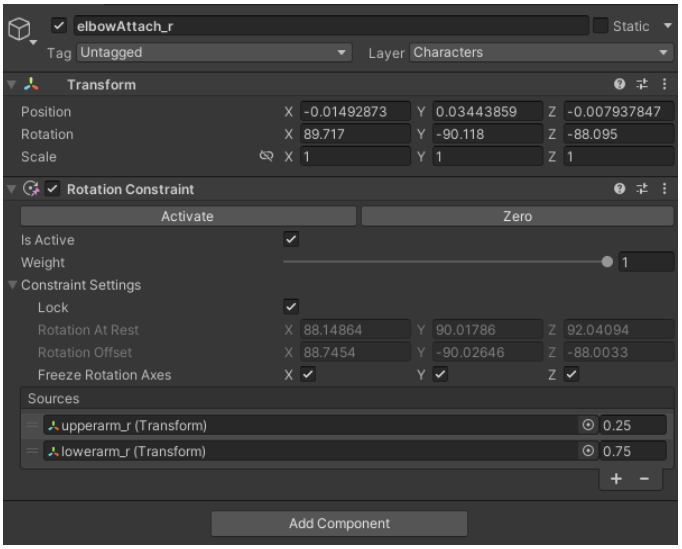
Within your generated character prefab you can navigate to the joint you want to drive with a constraint and add one or a combination of the following constraints

- **Aim Constraint**
- **Look At Constraint**
- **Parent Constraint**
- **Position Constraint**
- **Rotation Constraint**

- **Scale Constraint**

Aim Constraint

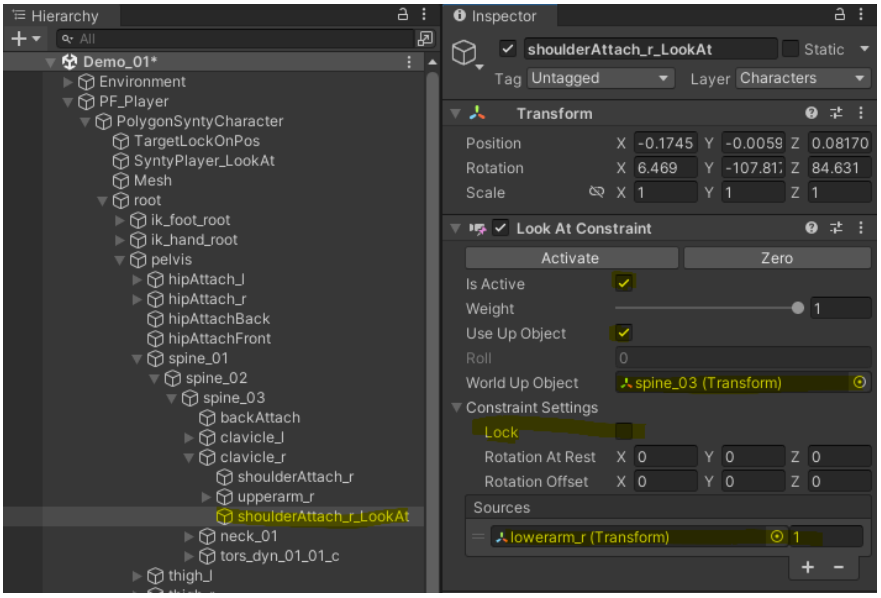
For example, a user would like an elbow attachment to only follow the rotation of the lower arm by half the intensity of its default movement (the 100% from the elbow mentioned above)



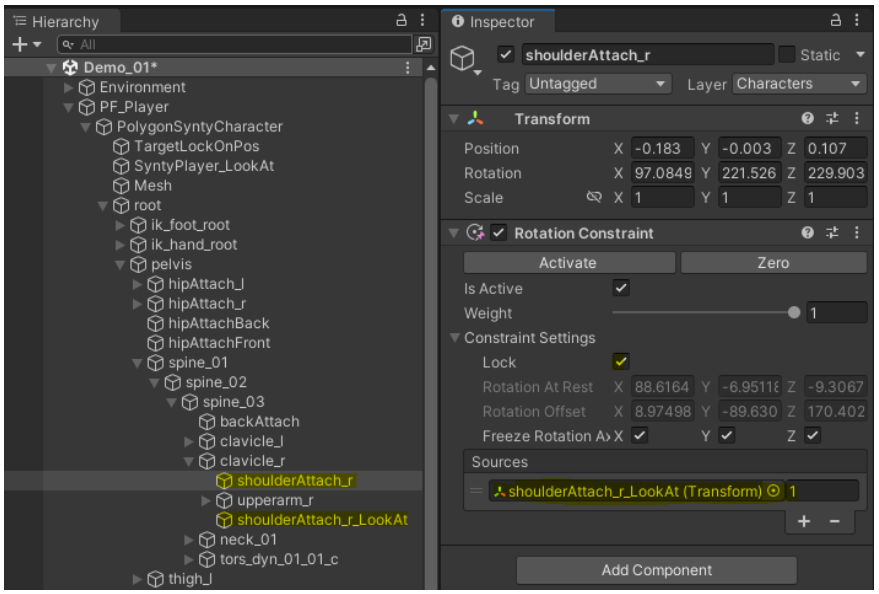
1. Select the joint you want to drive
2. add **Aim Constraint** component
3. In constraint settings, **Lock** the transform values first so that you don't include offsets by default (unless that is the intention)
4. In **Sources**, click the **+** to add windows that driver joints can be dragged in.
5. treat 1.0 like 100%, divide up and apply the influences across the sources to find the movement that is preferred on the driven joint in motion. (each **source** can also be set to 1 and exceed 100% across all influences

