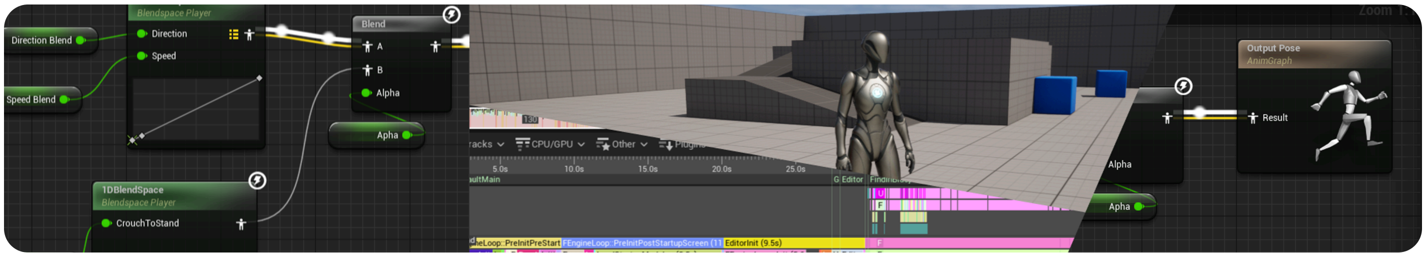
Animation Optimization

Use a variety of methods and techniques to optimize Animation Blueprint's performance and stability.



When developing animation systems in **Unreal Engine** using Animation Blueprints you can use several animation optimization techniques to increase your project's performance.

You can use the following document to learn more about a few best-practice techniques to optimize your animation system in Unreal Engine.

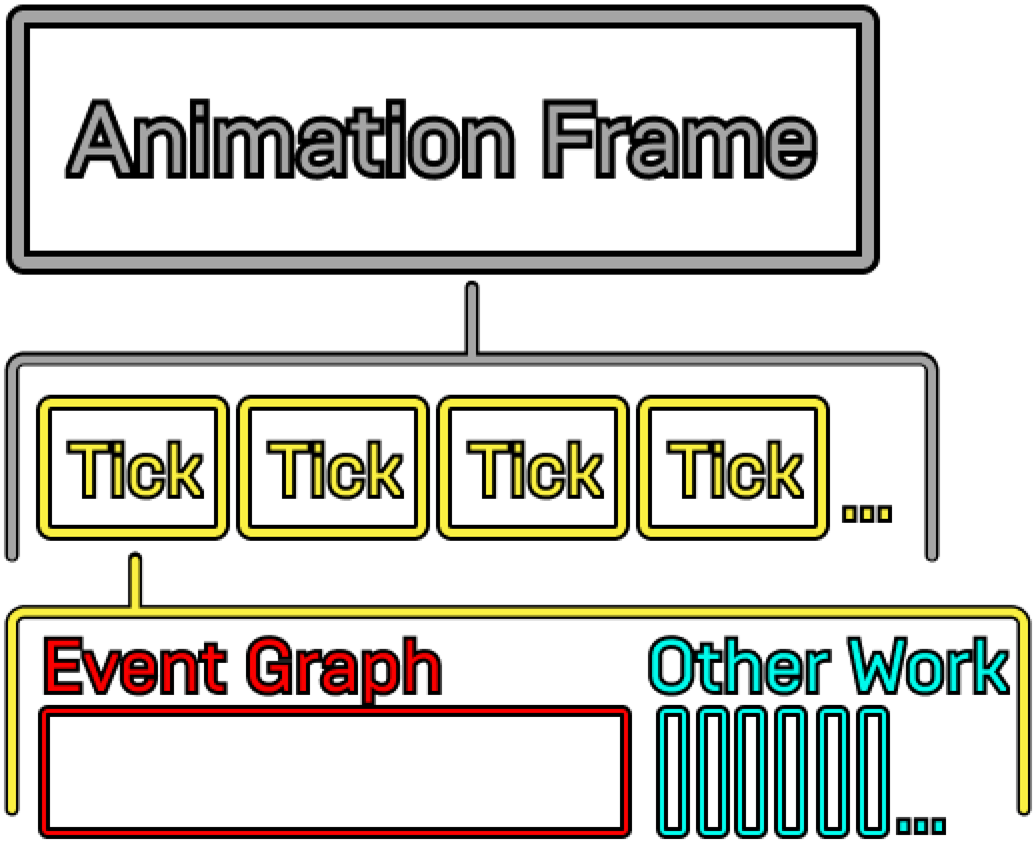
Overview

Your project's Animation System performance, or how efficiently each frame is evaluated, is based on the amount of time your **Game Thread** and **Worker Threads** require to process your animation system each **Tick**.

Animations are added to an Animation Blueprint, where they will be evaluated and played back on the character at runtime. Additional processes, such as animation blends, IK evaluations, physics simulations, and more, will each subtract from your project's performance budget in order to be evaluated. Some processes are simple and don't require much performance budget to evaluate, other processes perform more advanced operations that result in better looking animations, but may require a lot of your performance budget. All animation system features have an associated performance cost.

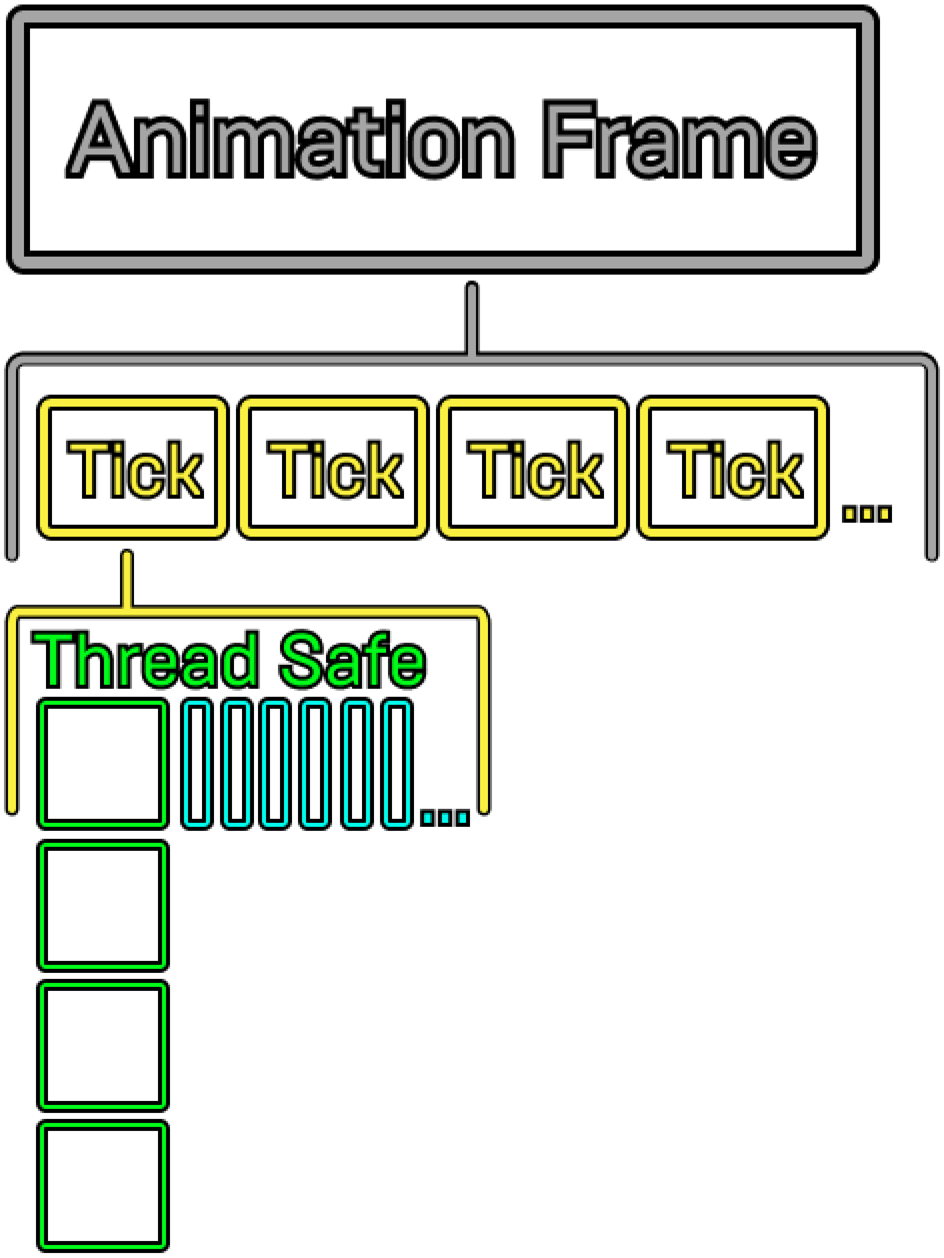
Generally speaking, the most performance-expensive operation of an Animation Blueprint is the **Event Graph** logic. While **AnimGraph** logic can be optimized using systems like Fast Path, it is recommended that you reduce your Event Graph logic as much as possible to achieve the best performance. The Event Graph is evaluated each Tick, with each process occurring sequentially on the **Game Thread**.

