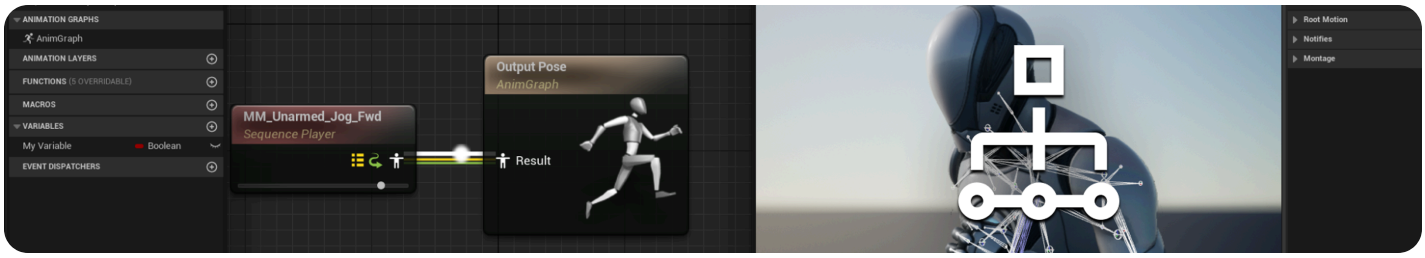
Animation Blueprint Linking

Modularize your Animation Blueprint logic by using Animation Blueprint Linking and Templates.



As you start to create more and more complex Animation Blueprints for your characters, there may be times when you want to reuse portions of an Animation Blueprint within another Animation Blueprint. There are a variety of ways you can do this, from linking specific Animation Blueprints, linking Animation Layers, or using Templates.

This document provides an overview of modularizing your Animation Blueprints in various ways.

Prerequisites

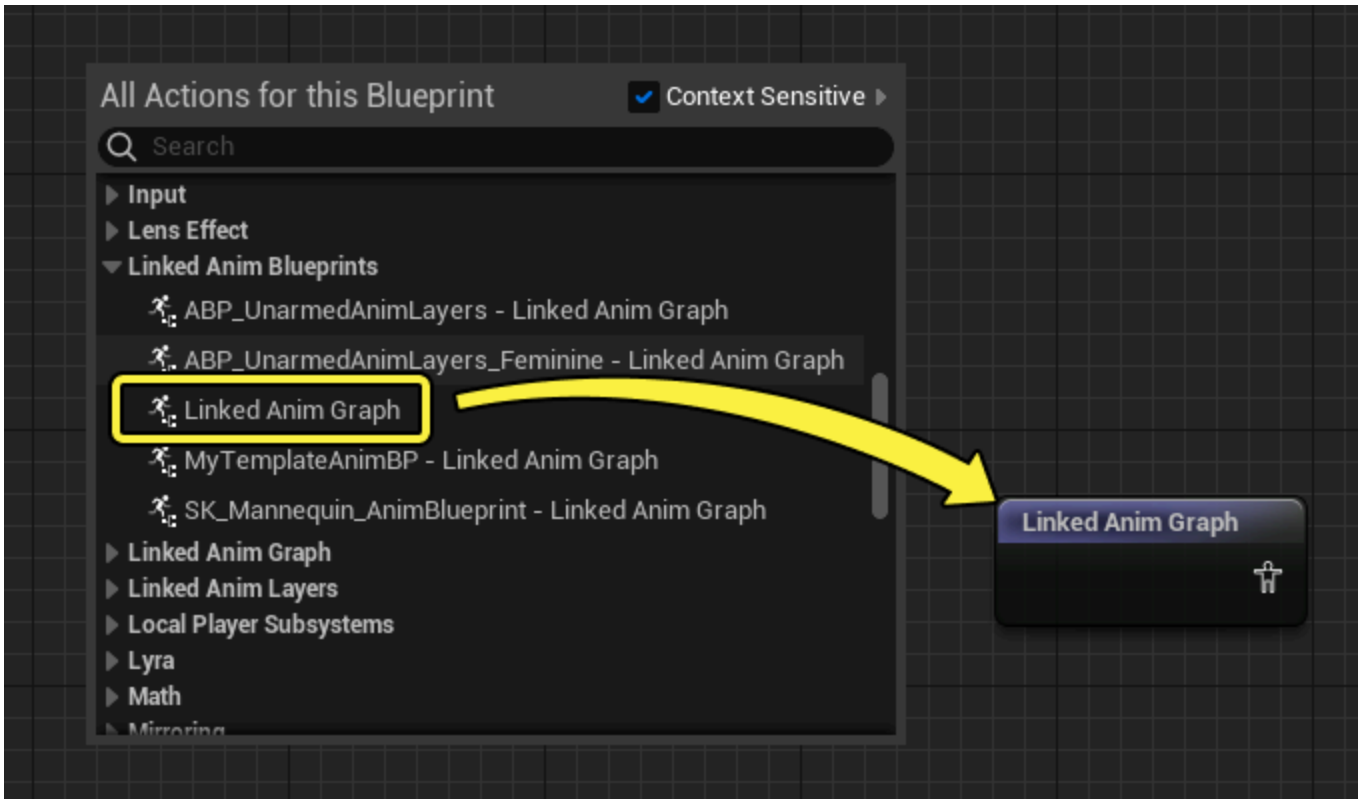
- You have created and have an understanding of Animation Blueprints.

Linked Anim Graph

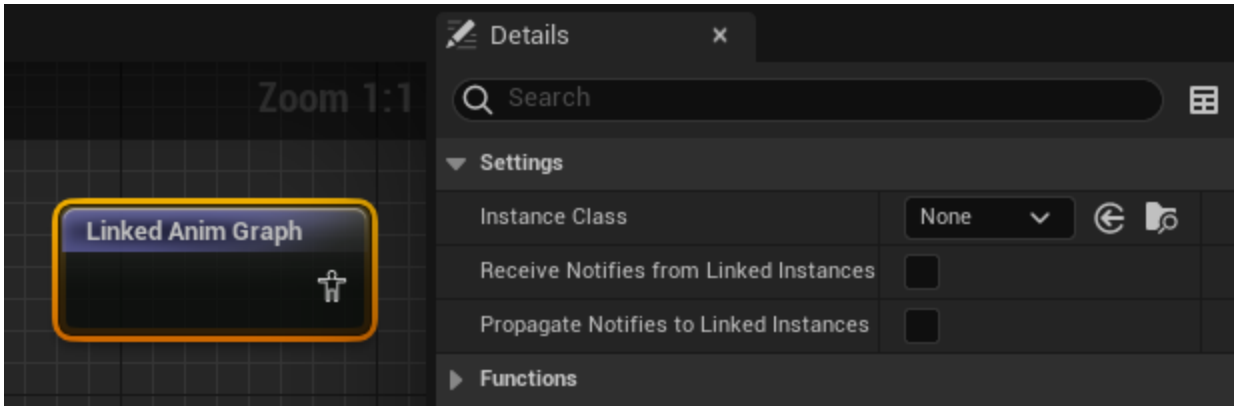
Within an Animation Blueprint, you can reference another one by using the **Linked Anim Graph** node.

Creation and Overview

To create this node, right-click in the **Anim Graph** and select **Linked Anim Graph** from the **Linked Anim Blueprints** category. You can also select a specific Animation Blueprint reference from this list to add the node and reference it.



The node contains the following properties in the **Details** panel. These settings also apply for **Linked Anim Layer** nodes:



