Dynamic Paraglider

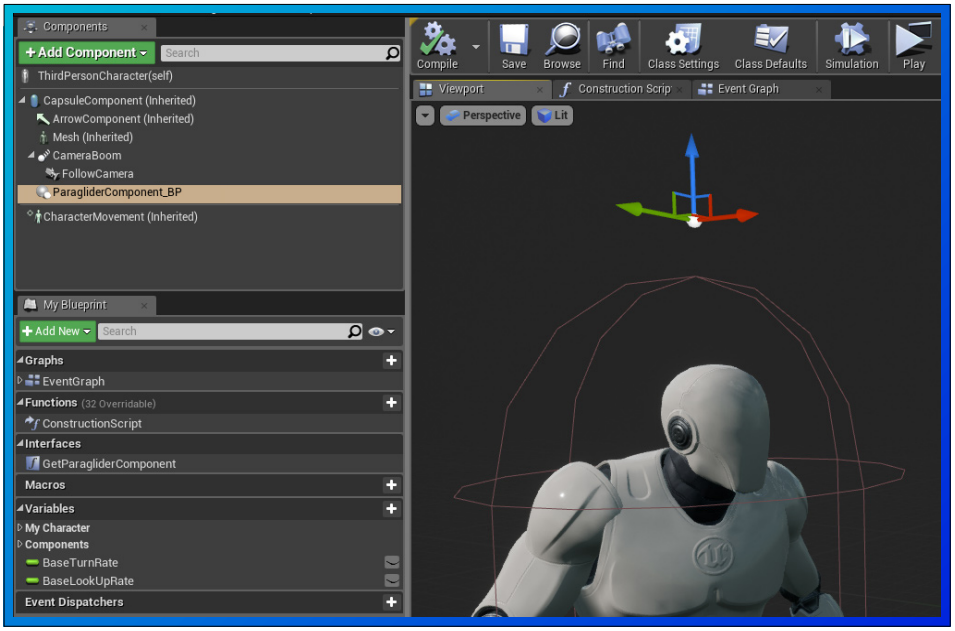
Hello, and thank you for your purchase! This documentation covers the following topics:

- Merging Into Your Project
- Integration with ALS
- Updraft Blueprint
- Creating Child Paraglider BPs
- Customizing The System
- Useful Functions
- Mixamo Retarget Crash

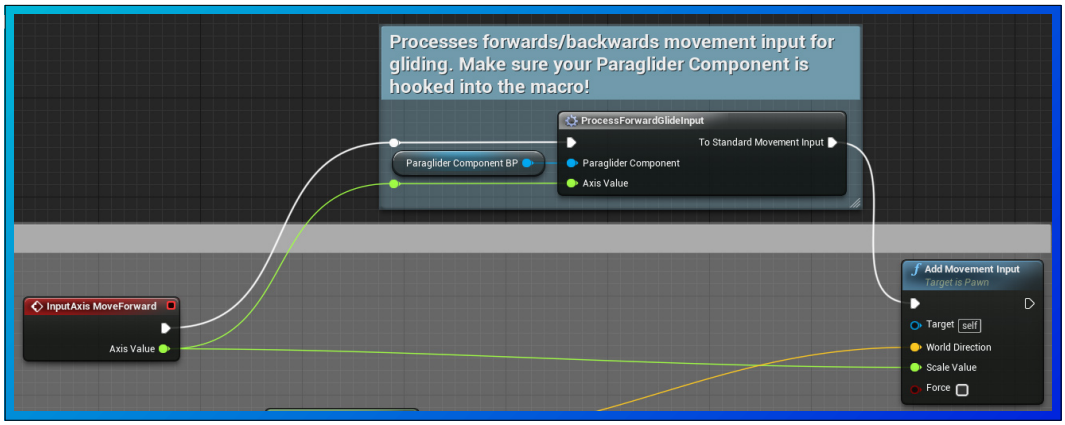
Merging Into Your Project

The following steps will guide you through migrating this system into your project and merging it into an existing character.