The following diagram is a conceptual breakdown of a single frame of animation. Each frame of animation contains several Ticks, and with each Tick the Event Graph is evaluated. Event Graph evaluations are typically the largest operation performed each tick. Event Graph evaluations are sequential, meaning each Tick's evaluation takes longer to complete.



You can optimize this process by relocating Event Graph logic to **Thread Safe functions** that can be evaluated simultaneously on available **Worker Threads**.

The following diagram illustrates the drastic reduction in the time each Tick takes to complete. When relocating all Event Graph operations to Thread Safe functions, operations can be performed simultaneously, thus reducing the amount of time required to evaluate each Tick by a significant margin and improving your animation systems performance.



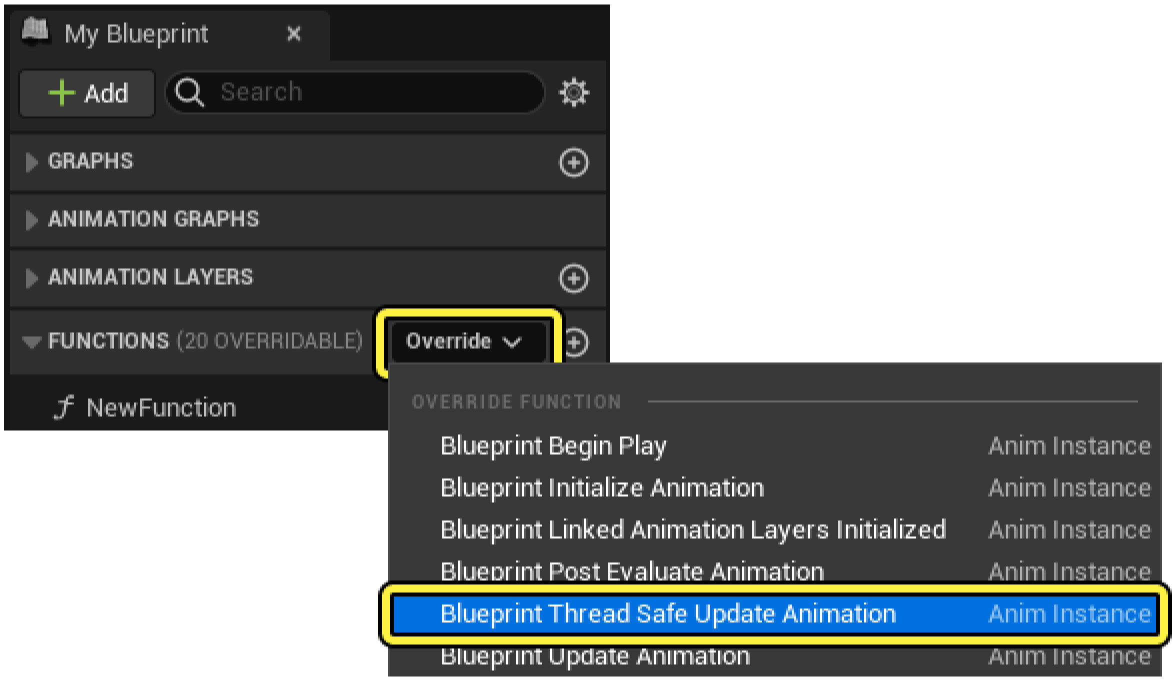
Using Multi-Threaded Animation Updates

Animation Blueprint Event Graphs always run on the **Game Thread**. In order to optimize the logic within the Event Graph to take advantage of multi-threading, you can instead build logic using **Thread Safe functions**.

To ensure thread safety, all references to data derived from other blueprints and components within your project, such as variables, must be called by your Animation Blueprint, rather than pushed to it.

Blueprint Thread Safe Update Animation

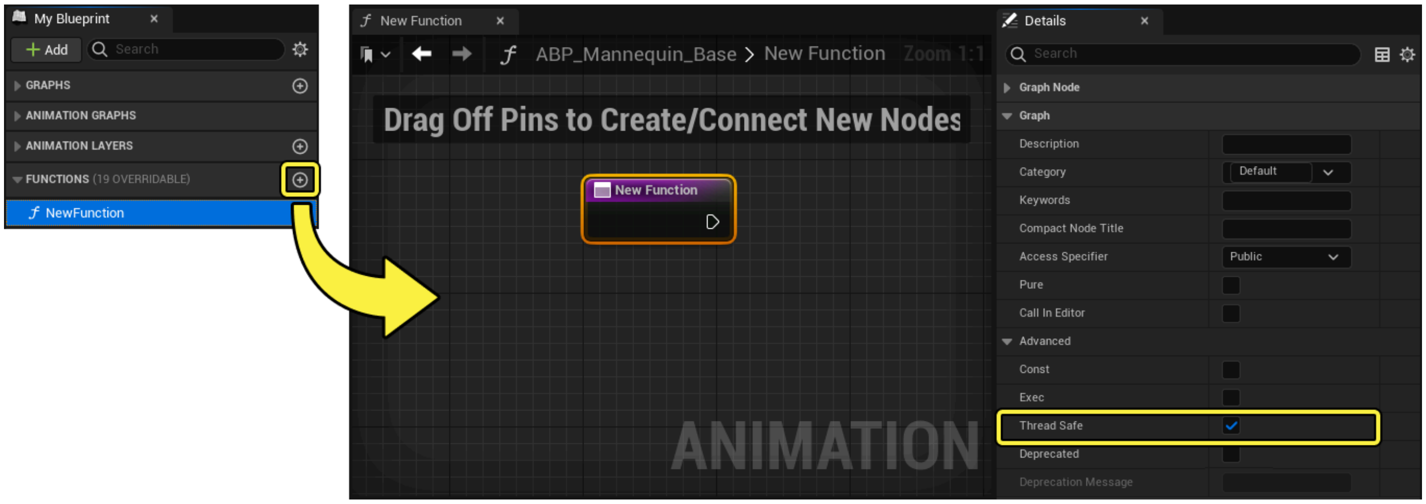
You can use the **Blueprint Thread Safe Update Animation** override function to evaluate logic in your Animation Blueprint in a Thread Safe manner. You can add the Blueprint Thread Safe Update Animation function to your Animation Blueprint in the My Blueprint panel by selecting it in the **Override** down-down menu adjacent to the **Function** section.


My Blueprint
Add
Search
GRAPHS
ANIMATION GRAPHS
ANIMATION LAYERS
FUNCTIONS (20 OVERRIDABLE)
Override
NewFunction
OVERRIDE FUNCTION
Blueprint Begin Play
Anim Instance
Blueprint Initialize Animation
Anim Instance
Blueprint Linked Animation Layers Initialized
Anim Instance
Blueprint Post Evaluate Animation
Anim Instance
Blueprint Thread Safe Update Animation
Anim Instance
Blueprint Update Animation
Anim Instance