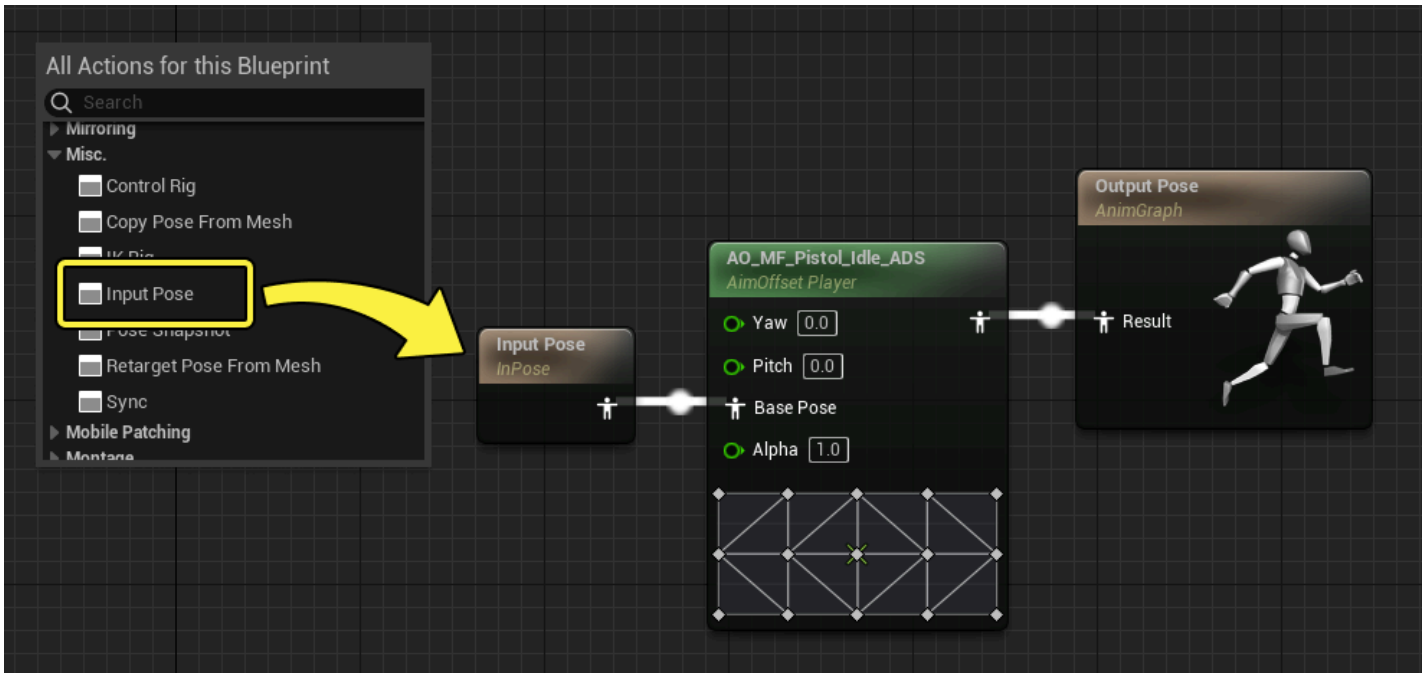
Name	Description
Instance Class	The Animation Blueprint class to use for this Linked Anim graph.

Name	Description
Receive Notifies from Linked Instances	Whether Skeleton Animation Notifies will be received by this linked instance from other Linked Anims. This can be useful in situations where you want to encapsulate Animation Notify handling in one Linked Anim instance, while playback can happen in another.
Propagate Notifies to Linked Instances	Whether Skeleton Animation Notifies will be sent from this linked instance to other linked instances. **Propagate Notifies to Linked Instances** must also be enabled on the target Linked Anim instance for notifies to successfully propagate.

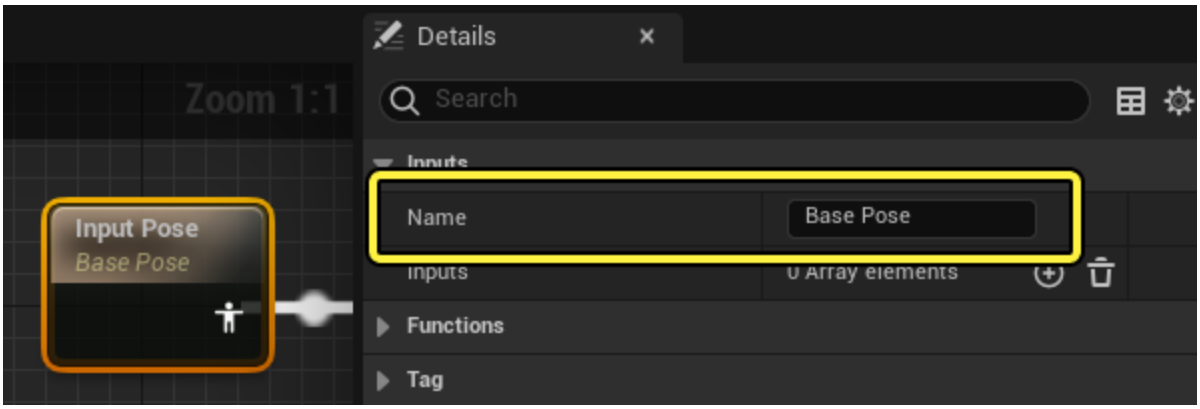
Double-clicking a Linked Anim Graph node will open the linked Animation Blueprint asset, if one is assigned,

Input Pose

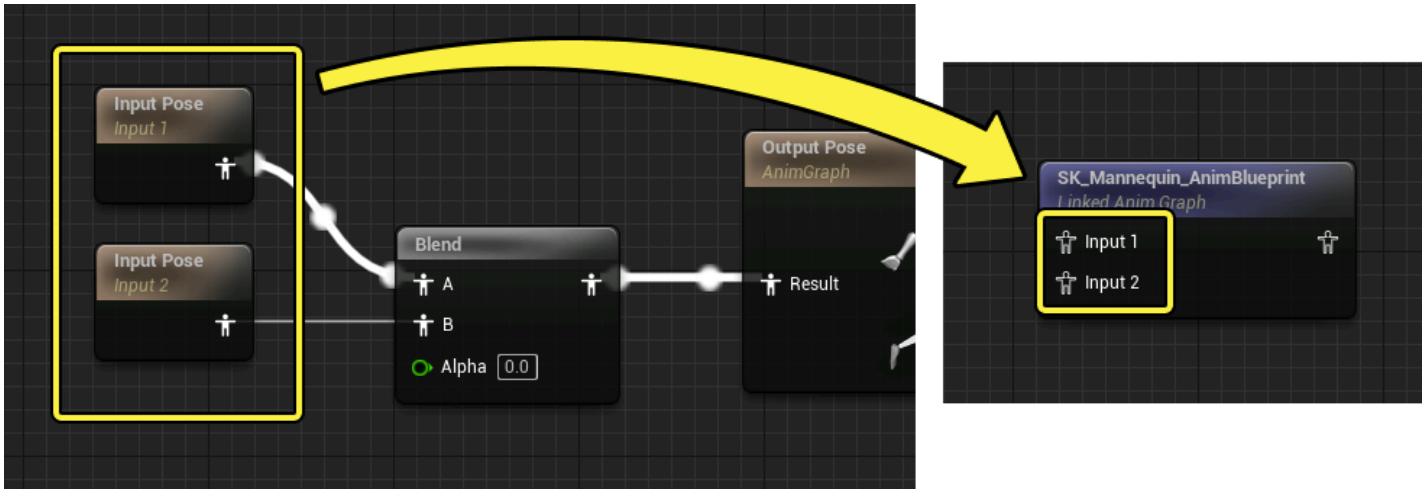
One of the ways you can use your Linked Anim Graph is by exposing animation inputs using the Input Pose node. Create this node by right-clicking in the **Anim Graph** of your linked Blueprint and select **Input Pose** from the **Misc.** category. You can then connect this node to an animation node.



You can optionally rename the Input Node by selecting it and changing the **Name** in the **Details** panel.

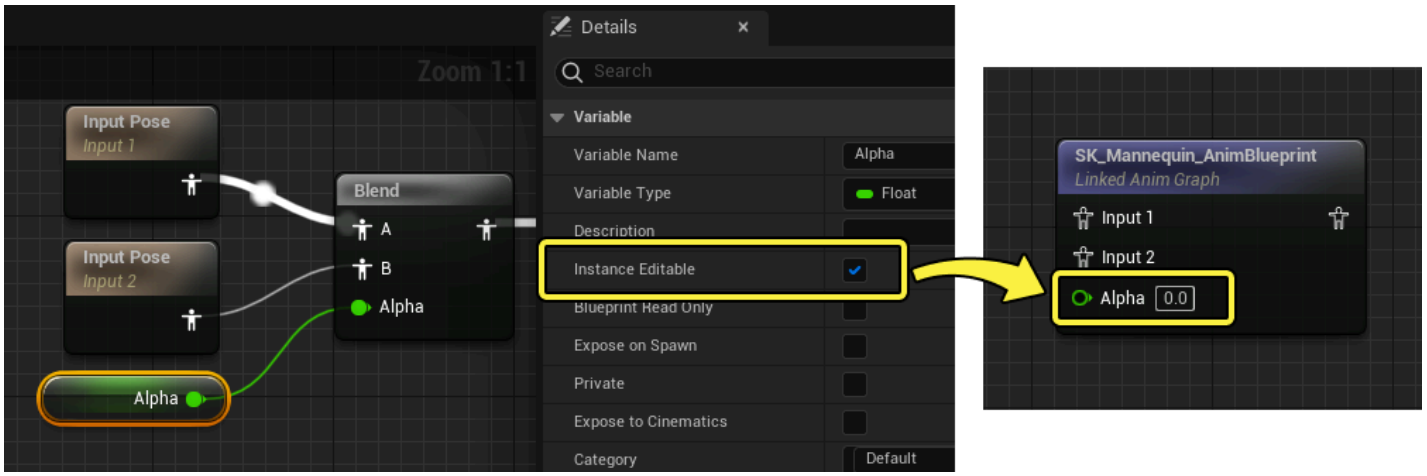


You can create as many Input Poses as needed, and all of them will display on the Linked Anim Graph Node as input pins.