1. Open the provided **Dynamic Paraglider** project.
2. Right click on the **DynamicParaglider** folder in the content browser and select **Migrate**. Select the **Content** folder of the project you want to migrate to.
3. Close the **Dynamic Paraglider** project and open your project.
4. Create an **Action Mapping** in your input settings (**Edit → Project Settings → Input**) with the name **ToggleGlide**, and assign it to any key/button you'd like.
5. Open your existing character BP and add the **ParagliderComponent_BP** to it as a child of the root **CapsuleComponent**. (Recommend keeping the Paraglider component's default name to simplify the next steps)
6. Move the component up on the Z-axis so it's a short distance above your character's head, but within arm's reach – the position of this component defines where the Paraglider will be spawned.

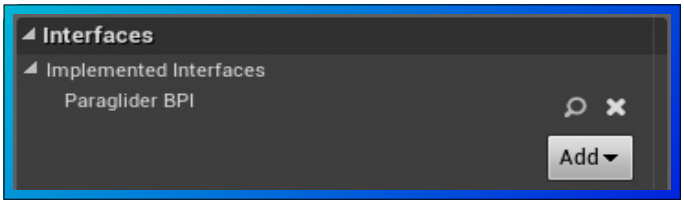


7. Open the provided **ThirdPersonCharacter_BP** blueprint (DynamicParaglider\Blueprints\Characters) and copy the small sections of nodes surrounded by blue comment boxes. There should be three sections.
8. Open your existing character BP and paste these nodes into its Event Graph. Note that two of them should connect to pre-existing character logic – refer to the provided character BP and carefully recreate exactly how they connect. One section connects to InputAxis MoveForward logic, and another connects to InputAxis MoveRight logic.

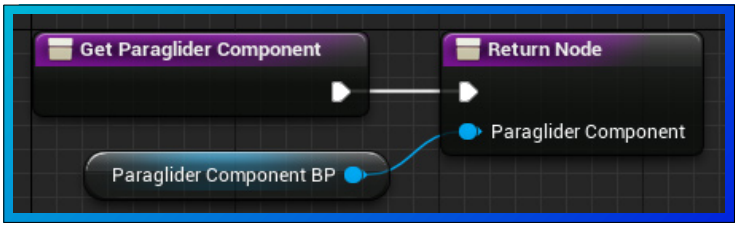


(**NOTE:** Each of these three sections should have a connection to the **ParagliderComponent_BP**. If you kept the component's default name, the connections should automatically be valid. If you changed the component's name, you'll need to replace the invalid references to ones that correctly reference your component.)

9. In your character BP's class settings, add the **Paraglider_BPI** interface.

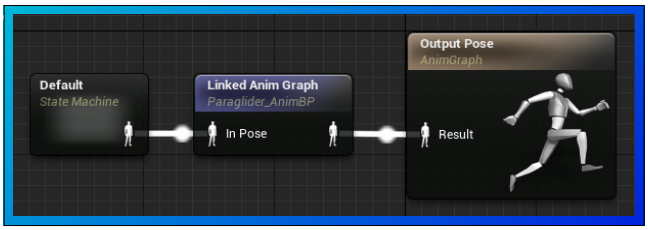


10. On the left side of the character BP, expand the **Interfaces** category and double-click on the **Get Paraglider Component** function. Click on your **ParagliderComponent_BP**, drag a reference into the graph, and plug it into the **Paraglider Component** input on the **Return Node**.



11. Compile and close the character BP.
12. Make sure that your character's skeleton is ready for retargeting – open the skeleton, go to the **Retarget Manager**, and choose the **Humanoid** rig in the **Select Rig** dropdown. Save the skeleton.

13. Right-click on the included **Paraglider_AnimBP** (DynamicParaglider\Animations) and select **Retarget Anim Blueprints → Duplicate Anim Blueprints And Retarget**. Select your skeleton and complete the retarget. NOTE: There is a potential issue here when retargeting to a Mixamo rig, and possibly other rigs that are not based on the UE4 skeleton. If you encounter a crash here, see the section about Mixamo for a work-around.
14. Open the AnimBP you're using for your character. In the main AnimGraph, create a **Linked Anim Graph** node (also called **Sub Anim Instance** in older engine versions). Set its **Instance Class** to the retargeted **Paraglider_AnimBP**.
15. Plug your existing state machine / final pose result into the **In Pose** on the **Linked Anim Graph**, then connect its output to the **Output Pose**.