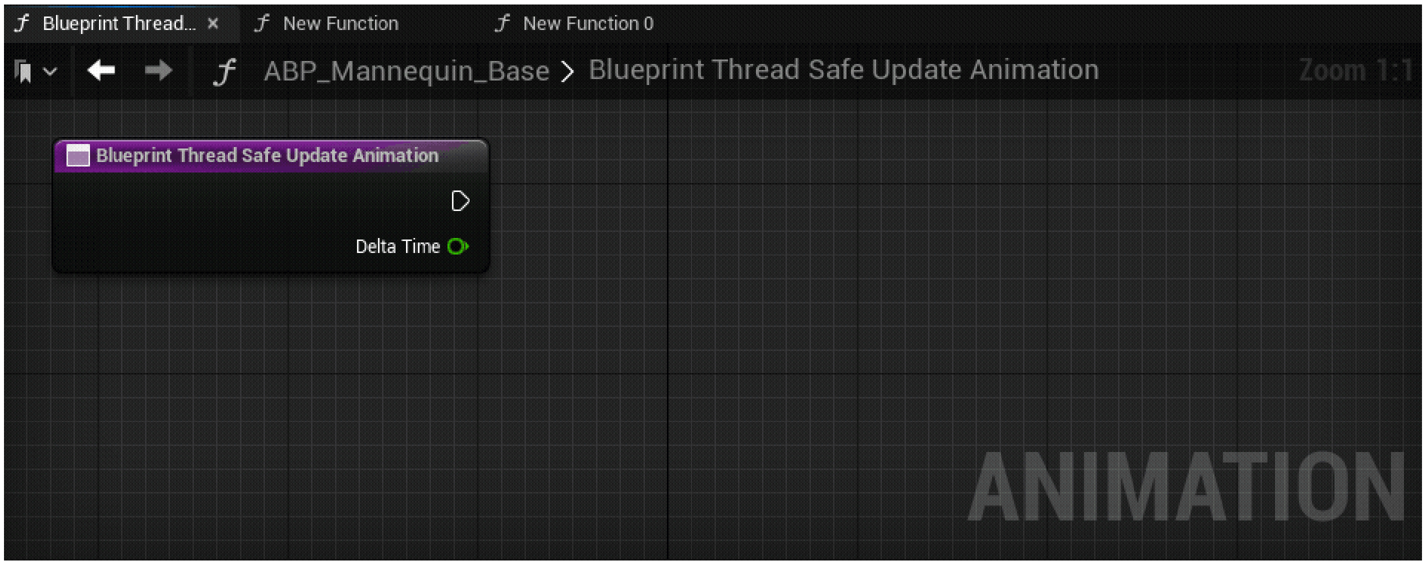
Thread Safe Functions

Thread Safe functions are Blueprint Functions you can use to perform logic to set variables and properties that can be used by your animation system, in addition to performing other operations typically performed in the Event Graph.

To create a Thread Safe function in your Animation Blueprint, create a new function in the **My Blueprint** panel using (**+**) **Add**. After creating a new function, open its **Details** panel and enable the **Thread Safe** property.

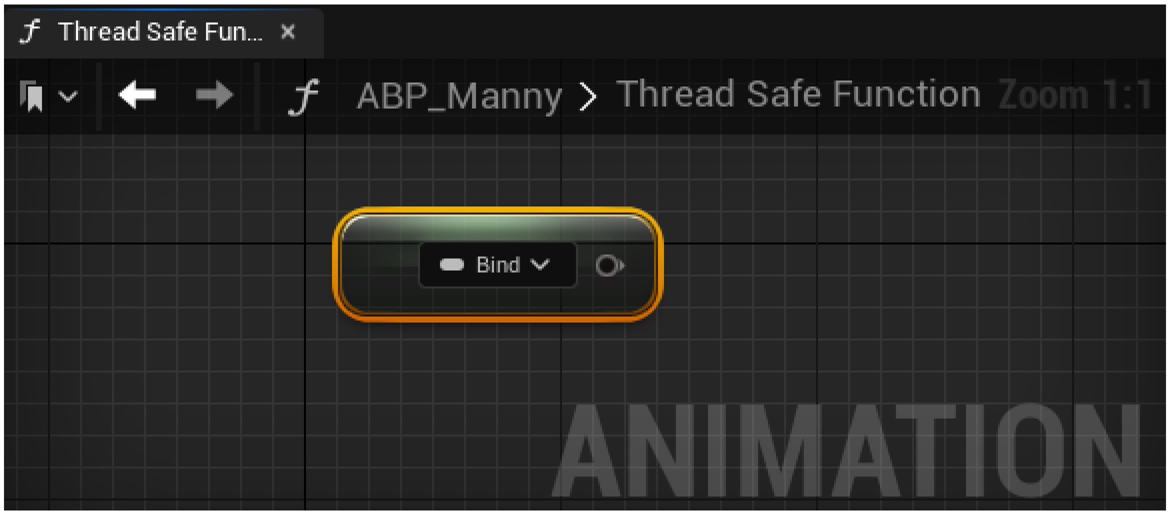


Thread Safe enabled functions can then be added to the Blueprint Thread Safe Update Animation override function to be evaluated simultaneously when worker threads become available each Tick.


Blueprint Thread...
New Function
New Function 0
ABP_Mannequin_Base > Blueprint Thread Safe Update Animation
Zoom 1:1
Blueprint Thread Safe Update Animation
Delta Time
ANIMATION

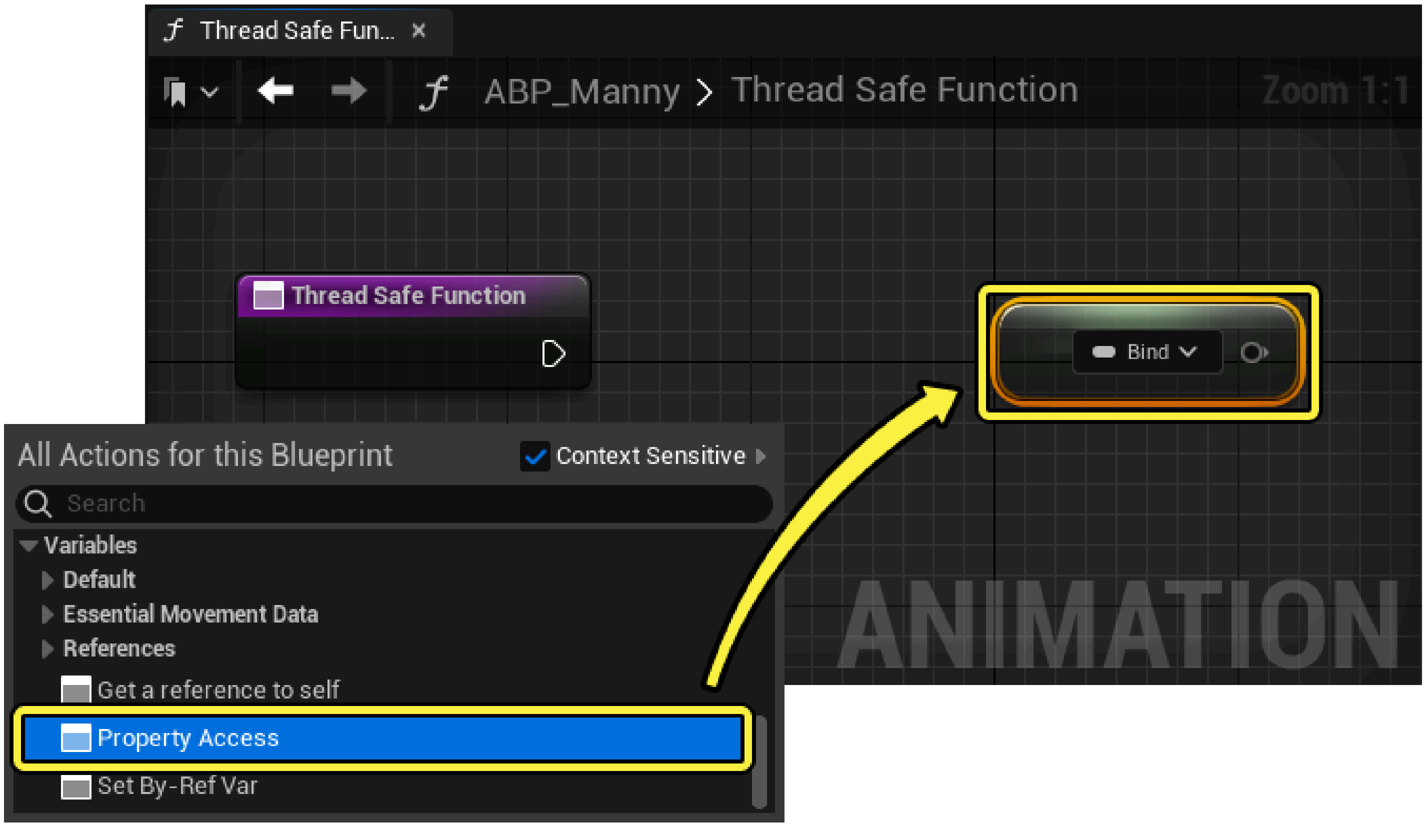
Property Access

Thread Safe functions are unable to directly access non-thread safe blueprints and components. In order to access non-thread safe blueprints and their properties safely, you can use the **Property Access** feature to read their data and call their functions. Property Access can be used as a standalone node within a thread safe function's graph, or within a pin's properties on AnimGraph nodes.