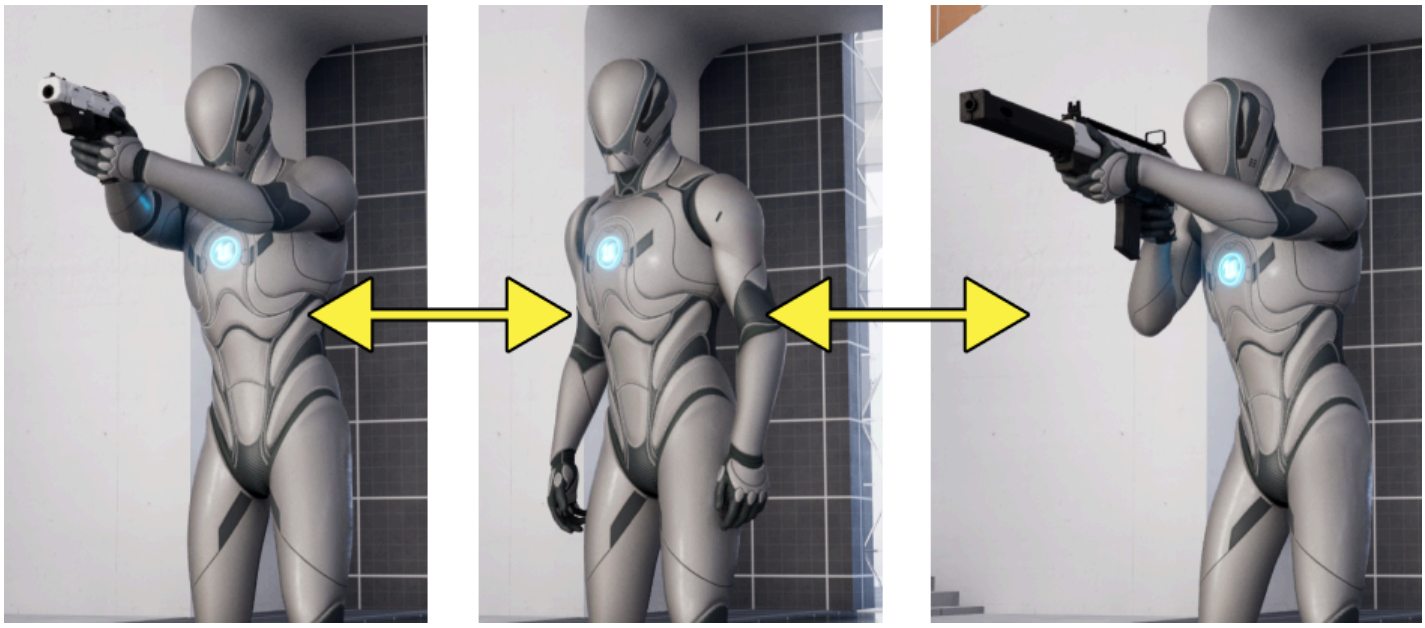
Input Variables

Variables can also be exposed to the Linked Anim Graph, in a process similar to the Blueprint process of making a variable public. Select your variable and enable **Instance Editable**. This will expose the variable on the Linked Anim Graph.



Linked Anim Layer

With Linked Anim Graphs, you link to a specific Animation Blueprint. With **Linked Anim Layers,** you use linking in a more standardized form. Linked Anim Layers require you to create an **Animation Layer Interface**, which defines the preset Animation Layers of your graph. You can then create a primary Animation Blueprint which contains the main logic structure for these different layers, and then different Animation Blueprints for each gameplay differentiator.



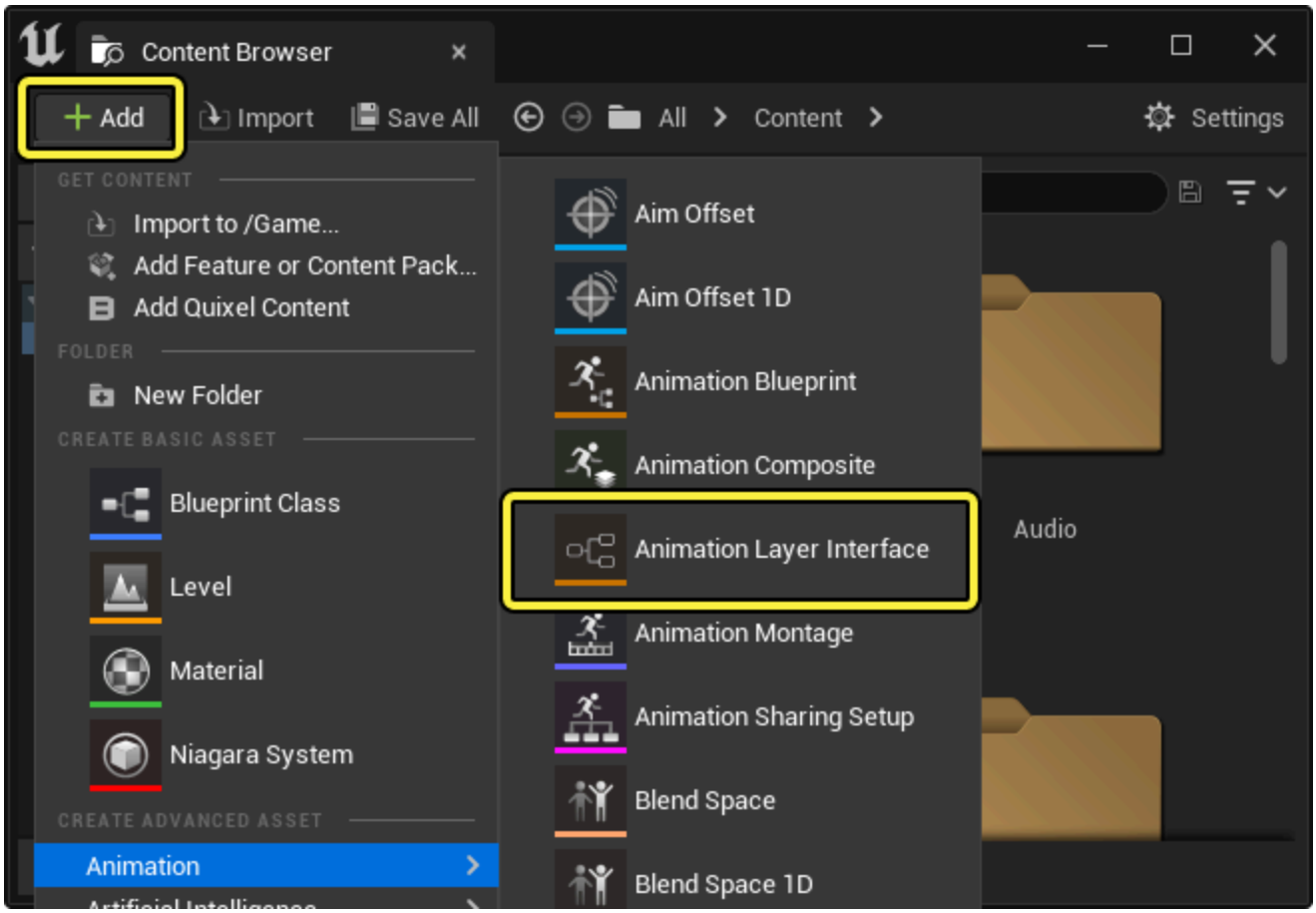
You will typically want to use Linked Anim Layers when you are creating an animation project of high complexity, for example a live-service project where new animations are added and updated. This is due to the workflow centering around creating new, or removing, Animation Blueprints for specific gameplay logic.

Rather than continuously update and manage a single Animation Blueprint, you can separate your logic into these layers. This enables multi-user collaboration and memory savings, as the system will only load the relevant Animation Blueprints if they are being linked and used.