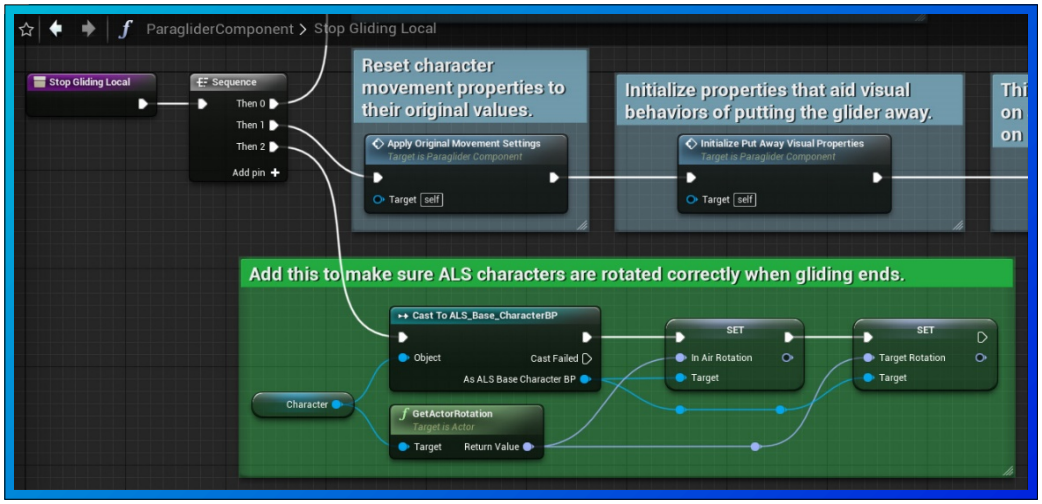


16. Compile and save the AnimBP, and you're done! I recommend running the game and testing the glide. If the glider is too high and the character's arms ever over-extend while attempting to reach the handles, or if the glider is too low, you can adjust the **ParagliderComponent_BP**'s Z-location inside your character BP.

Integration With ALS

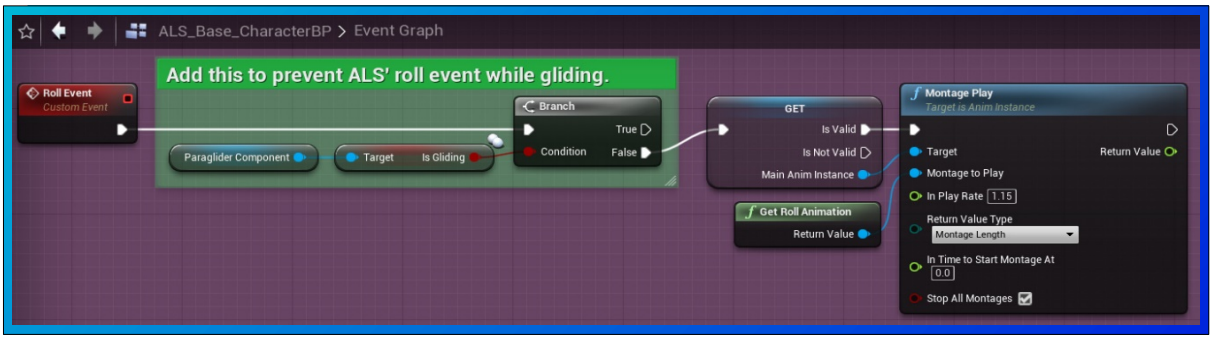
If you're using Advanced Locomotion System, there are a few extra steps to take for best results. After completing the previous section, I recommend doing the following:

1. When gliding ends, we want to inform ALS of the desired character rotation. Open the **ParagliderComponent_BP** (DynamicParaglider\Components), open the **StopGlidingLocal** function, and add a new pin to the **Sequence** node. From this **Sequence** pin, we want to cast the character to the ALS character BP and set its **InAirRotation** and **TargetRotation** variables to the character's current rotation.



2. Next, we want ALS to stop the character from gliding when they mantle a ledge. Open the ALS character BP, open its **MantleStart** function, add a new pin to its **Sequence** node, and call the **ParagliderComponent_BP**'s **StopGliding** function.