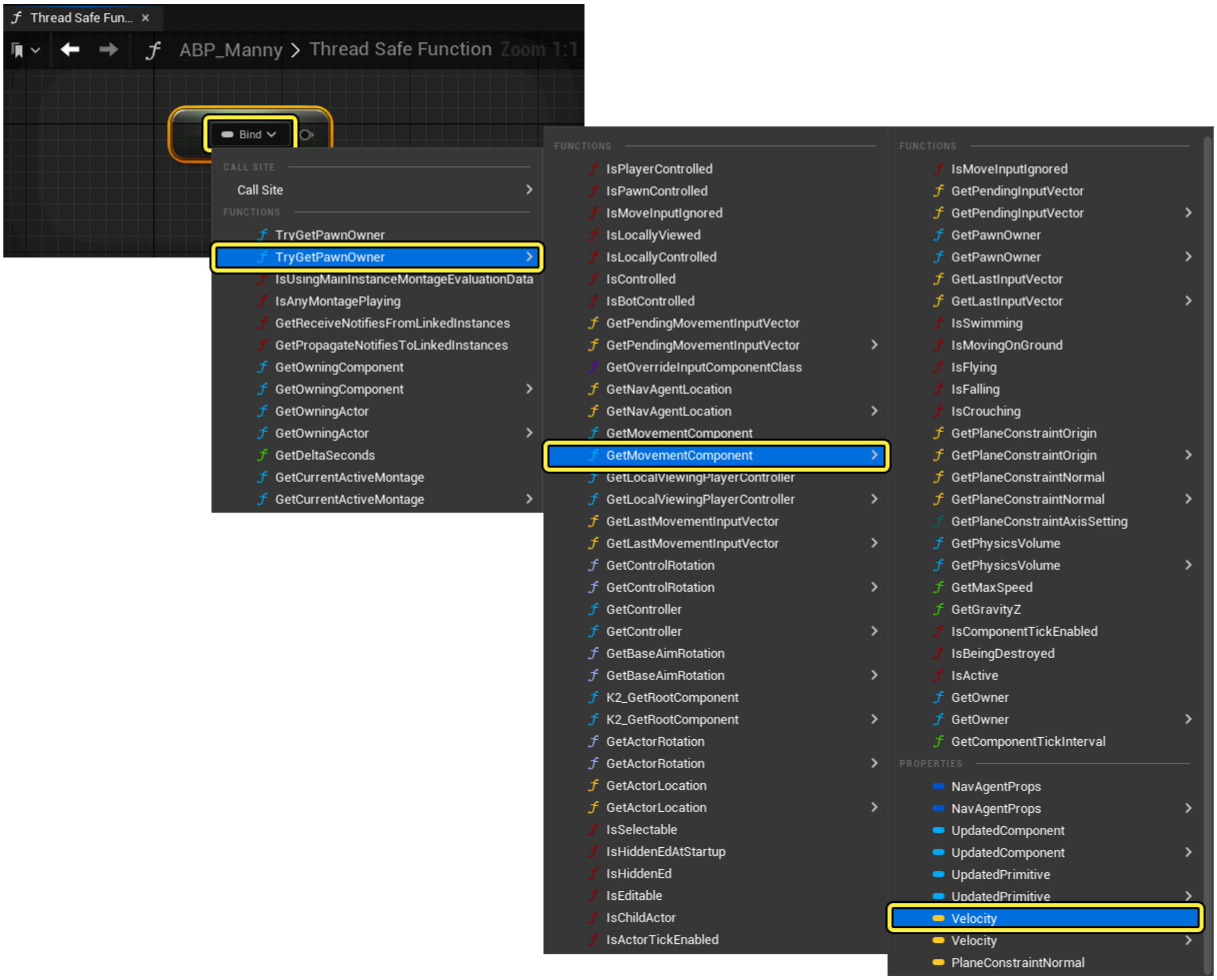


Property Access Node | Property Access Pin

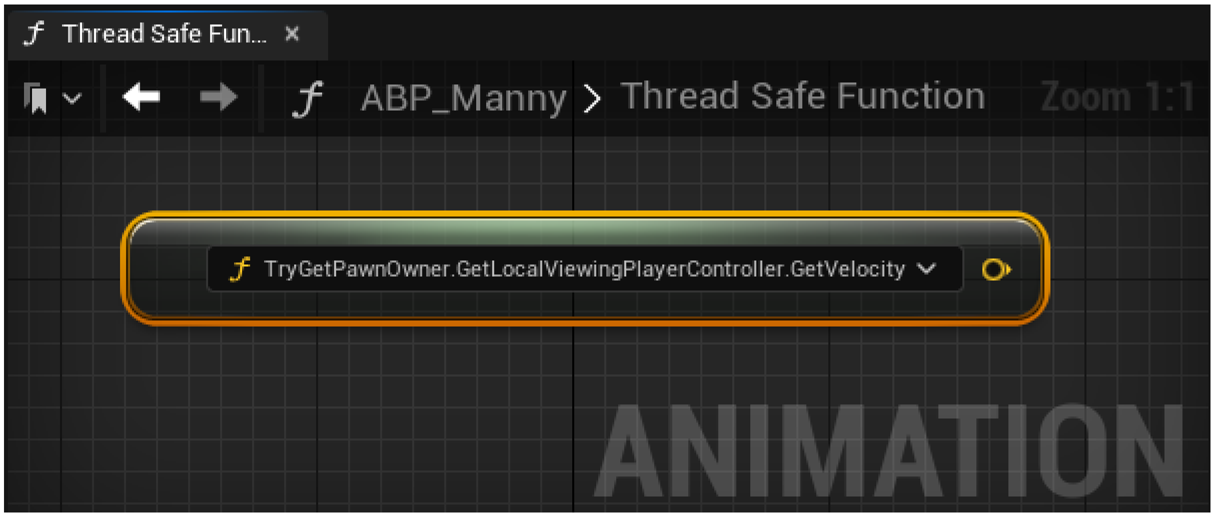
To create a Property Access node, right-click in the graph of a Thread Safe function, and select the **Property Access** option in the context-menu.



You can use Property Access nodes to reference variables, components, and data found in other blueprints and game objects relative to your Animation Blueprint. To define the source of a Property Access data call, select the **Bind** drop-down menu, and then select the component or property you want to reference. By selecting a property or component you can reference the object or data directly, or you can navigate in the nested options of a property or component to directly access its components or properties.



You can now use a bound Property Access reference to set additional properties or variables within a Thread Safe blueprint function.


Thread Safe Fun...
ABP_Manny > Thread Safe Function Zoom 1:1
TryGetPawnOwner.GetLocalViewingPlayerController.GetVelocity
ANIMATION

For more information about using Thread Safe functions, and Property Access data, see the Graphing in Animation Blueprints documentation.

For a workflow example of using Thread Safe functions and Property Access nodes to get Animation Variables, see the How to Get Animation Variables documentation.

For a workflow example of optimizing an Unreal Engine project to use Thread Safe functions, see the Adapting Lyra's Animation System blog post.