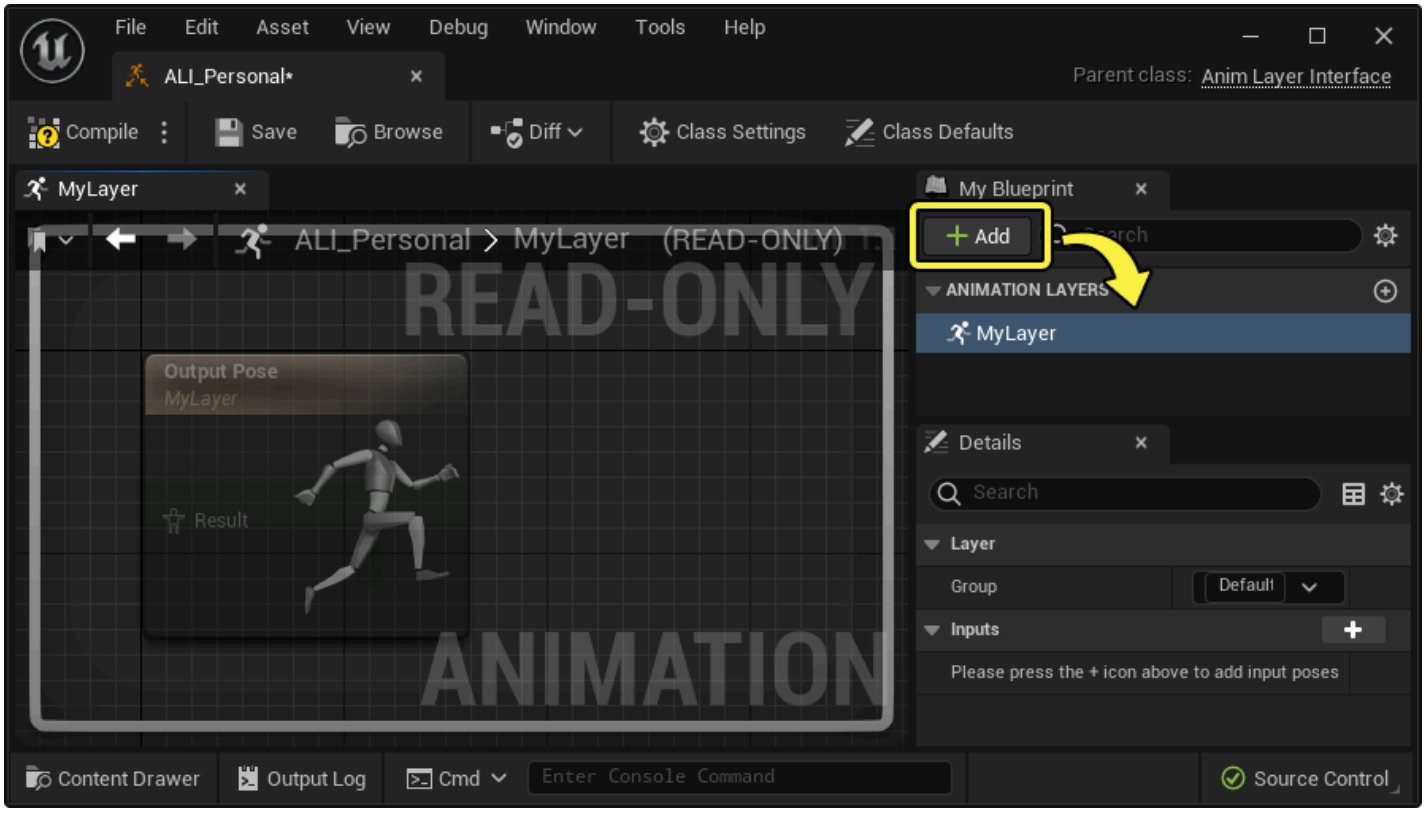
> You can control animation memory with Linked Anim Layers by breaking your Animation Blueprints into multiple asset files. An Animation Blueprint that contains linked layers can be loaded and unloaded when necessary using Unreal Engine's memory management systems. Therefore, you must set up additional streaming logic if you want to remove Animation Blueprints from memory when they are unlinked.

Animation Layer Interface

To create a Linked Anim Layer system, you must create an Animation Layer Interface Asset, where you will define the animation layers to use in your Animation Blueprints. In the **Content Browser**, click **Add (+)**, then select **Animation > Animation Interface**.

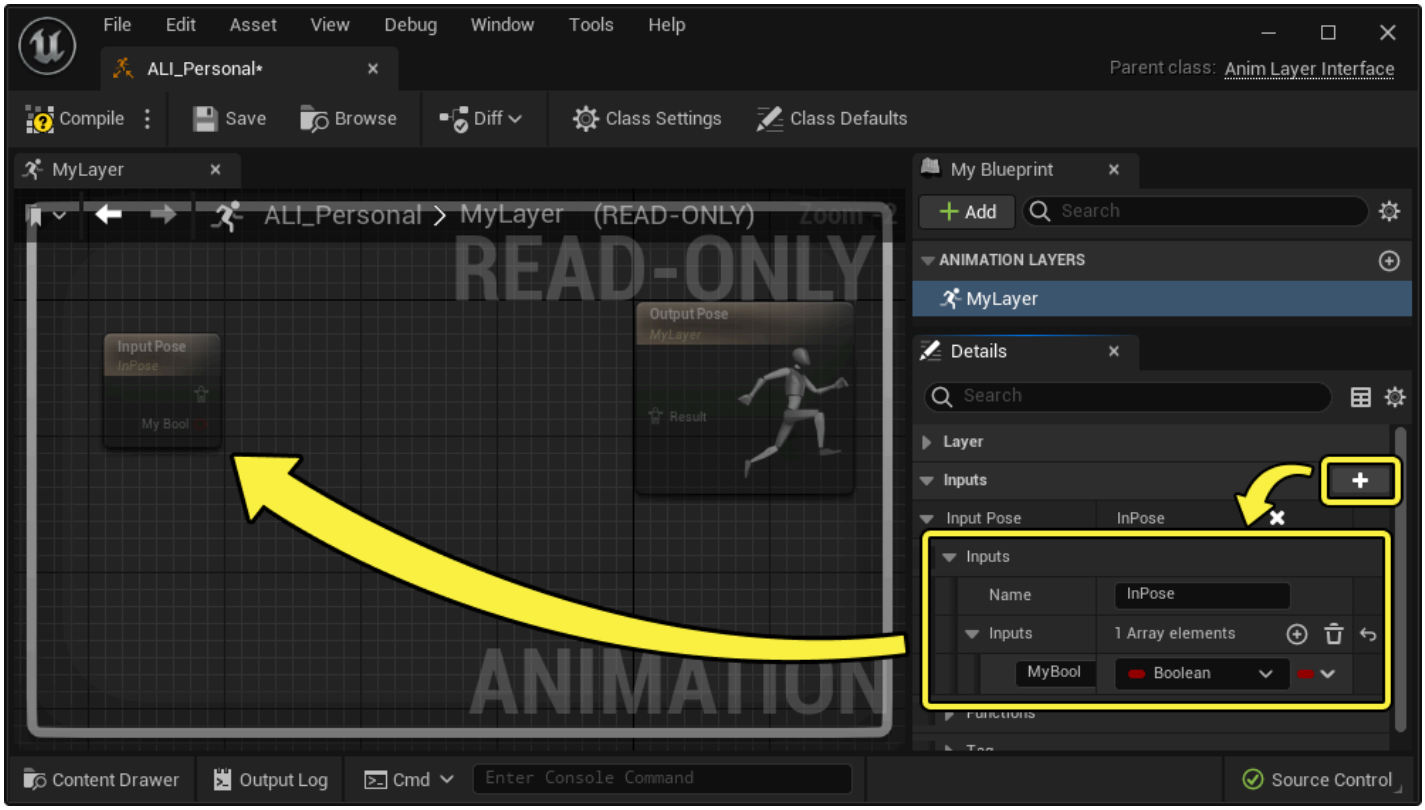


Open the Asset to view the Animation Layer Interface editor. The primary way you will interact with this asset is by adding **Animation Layers**, which can be done by clicking **Add (+)** in the **My Blueprint** panel.



Layers under the same non-default **Group** name will share the same Animation Instance. It is recommended to use a new/non-default shared name for the layers in your Layer Interface.