Look At Constraint

For example, a user would like a shoulder attachment to follow the rotation of the upper arm so that it doesn't clip when the arm lifts.



1. create an **Empty** under the Skinned joint you want to drive with the **Look At Constraint** (in this case, the **shoulderAttach_r**) to store its location on the Empty, and then reparent it under the targets parent (in this case, the clavicle_r) and rename it to indicate its purpose
2. Add the **Look At Constraint** to this **Empty**, without Locking the constraint so that it reorients to look at the target, you need to set a target that does not occupy the same world space as the object with the **Look At Constraint,** this can cause problems with how it solves the orientation. In this case the Empty is set to look at the **lowerarm_r** joint.
3. Tick on **Use Up Object** and set this to use a joint that would dictate the world up, often the **spine_03** (top of the spine) or **pelvis** (bottom of the spine). This is to control the behaviour correctly when a character is upside down.
4. On the skinned joint, add a **Rotation Constraint** with the Constraint Settings locked to maintain the offset between the Skinned joint, and add a Source that is driven by the Empty



5. In this case the User should now see when they rotate the **upperarm_r** that the **shoulderAttach_r** moves with the arm when it rotates up and down, but because the target isn't moving when the arm twists, the **shoulderAttach_r** doesn't twist, and moves correctly. A User could even translate the **shoulderAttach_r** up in world space and the joint will continue to rotate to look at the lower arm, while maintaining the offset.

Parent Constraint

The **Parent Constraint** links the Position, Rotation and Scale of a **GameObject** to one or more targets.

1. Add **Parent Constraint** component to object
2. click the '+' button next to the **sources** list
3. Drag and drop the objects you want to be the parents into the added source rows
4. Adjust the weight of each source to control their influence on the target object.
5. Check on **Freeze** on the channels you want to keep locked and not influenced by the source, and tick on **Lock** if you want to maintain the offset.
6. Tick on **Is Active**

Position Constraint

The **Position Constraint** works the same as the parent constraint, however it only works to link the *Position* of a **GameObject** to one or more targets

Rotation Constraint

The **Rotation Constraint** works the same as the parent constraint, however it only works to link the *Rotation* of a **GameObject** to one or more targets

Scale Constraint

The **Rotation Constraint** works the same as the parent constraint, however it only works to link the *Scale* of a **GameObject** to one or more targets

Dynamic Joints

In many of the parts there are additional joints outside of the humanoid skeleton that are added that can be used to drive tertiary motion through physics. Each bone is named following a specific convention to avoid naming clashes across assets.

*"mesh"*_**dyn**_*"chain#"*_*"#-in-chain"*_*"parent-joint-#-in-fork"*_*"side"*

_ *- underscore divider*
mesh *- name of the mesh the dynamic joint is skinned to*
dyn *- string in name to identify this is a dynamic joint to make searching easier*
chain# *- chain number that join belongs to*
#-in-chain *- identifier of how far down the hierarchy in the chain this joint is*
parent-joint-#-in-fork *- if two dynamic joints are parented to a single parent dynamic joint, a*
side *- **l**eft, **r**ight, **c**entre, bac**k** to identify what side of the body the joints are on.*

Example: **ashl_dyn_01_02_l** Is a shoulder attachment left joint, on the left side of the character. It is in the first chain, and it is the second joint in that chain.
Example 2 **: hips_dyn_01_02_01_k** is a hips attachment joint on the back of the character. It is in the first chain, and it is the second joint in that chain, where it has forked off from the first joint in the chain (the parent for this example would be name **hips_dyn_01_01_k**)

Users can then attach dynamics solvers to these joints to add secondary motion onto their characters using a variety of different dynamics solutions available.