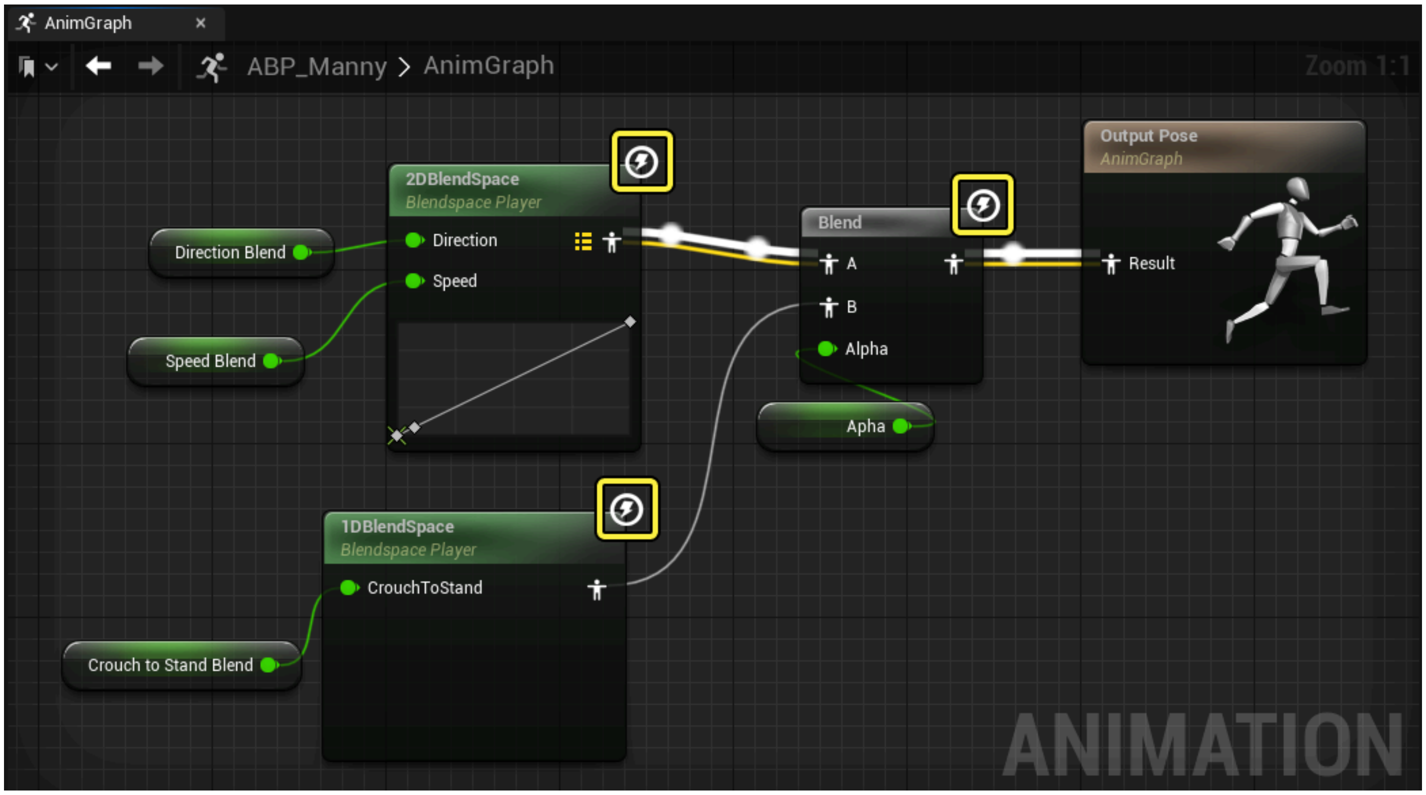
Animation Fast Path

Animation Fast Path provides a way to optimize variable access inside the **AnimGraph** update. This enables the engine to copy parameters internally rather than executing Blueprint code, which involves making calls into the **Blueprint Virtual Machine**. The compiler can currently optimize the following constructs:

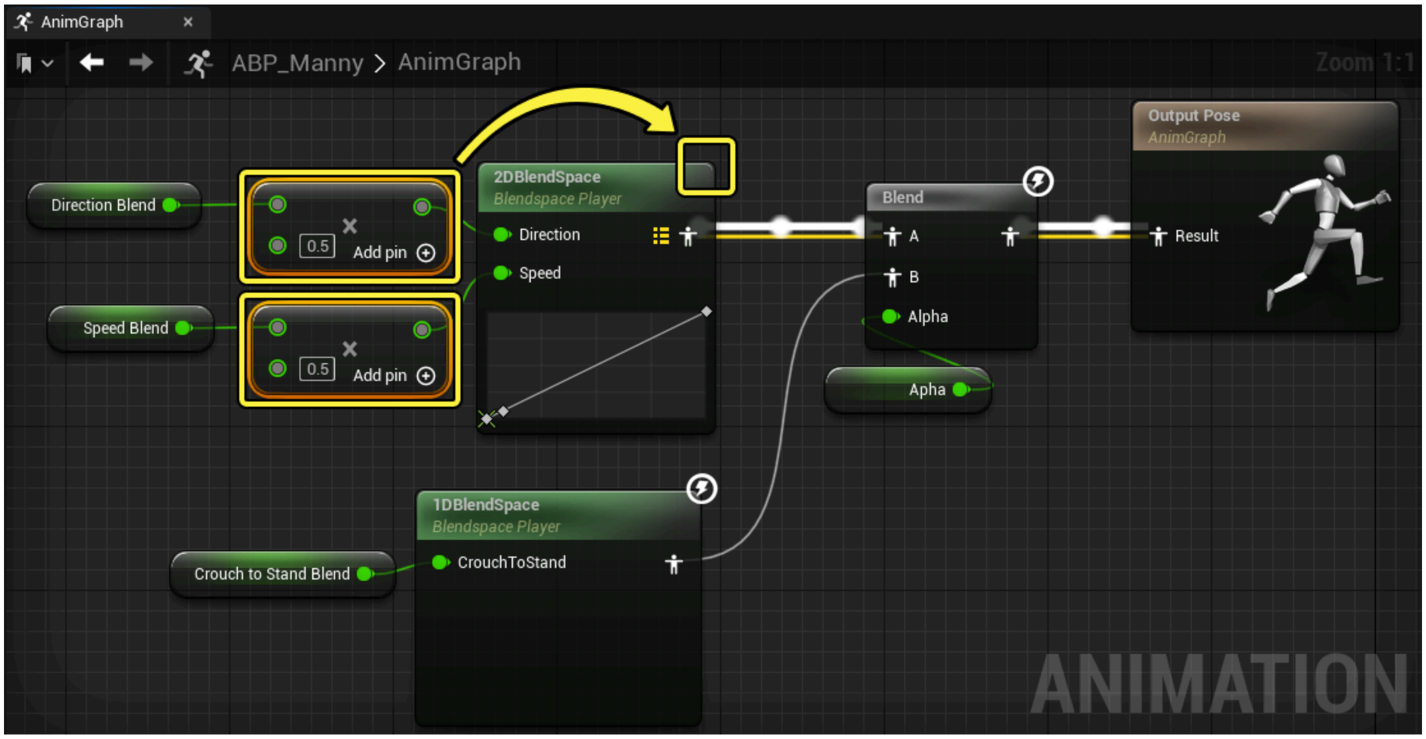
- **Member Variables**
- **Negated Boolean Member Variables**
- **Members of a Nested Structure**

To make use of Animation Fath Path, inside the AnimGraph of your Animation Blueprints, ensure that no Blueprint logic is being executed.

In the following example blueprint, the AnimGraph is reading several float values, which are being used to drive multiple **Blend Space** assets and a **Blend** node resulting in the **Output Pose**. Each node denoted with the lightning icon in the upper right corner is utilizing Fast Path as no logic is being executed.



If any form of calculation is added to the graph, the associated node would no longer be using Fast Path. In the following example, a simple multiply function is added to the float variables, resulting in the Blend Space node being unable to use Fast Path. After the graph is compiled, the lightning icon is removed to denote this change.