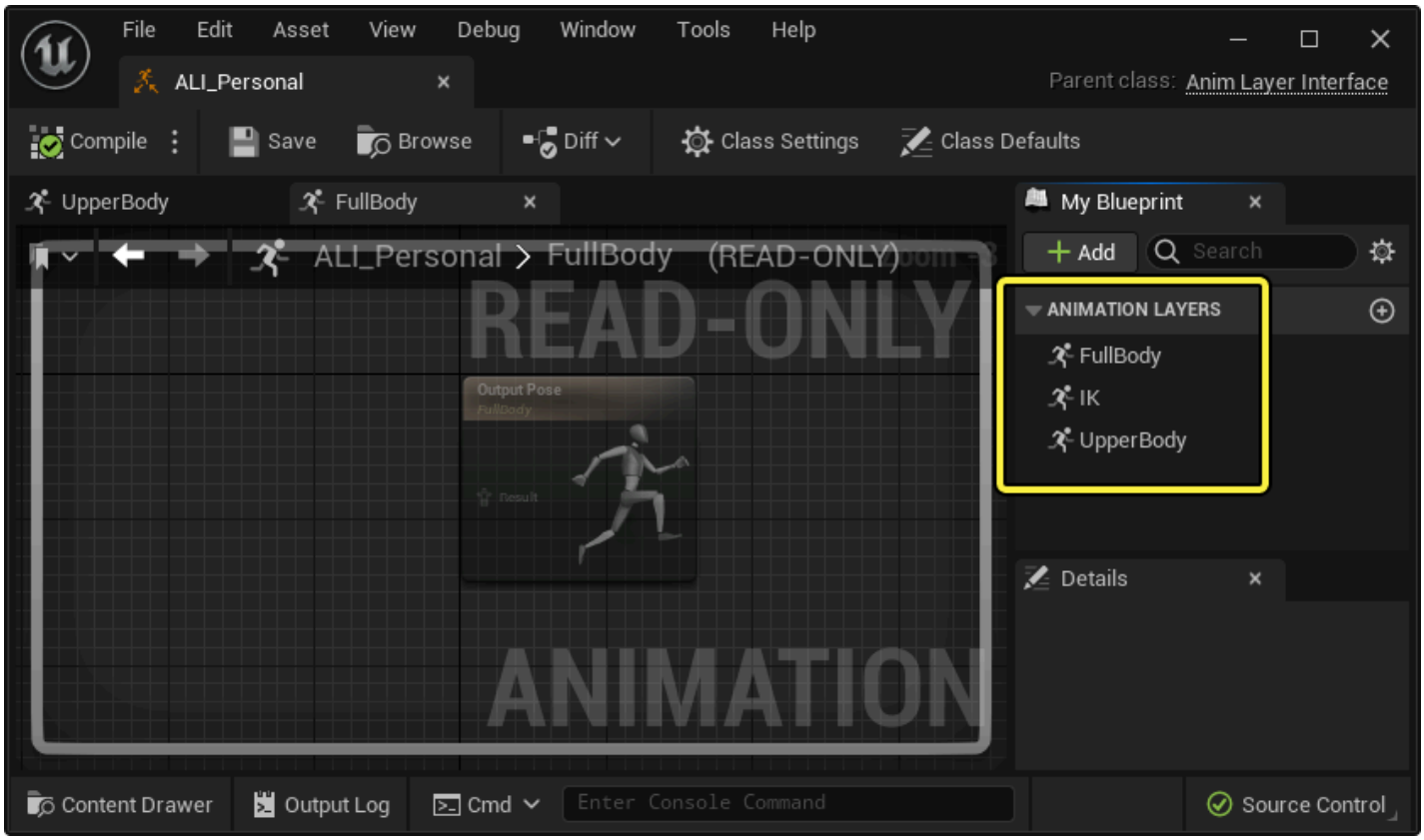
For some layers you may want to also expose pose and property inputs, as this may be necessary for the Linked Anim structure, such as if this layer is to modify an incoming pose. To do this, click **Add (+)** on the **Inputs** section of the selected layer's **Details** panel. You can also add property inputs under the **Inputs** property.



For an example Animation Layer setup, you may want to create a general system of **Full Body**, **Upper Body**, and **IK** layers. These layers provide the following functionality for your character:

- **Full Body**: The overall base locomotion and animation of your character.
- **Upper Body**: If your character holds items or weapons, you could have a layer that contains this logic.
- **IK**: Final arm or leg adjustments on your character. Typically this would be leg IK to adjust foot placement, or arm IK to ensure the hands are holding weapons correctly.

In this case, your Animation Layer Interface may look like this:

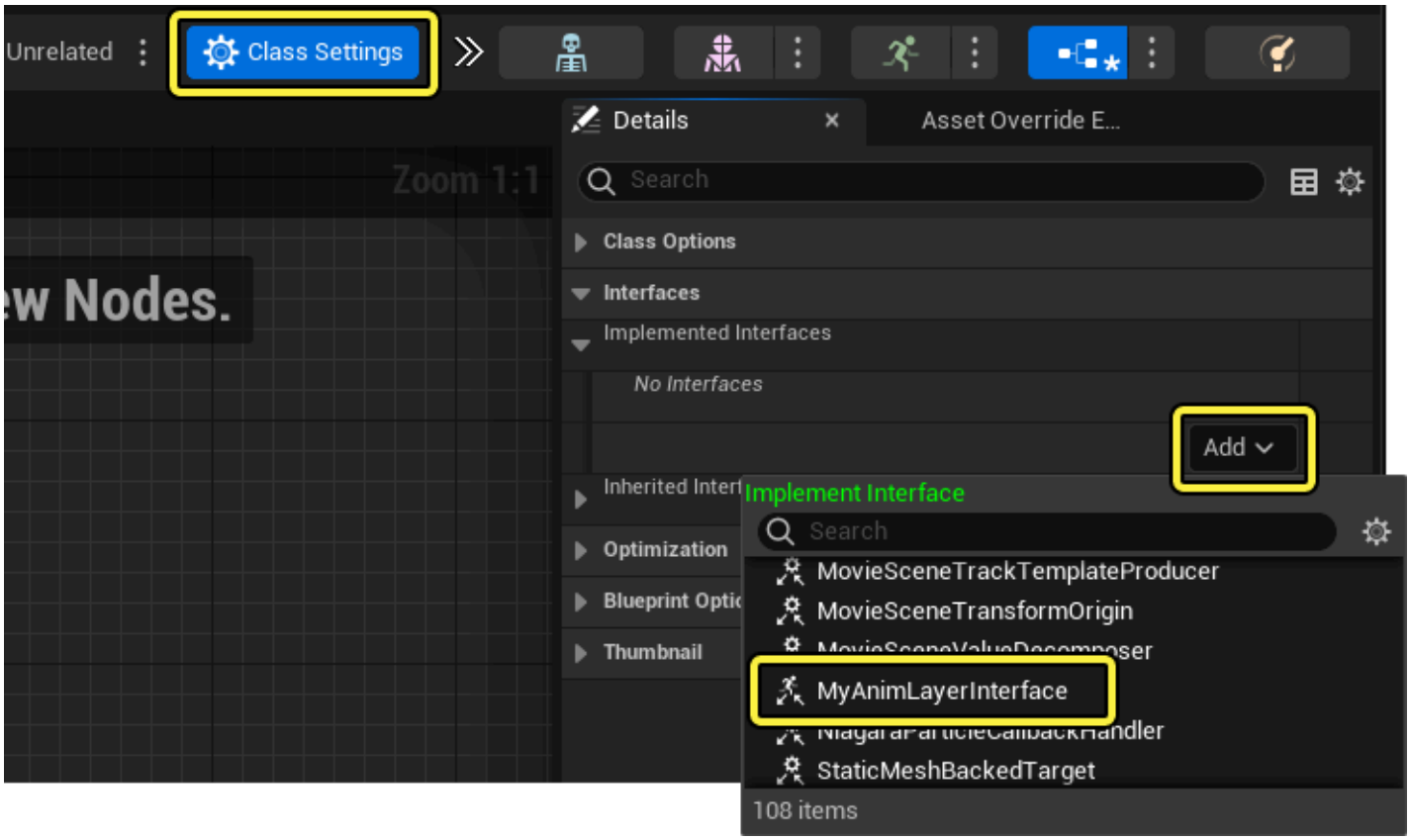


The Animation Layer Interface graph is read-only and is only used to pre-define a list of shared Animation Layers and inputs for your Animation Blueprints.