9. Third party assets

These are some potential third party assets that may be useful in your endeavors to create the best characters for your games/individual needs

Dynamics solutions

uniVRM - Springbone

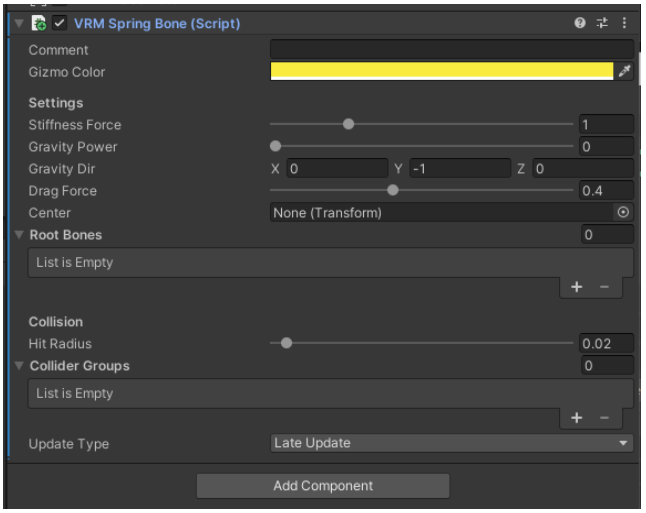
UniVRM is free to use, open-source project, however, while UniVRM itself is free and open-source, you should always check the specific licenses of any models, assets, or other software you use in conjunction with it, as they may have their own usage restrictions or fees.

This adds extra secondary movement to the motion applied to a character to give it more life, this Method does not require you to make a VRM model but you can utilize the free Springbones scripts it has for tracking just within Unity

uniVRM: https://github.com/vrm-c/UniVRM/releases/tag/v0.116.0
Video Resources: https://youtu.be/i2pOourRdFU?t=362

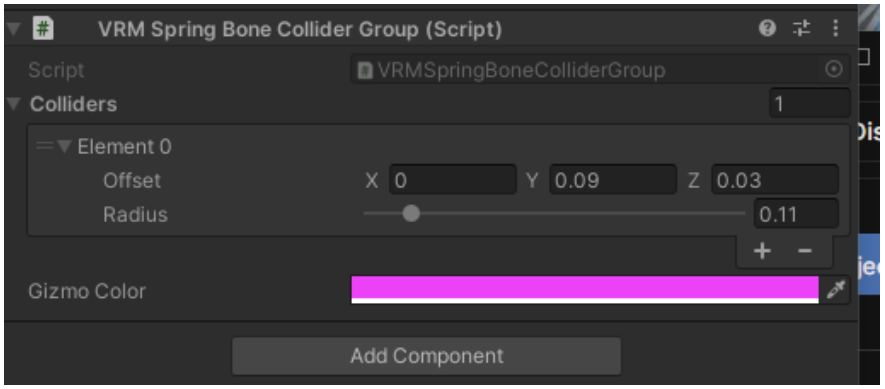
Steps

1. *Add uniVRM package*
2. *Add a 'VRM Spring Bone' component to the parent of the character*
    a. *Ignore this step if creating an actual VRM model, this is just for use in Unity*

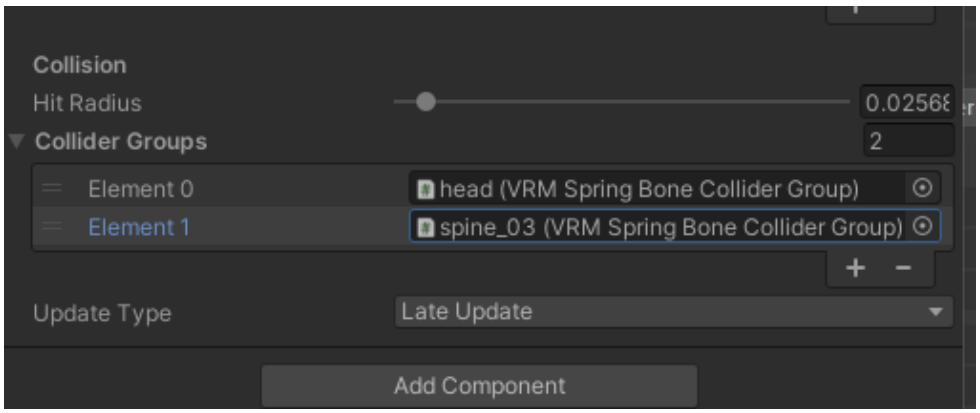


3. *Add the joints you wish to have physics to the root bones and you should see them appear on your character*
    a. *Note: You can add multiple Spring bones per type of material you want to move with different forces.*
4. *You should see them sway around if you hit Run and move the character around, but the issue now is that there is no collisions setup*

5. *You want to add 'VRM Spring Bone Colliders' to the base joints the spring bones will collide with so that the colliders will follow the body as they move. (if you add them to the root node they will be static).*
   In this case I have added them to the spine_03 and the head joints and they show up as colored spheres in the scene



6. *Once setup, return to the main VRM Spring bone component and add them to the collider groups.*



7. *Some adjustments may need to be made, with the hit radius but this should fix most issues.*

10. Terms of use

For all the terms and conditions of the license, please read the full EULA available at https://syntystore.com/. By using this Asset, you agree to be bound by the terms of the EULA.

11. Glossary

Term	Definition & Context
Interpenetration	to penetrate between, within, or throughout. In this context, two meshes clipping through one another
Chain	In terms of joints, this is used to define the hierarchy of the joints to separate naming and remove clashing
Blendshape Target	a specific deformed version of a mesh, each target represents a particular shape that the mesh can morph into
Blendshape node	The parent component that houses multiple Blendshape Targets and manages the interpolation between the original shape and one or more of the targets