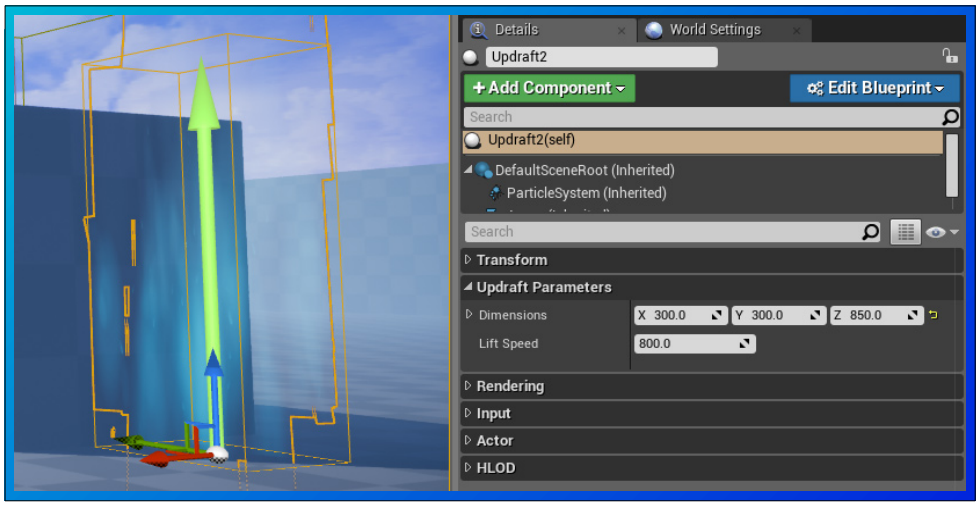
3. Finally, we want to prevent ALS' dodge roll while gliding. Locate the **RollEvent** event in the ALS character's Event Graph and add a branch that only allows the event to proceed if the **ParagliderComponent_BP**'s **IsGliding** boolean is false.



4. Compile and save the BPs, and you're done!

Updraft Blueprint

The **Updraft BP** (DynamicParaglider\Blueprints\Updraft) can be used to create an area of upwards wind that lifts a gliding player. After placing the BP in your level, you can find parameters that control its **Lift Speed** and **Dimensions** in its **Details** panel.



Creating Child Paraglider BPs

You can create child **Paraglider_BP**s by right-clicking on the **Paraglider_BP** (DynamicParaglider\Blueprints\Paragliders) and selecting **Create Child Blueprint Class**. I recommend doing this if you would like to create multiple gliders with different appearances and/or flight behaviors. You can learn how to customize your child BP (or the original BP if you prefer) in the next section.

Customizing The System

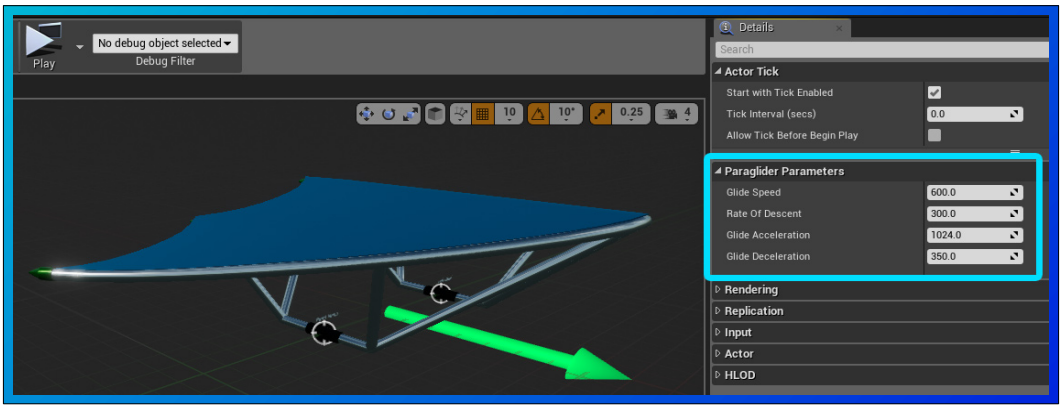
Glider Location

The location of the glider model depends on how you position the **ParagliderComponent_BP** inside your character BP. This is done so that there is enough flexibility to accommodate character and paraglider models that have different proportions. If you ever see your character's arms over-extend, or fail to reach the handles on the glider, then lower this component until the glider is close enough for the hands to reach. I recommend leaving the component's X and Y locations at 0.