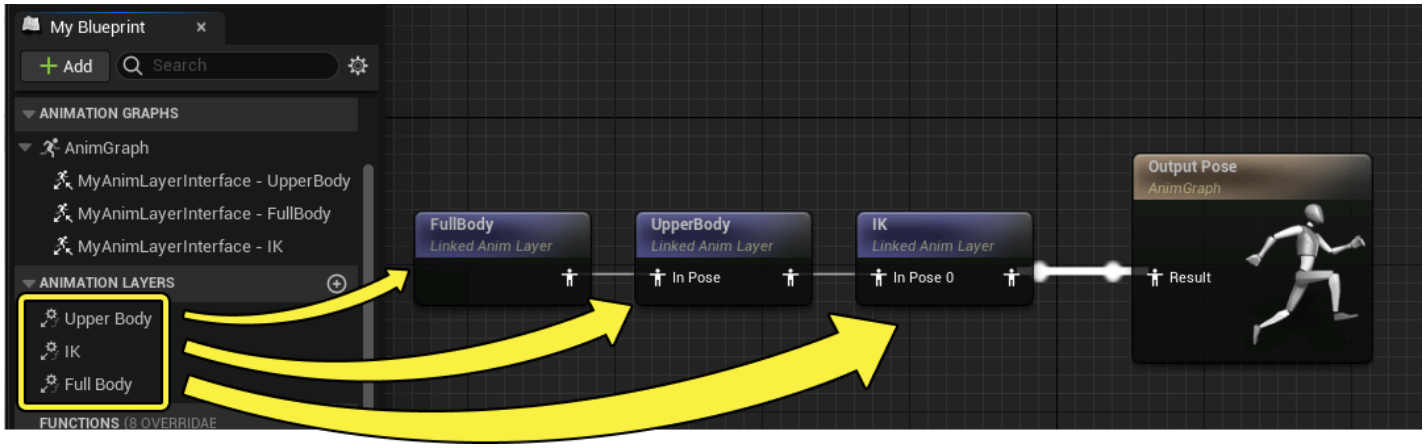
Base Blueprint Setup

When using Linked Anim Layers, you construct a "Base" Animation Blueprint, which you assign to your character. This base may contain the general structure of your linked animation layers, but not necessarily the logic or data within them.

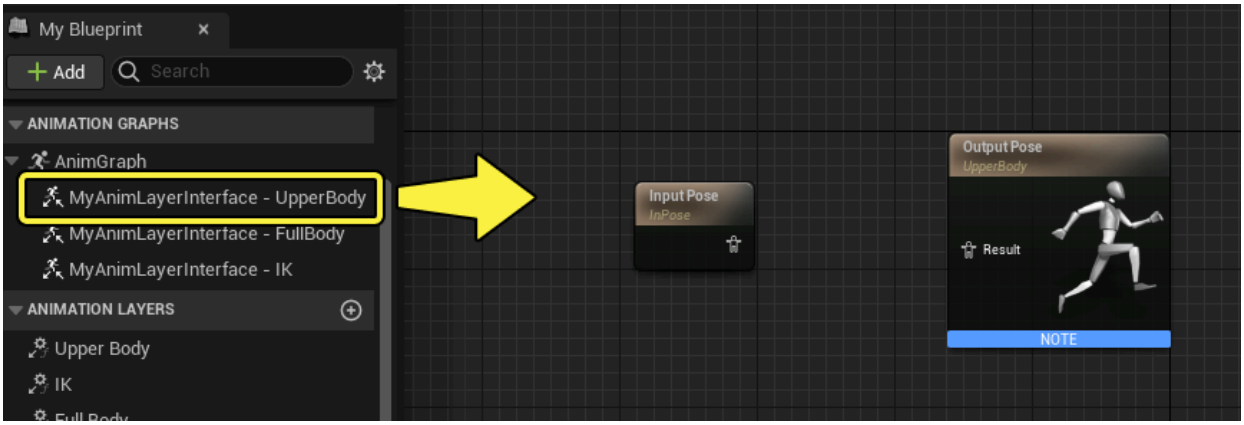
First, you need to bind the Animation Layer Interface to your Animation Blueprint by selecting **Class Settings**, then clicking the **Add** dropdown menu in the **Interfaces** section. Select your **Animation Layer Interface**.



You will now see your predefined layers appear in the Animation Layers section of your Blueprint. Drag these layers into your AnimGraph to build the base logic. In this example the overall structure order is **Full Body > Upper Body > IK**. Both the Upper Body and IK layers have pose inputs added to them to allow for animation to be passed through and modified.



Depending on how complex and modular you want your setup, you can choose to not create any specific logic within your base Animation Blueprint. Your layers can be left empty and default, with the true logic coming from various linked Animation Blueprints.



Specific Blueprint Setup

Next, you need to create your specific logic for one of the Animation Blueprint states on your character. This requires you to create a second Animation Blueprint and also bind the Animation Layer Interface to it.

Once you have created, opened, and bound the interface to the Blueprint, you can now create the logic within each of the linked layers. To continue the example from before, this Animation Blueprint handles the overall animation and behavior for a weapon-equipped character. The layers contain the following logic:

- **Upper Body** contains an **Aim Offset** node that controls the character's weapon aiming behavior, specific to that weapon.
- **IK** contains an **IK Rig** node that controls the final hand placement for the character, specific to that weapon.
- **Full Body** contains a **State Machine** that controls the general locomotion of the character, specific to that weapon or movement-set.

In this way, you partition the entirety of a weapon's Animation Blueprint logic into this Blueprint exclusively, enabling different users to work on these Blueprints. You can create as many different Animation Blueprints as required, with different logic in them depending on the context.

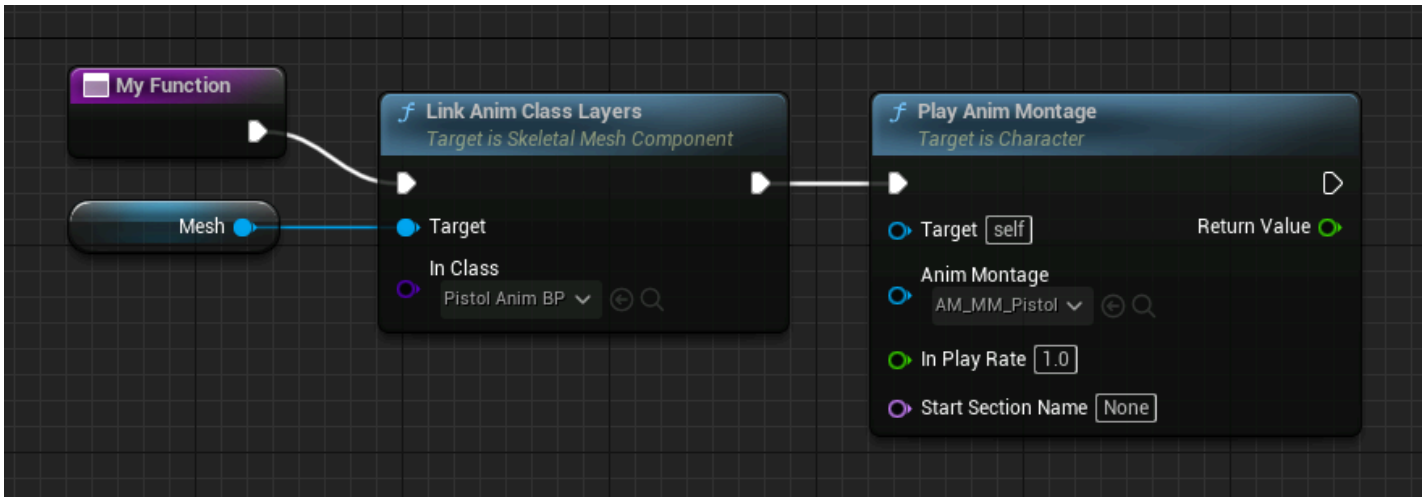
> For these logic-specific Animation Blueprints, you do not need to recreate the same Anim Graph logic that exists in the base Blueprint. Instead you can consider the base Blueprint to be a "parent" Blueprint in terms of maintaining that overall logic structure in the Anim Graph. You only use these specific Blueprints for creating logic within each linked layer.

Linking Anim Layers

The final step in using the Linked Anim Layers is adding the **Link Anim Class Layers** Blueprint node. You use this node to bind the layers of one of your specific Animation Blueprint classes to the Animation Layer system, causing its internal layer logic to overwrite the current logic.

- **Target** is where you connect your Skeletal Mesh component.
- **In Class** is where you specify your Animation Blueprint class.

Continuing the previous example, create a **Link Anim Class Layers** node, with the character mesh connected and the **In Class** input set to the specific weapon Animation Blueprint. In some cases, you may also want to play a Montage after the layers change, to help transition the change with an animation, such as a "weapon equip" animation.