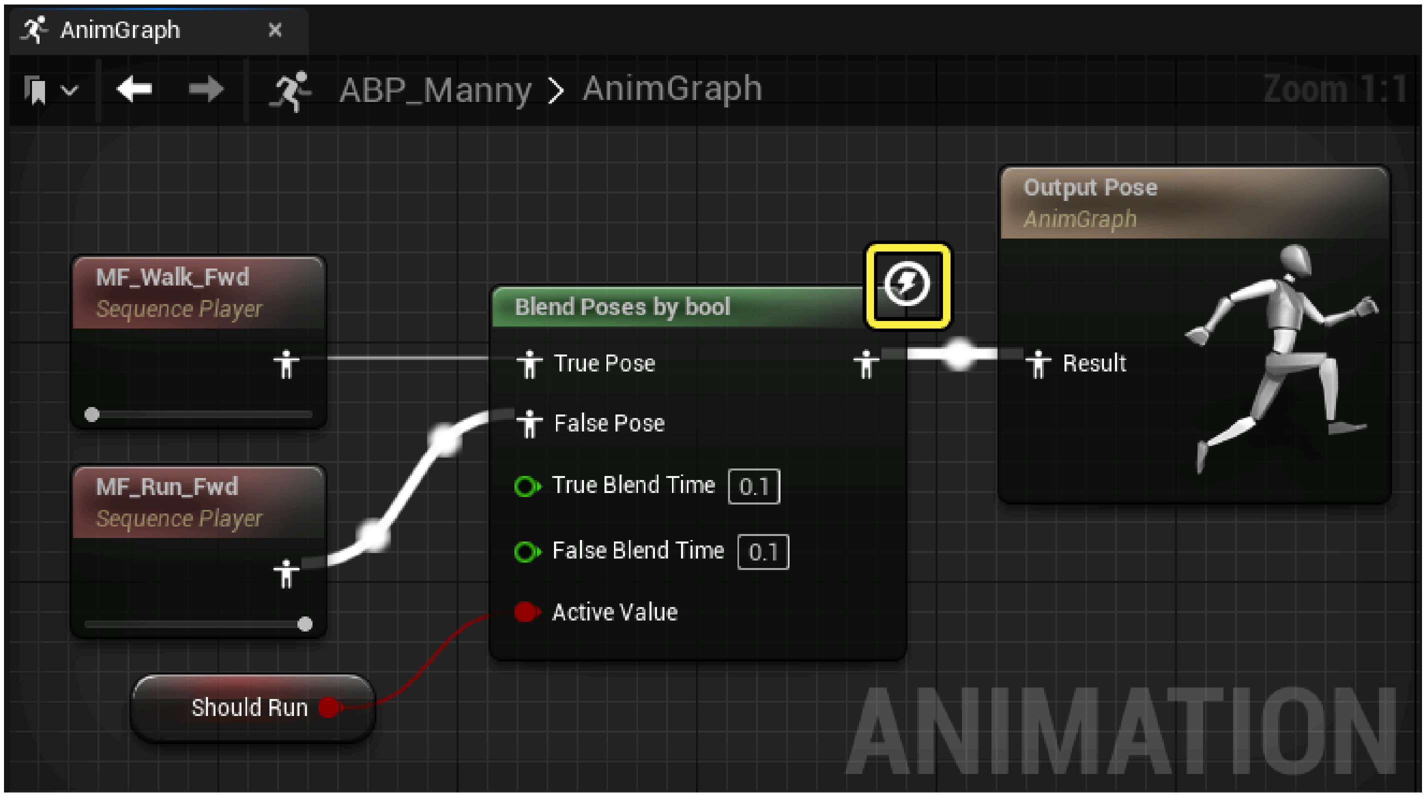


Fast Path Methods

The following are the methods you can use to implement Fast Path variable access in your Animation Blueprints.

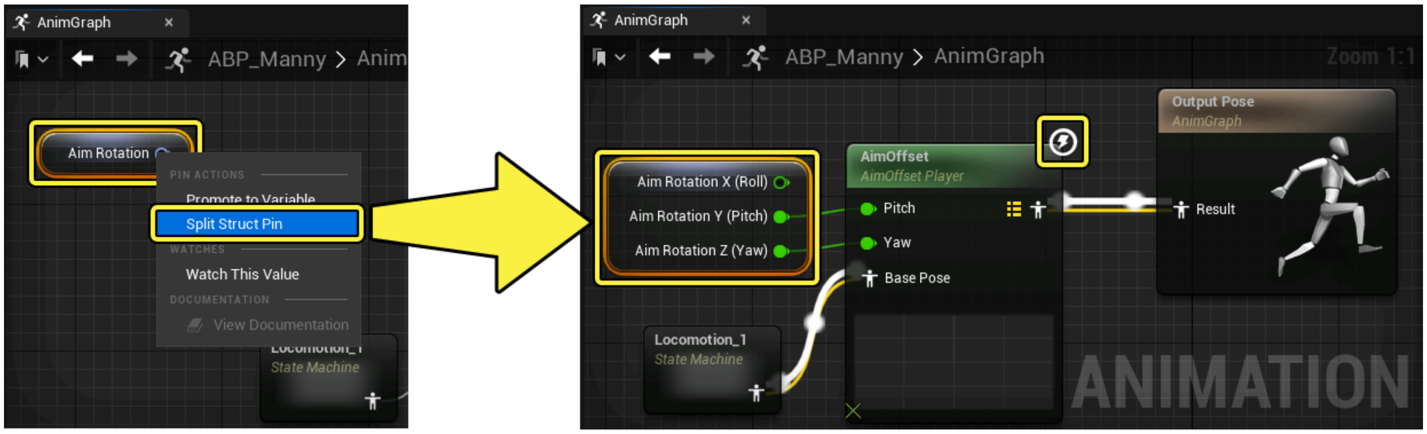
Access Member Variables Directly

You can use Fast Path by directly accessing and reading the value of a variable to determine the output pose.



Access Members of a Nested Struct

You can break nested structs, such as a rotator variable, to directly access its components while still using Fast Path. You can break a struct directly by right-clicking the variable in the graph, and selecting **Split Struct Pin** from the context menu or by using a **Break** node. You can now access its component values directly.

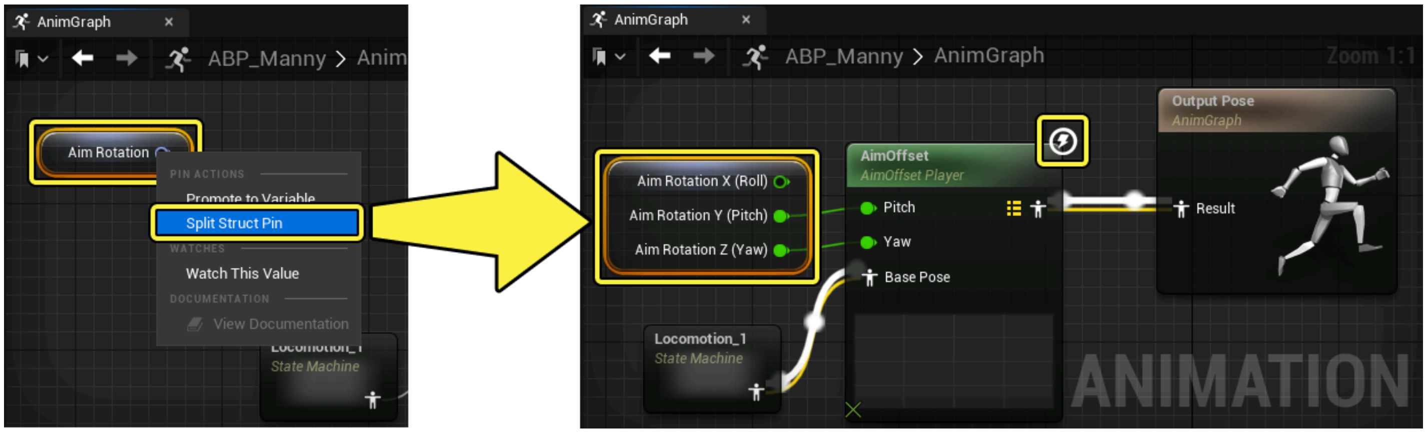


Some **Break Struct** nodes like **Break Transform** will not use Fast Path as they perform conversions internally rather than simply copying data.