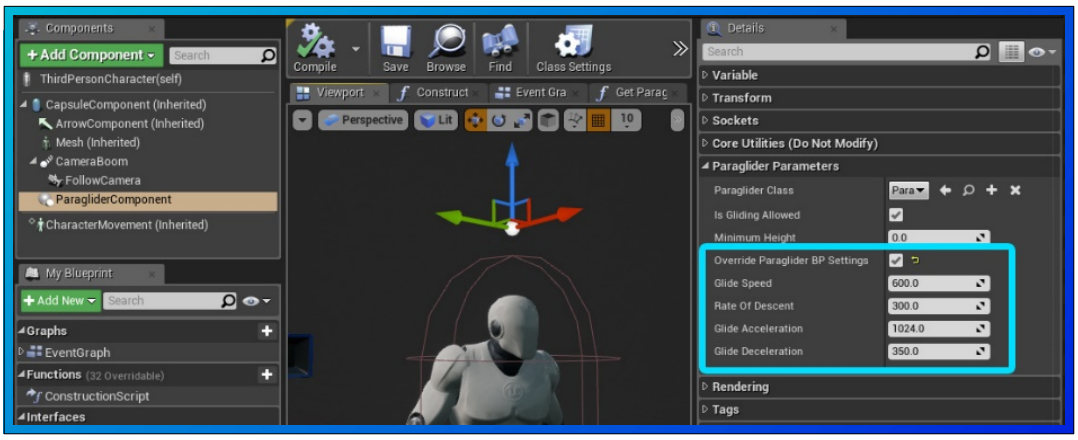
Glider Behavior

There are two ways to define your gliding behavior:

Method 1: By default, gliding parameters are defined on the **Paraglider_BP** (DynamicParaglider\Blueprints\Paragliders). This method allows you to have multiple glider BPs with unique flight behaviors. These settings can be changed by opening the **Paraglider_BP** and modifying them in its Class Defaults.

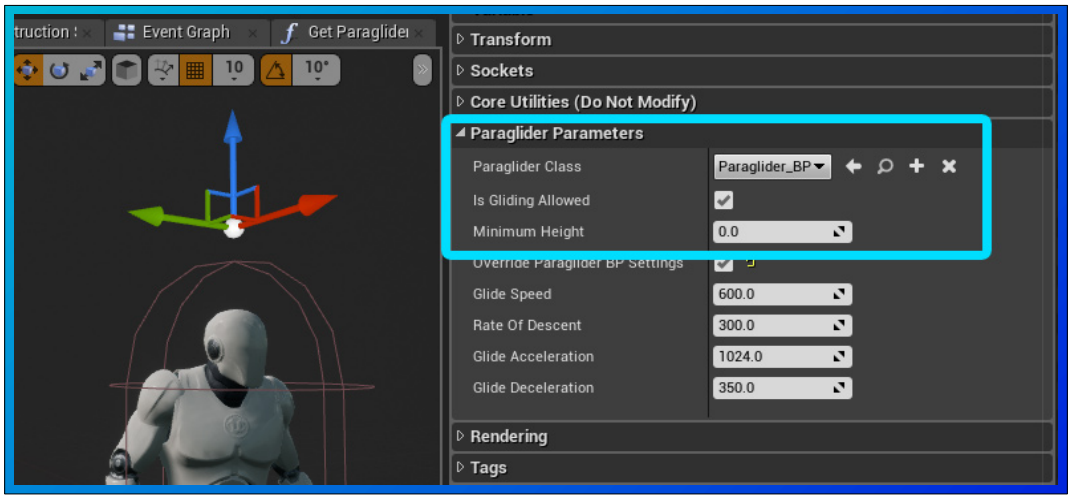


Method 2: The **ParagliderComponent_BP** in your character BP has an option to **"Override Paraglider BP Settings"** – if you enable this, it allows you to define gliding parameters on the component itself. Instead of using the **Paraglider_BP**'s settings, it uses the component's settings, causing all gliders to behave the same regardless of what configuration may be on the **Paraglider_BP** itself.