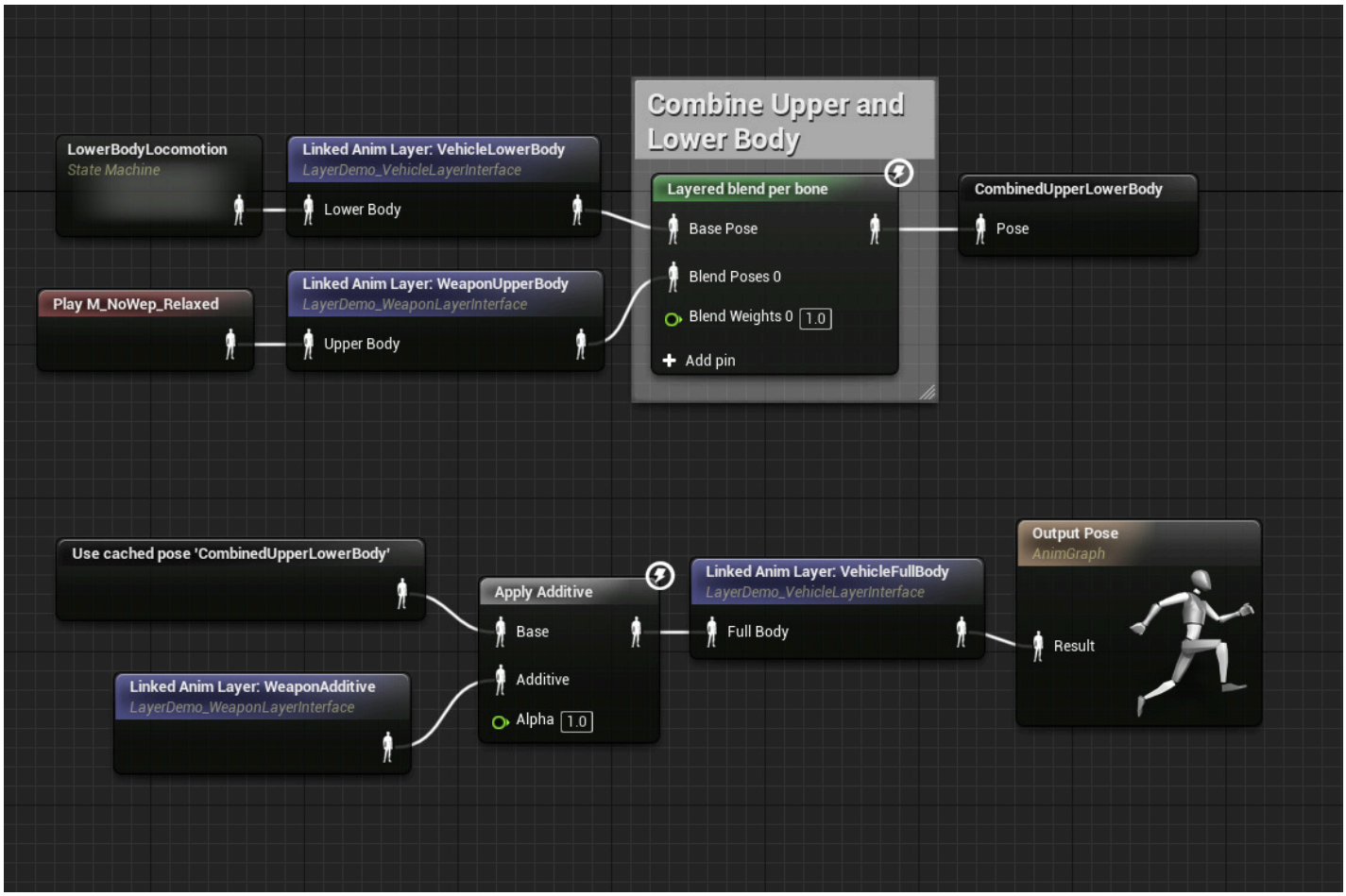


To summarize this example:

- The weapon Actor Blueprint stores a reference to the Weapon Animation Blueprint Layer. This means that the animations related to that specific weapon are only loaded when the weapon is activated in the game.
- You can then create similar logic and assets for other weapons, such as a rifle, bow, or rocket launcher. Each weapon stores a reference to its associated Animation Blueprint Layer and therefore only those specific animations load when those weapons are loaded.

Additional Use Case Example

Below, a use case for Linked Anim Layers used on Fortnite is shown.



In this example, there are two interfaces—one for weapons, and another for vehicles. You can have both active at the same time. It is also possible for a single Animation Blueprint that implements one of these interfaces to override multiple points of the graph, such as a weapon overriding `WeaponUpperBody` and `WeaponAdditive`.

Here are some possibilities with the above setup:

- When driving a car, the vehicle layer overrides the entire pose of the character.
- When riding as a passenger, the vehicle overrides a seated animation on the lower body of the character while the weapon controls the upper body. If the character changes weapons, a new weapon Animation Blueprint will control the upper body while the lower body continues to play from the vehicle.
- A weapon can override the upper body pose, which will then be combined with the lower body pose from the main graph, then a custom additive animation, such as idle noise, will be played on top of the full body pose from the weapon.

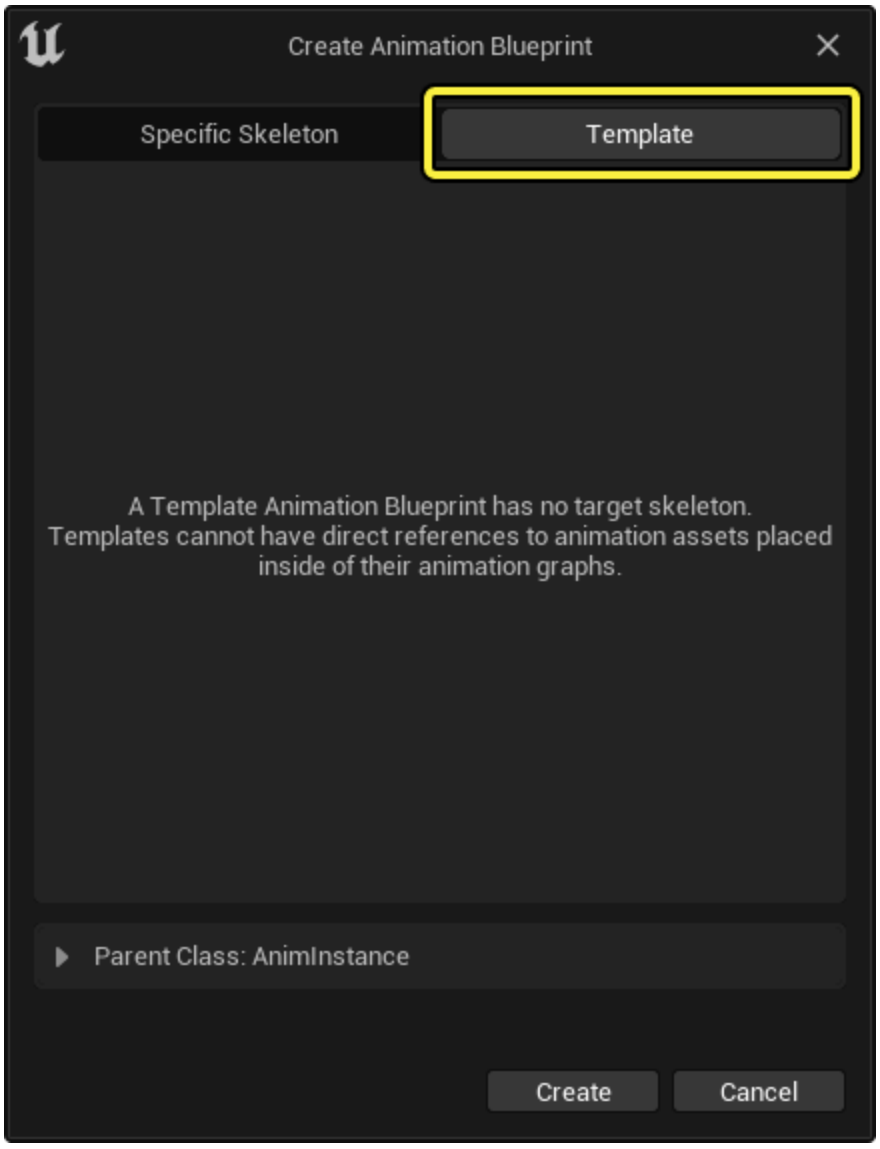
In this way, weapons can override several different points in the main graph. For example, a locomotion State Machine with states like Jump, Fall, Land, and Zipline, is used. Rather than duplicating this State Machine for each weapon, the State Machine lives in the main graph, and weapon Animation Blueprints have a Layer they can override for each state.

If a weapon doesn't need to override some states, then you don't need to connect anything to the output pose in the corresponding Layer. Additionally, Animation Blueprints containing Layers have their own Event Graphs. So, if you need to process data for a specific vehicle, you can keep it contained in that vehicle Animation Blueprint's Event Graph.

Animation Blueprint Templates

Animation Blueprints can also be created as Templates, which are Blueprints that do not reference a specific Skeleton asset or animations. This means you can reuse Animation Blueprint logic in a more modular way.