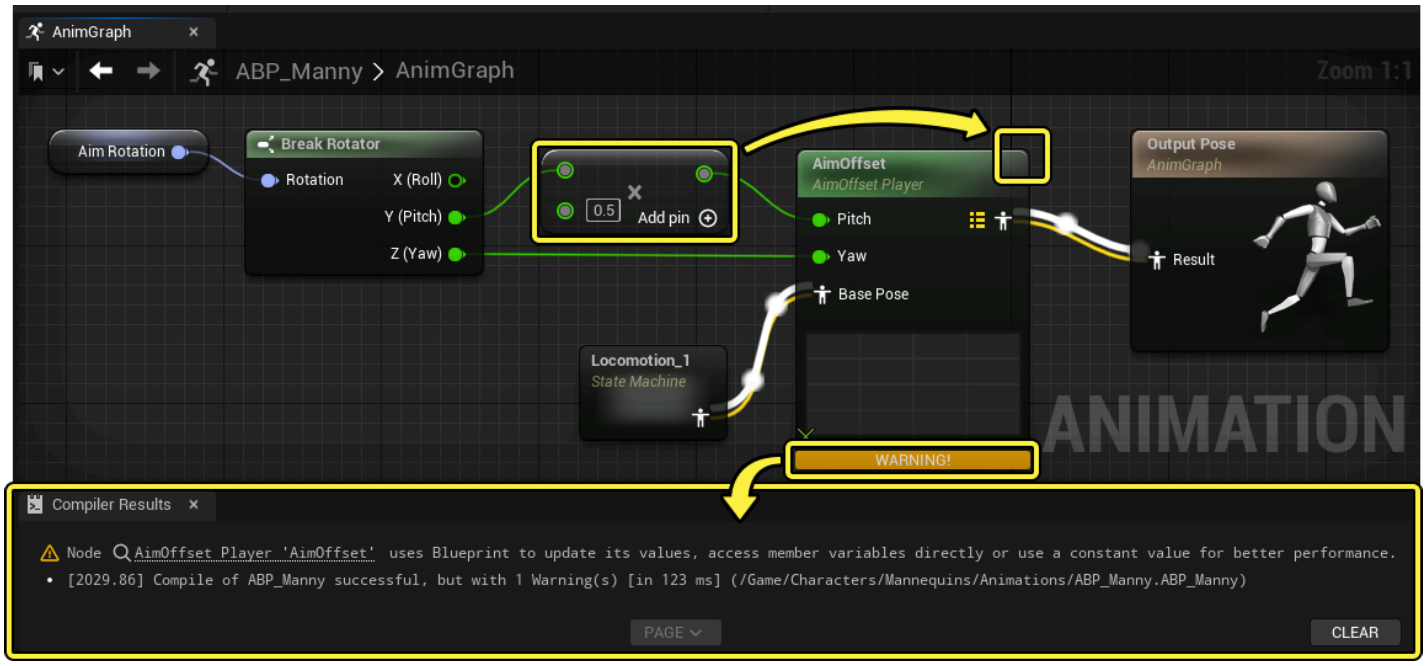
Warn About Blueprint Usage

To ensure that your Animation Blueprints are using Fast Path, you can enable the **Warn About Blueprint Usage** property. When Warn About Blueprint Usage is enabled, the compiler will emit warnings to the **Compiler Results** panel when a call into the Blueprint Virtual Machine is made from the AnimGraph.

To enable **Warn About Blueprint Usage**, enable the option inside the **Class Settings** of your **Animation Blueprint** under **Optimization**.



When executing Blueprint logic in the AnimGraph with the Warn About Blueprint Usage property enabled, a warning message in the **Compiler Results** panel will appear when a variable that can be accessed using Fast Path is not. You can click on the link in the warning message to focus the graph on the source of the warning. This can help track down optimizations that need to be made and will enable you to identify any node variable access that may be optimizable.



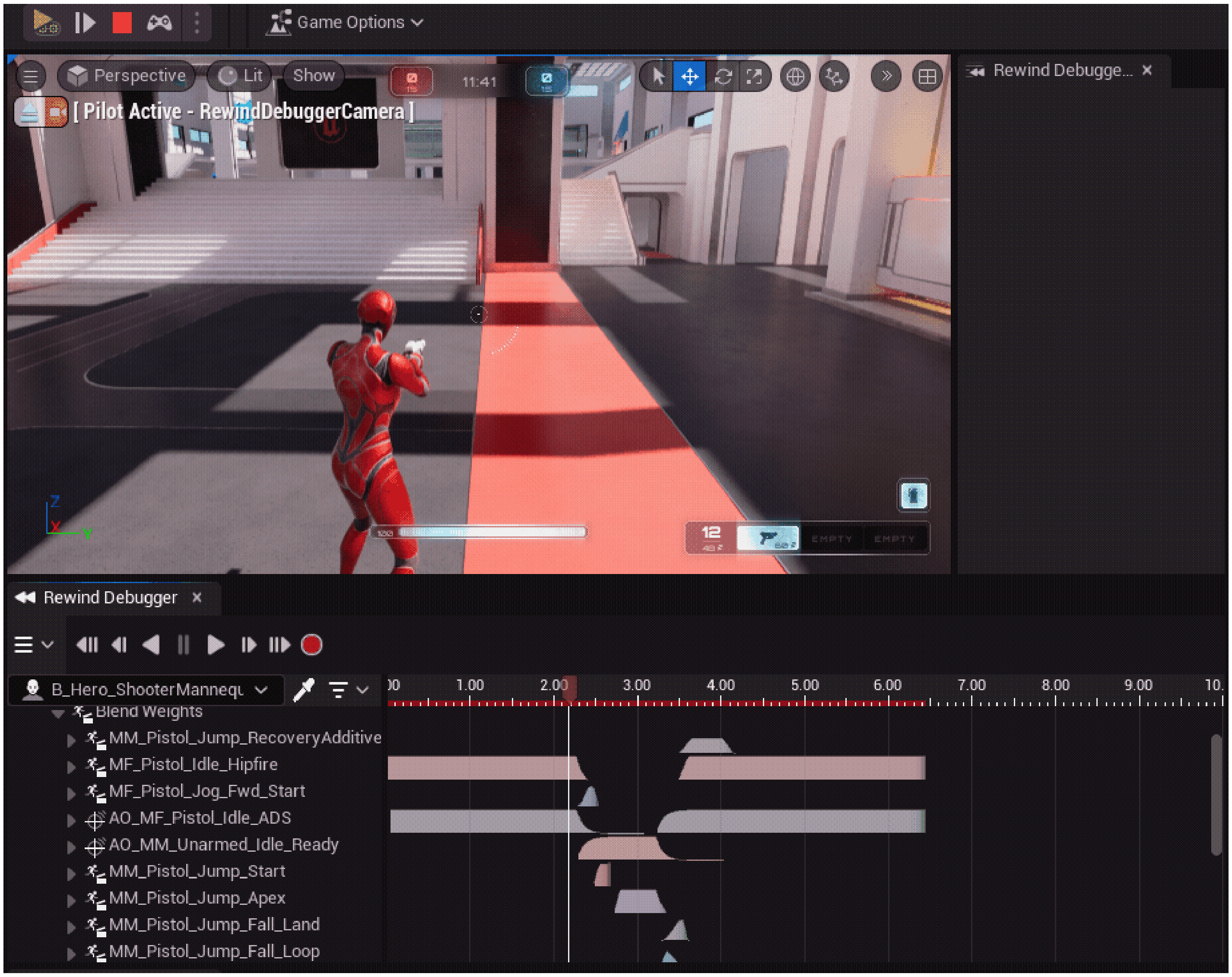
```
⚠ Node AimOffset Player 'AimOffset' uses Blueprint to update its values, access member variables directly or use a constant value for better performance.
• [2029.86] Compile of ABP_Manny successful, but with 1 Warning(s) [in 123 ms] (/Game/Characters/Mannequins/Animations/ABP_Manny.ABP_Manny)
```

Animation Optimization Tools

Unreal Engine provides a suite of debugging tools you can use to analyze your animation system, to find optimization opportunities.

Rewind Debugger

You can use the Rewind Debugger to record segments of **Play in Editor** (**PIE**) simulation, to analyze your animation system in real time. Using **Traces**, you can observe the evaluations of your animation system's properties to target bugs and performance issues that can be optimized.



For more information about setting up and using the Rewind Debugger, see the following documentation:

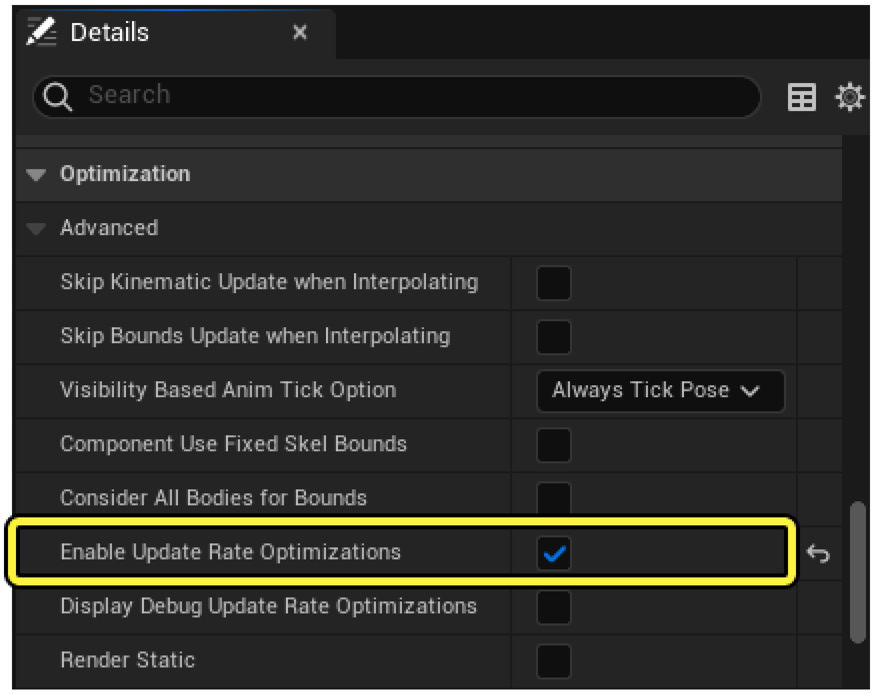
Rewind Debugger

With the Rewind Debugger you can record real-time segments of projects and preserve the data for debugging workflows.