Component Settings

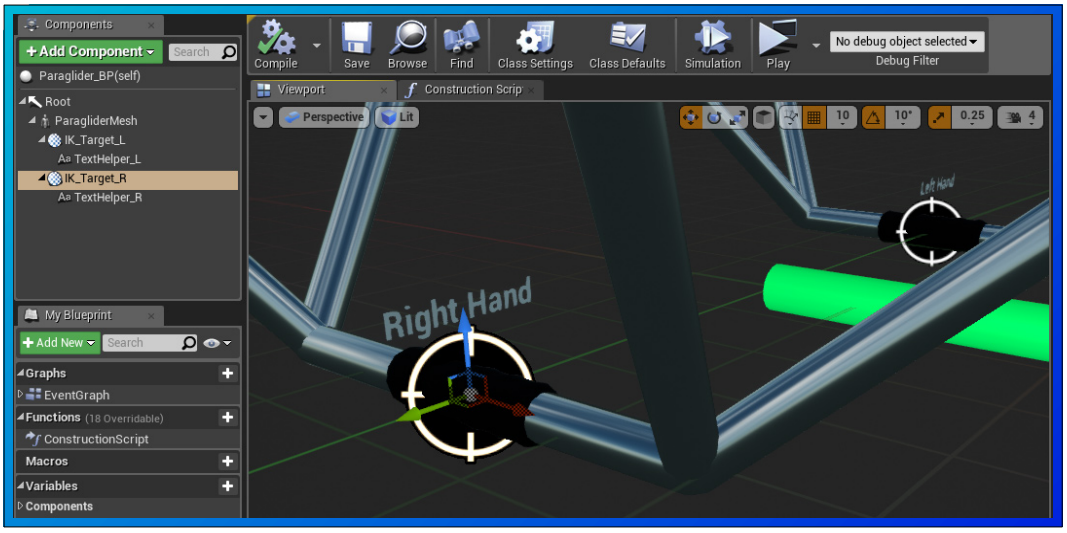
The **ParagliderComponent_BP** has some additional settings that define behavior at the character-level, including which **Paraglider_BP** class to use, the minimum height required to start gliding, and whether or not gliding is allowed.



Replacing The Mesh

If you would like to use a different mesh, you can do so by opening the **Paraglider_BP** and changing which mesh the **ParagliderMesh** component uses. If the handles are in a different location from the original mesh, you can move the **IK_Target_L** and **IK_Target_R** components to the new handles so that the animations know where the hands should be.

I recommend positioning the mesh so that its handles are at or near 0 on the Z and X axes.



Useful Functions

The following **ParagliderComponent_BP** functions may be useful if you would like to add or expand functionality for starting/stopping gliding, restricting when gliding is possible, or changing what glider is equipped.

- **SetIsGlidingAllowed** – Can be called by either host or client. Enables or disables the character's ability to glide. If the player is already gliding when it is disabled, the glide is immediately ended.
- **SetParagliderClass** – Can be called by either host or client. This can be used to change what Paraglider the character is using. When called, it destroys the previous glider (if one is present), creates a new one of the desired class, then re-initializes all necessary data.
- **StartGliding**, **StopGliding**, and **ToggleGliding** – Should be called by the owner of the character (client or host). As the names suggest, these functions can start, stop, or toggle gliding.

Mixamo Retarget Crash

For some reason, UE4 sometimes doesn't like retargeting to Mixamo rigs when the animations use certain additive settings. This results in an editor crash. If you're encountering this issue with a Mixamo rig (or possibly another rig that isn't based on the default UE4 skeleton), follow these steps as a work-around.

1. Select all animation assets that have **Additive** or **AdditiveBase** in the name (there should be 4).
2. Right click them and go to **Asset Actions** → **Bulk Edit via Property Matrix**.
3. Expand the **AdditiveSettings** category on the right side and change the **Base Pose Type** to **Skeleton Reference Pose**.
4. Close the **Property Matrix**, then retarget the AnimBP like normal.
5. Select the retargeted versions of those same 4 animations and re-open the **Property Matrix**.
6. Change the **Base Pose Type** back to **Selected Animation Frame**
7. Done!