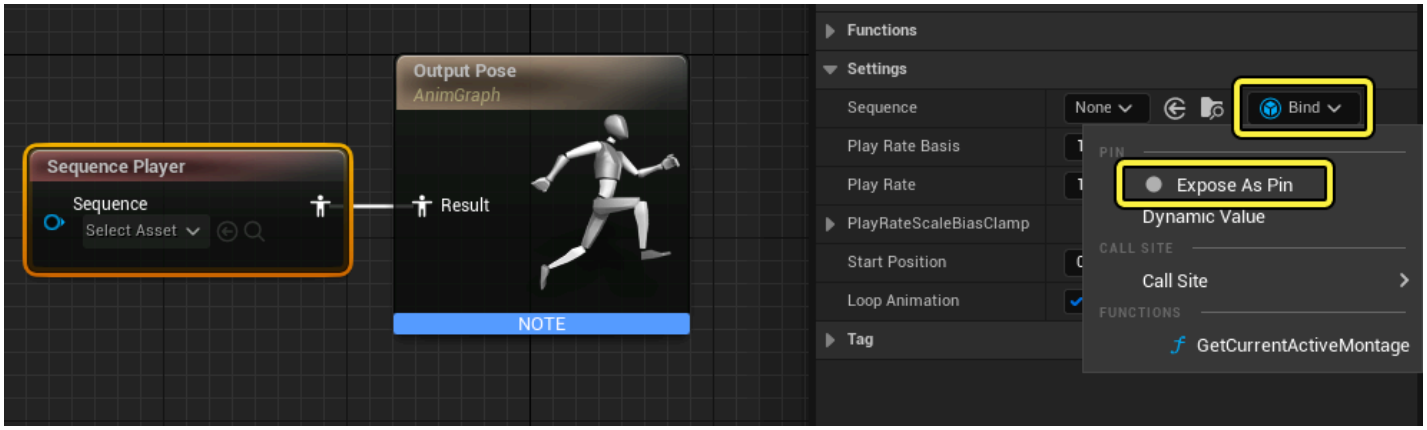
To create a Template Animation Blueprint, follow the normal Animation Blueprint creation process, but instead of specifying a Skeleton, click **Template**, then click **Create**.



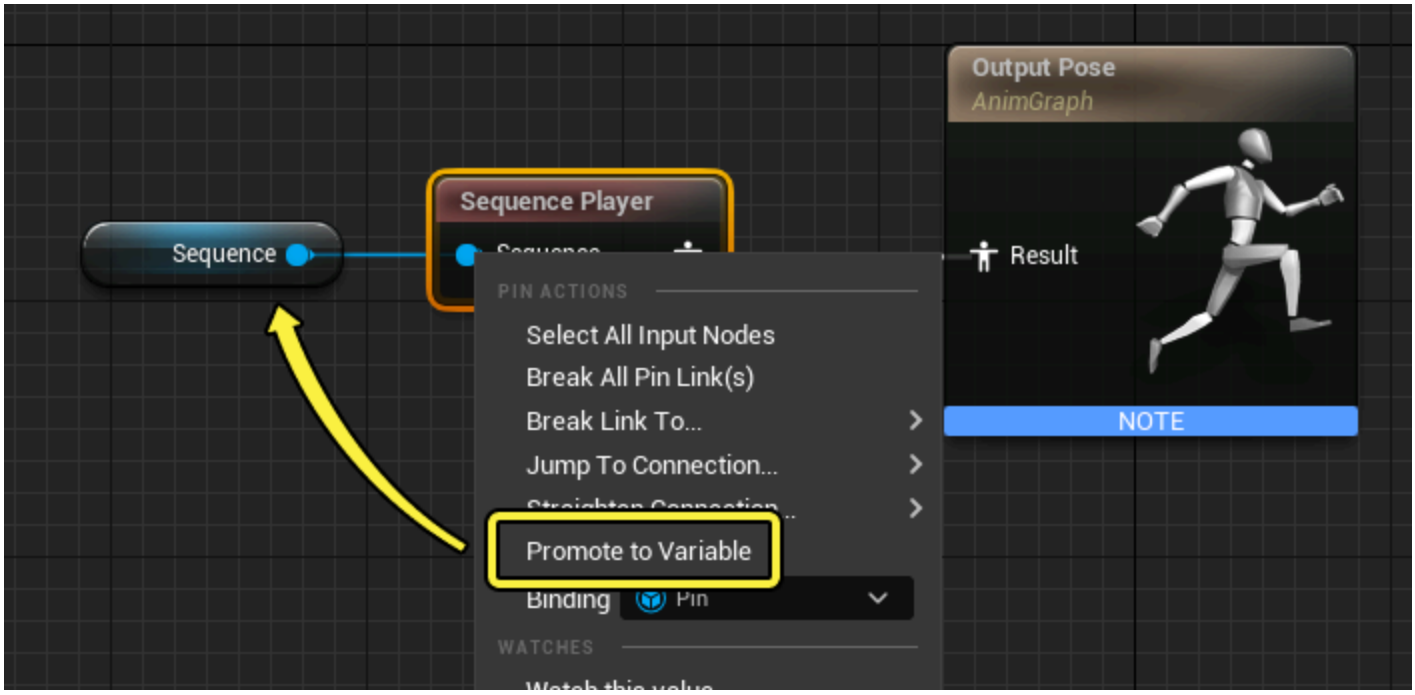
Template Usage

Animation Blueprint Templates contain the same interface and editor found in normal Animation Blueprints. However, because templates do not correspond to any Skeleton, you cannot directly reference animations or assets related to a specific Skeleton. Instead, you can parameterize your template logic and expose Variables that are set in another Blueprint.

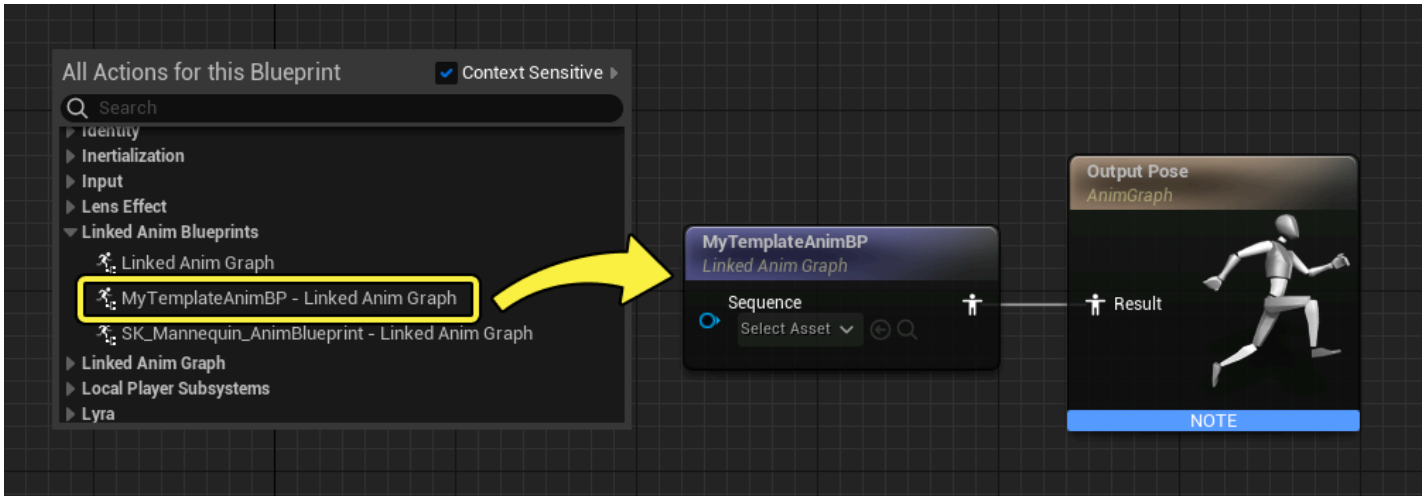
For example, you can create a **Sequence Player** node in the AnimGraph. Typically, this node is used to play Animation Sequences directly, but instead you can expose the **Sequence** property as a **Variable**. To do this, click the **Bind** dropdown menu in the Sequence Player's **Details** panel, and select **Expose As Pin**.



Next, right-click the newly exposed **pin** on the Sequence Player and select **Promote to Variable**, to create a Variable for the Animation Sequence to play. Similar to exposing Linked Anim Graph variables, you must also make this Variable public.



Animation Blueprint templates are referenced in the Anim Graph by right-clicking in the graph and selecting your template from the **Linked Anim Blueprints** section of the context menu.



You can then set any exposed variables on this template to correspond to your Blueprint requirements.