General Tips

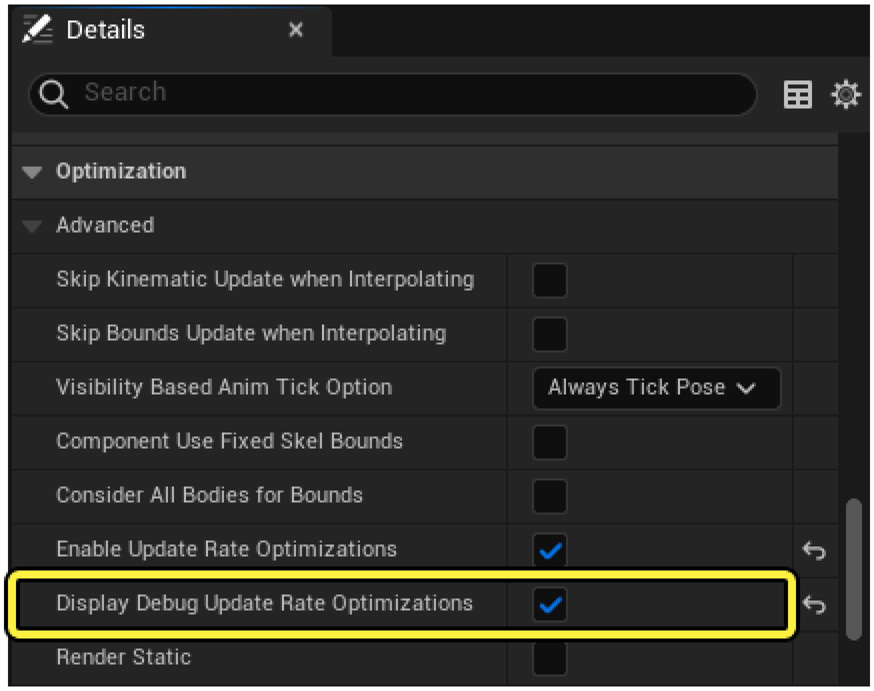
As you start to consider the performance of your project's animation system, you can consider the following guidelines during the optimization process. Each project has unique optimization requirements, based on the size and scope of your project, more invasive changes may be needed, however, the following guidelines provide a general approach most projects can benefit from.

- Make sure that the conditions for Parallel Updates are met.
    - In UAnimInstance::NeedsImmediateUpdate struct, you can see all the conditions that must be met to avoid the update phase of animation running on the **game thread**. If **root motion** is required for character movement, the parallel update cannot be performed as character movement is not multi-threaded.
- Use **Update Rate Optimizations** (URO) when possible.
    - URO will prevent your animations from ticking too often. Controlling how this is applied will depend on the needs of your project, but it is recommended to target Update Rates that perform at 15Hz and under, at appropriate distances, for most characters, as well as disabling interpolation.
    - To enable URO, navigate in your Skeletal Mesh Component's **Details** panel to the **Optimization** section, and select the **Enable Update Rate Optimizations** property. You can then use the `AnimUpdateRateTick()` struct to set and observe your blueprint's tick rate.



Skip Kinematic Update when Interpolating	☐
Skip Bounds Update when Interpolating	☐
Visibility Based Anim Tick Option	Always Tick Pose
Component Use Fixed Skel Bounds	☐
Consider All Bodies for Bounds	☐
Enable Update Rate Optimizations	✓
Display Debug Update Rate Optimizations	☐
Render Static	☐

- Optionally, you can also enable the **Display Debug Update Rate Optimizations** property, in the skeletal mesh components details panel, to enable an onscreen debug display of the rate of URO that is applied to your project during simulation.


Details

Search

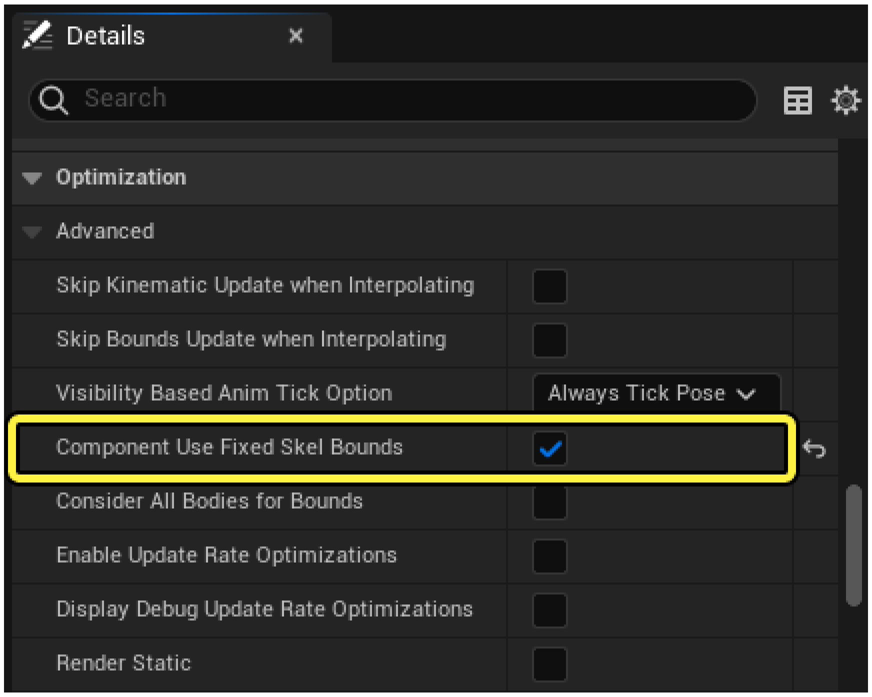
Optimization

Advanced

Skip Kinematic Update when Interpolating	☐
Skip Bounds Update when Interpolating	☐
Visibility Based Anim Tick Option	Always Tick Pose
Component Use Fixed Skel Bounds	☐
Consider All Bodies for Bounds	☐
Enable Update Rate Optimizations	✓
Display Debug Update Rate Optimizations	✓
Render Static	☐

Instead of using URO, it is recommended to use the Animation Budget Allocator plugin, to control the tick rate of Animation Blueprint evaluations.

- Enable **Component Use Fixed Skel Bounds**, when your character does not need access to its Physics Asset.
  - In your Skeletal Mesh Component's **Details** panel, enable the **Component Use Fixed Skel Bounds** property.


Details

Search

Optimization

Advanced

Skip Kinematic Update when Interpolating	☐
Skip Bounds Update when Interpolating	☐
Visibility Based Anim Tick Option	Always Tick Pose
Component Use Fixed Skel Bounds	✓
Consider All Bodies for Bounds	☐
Enable Update Rate Optimizations	☐
Display Debug Update Rate Optimizations	☐
Render Static	☐

  - This will skip using a Physics Asset and will instead always use the fixed bounds defined in the Skeletal Mesh.

    - This will also skip recalculating bounding volumes for culling for every frame, increasing performance.

Other Considerations

When profiling your project, using Animation Insights, you may see that FParallelAnimationCompletionTask is being run for Skeletal Meshes on the Main Thread after Worker Threads have completed their processes. This process will be the bulk of the main thread work that you will see in your profile once the conditions for parallel updates are satisfied. It will typically consist of a few things, depending on your setup:

- Calculating the movements of your project's components, such as updating physics objects.
    - When possible, avoid updating physics for things that don't actually need it. This will result in the most significant reduction of the FParallelAnimationCompletionTask.
- Initiating Animation Notifies.
    - All Notifies should be non-Blueprint based, to avoid calls to the Blueprint Virtual Machine.
    - Notifies should be performed on the **game thread**, as they can affect the animated object's lifetime.
- Interpolation of animation if URO is enabled.
- Blending of curves if Material or Morph Target curves are in use.