Takedown Animations

Documentation

Documentation version: 1.0

(ENG)

Table of contents

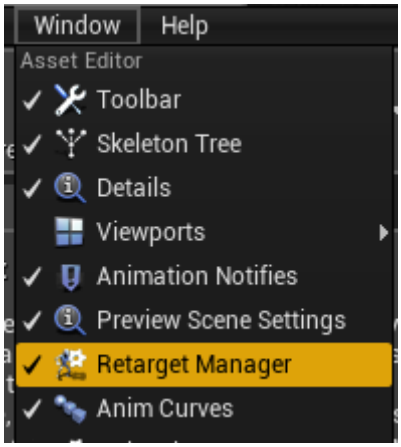
1. Retarget Animations.

1.1 Unreal Engine 4.

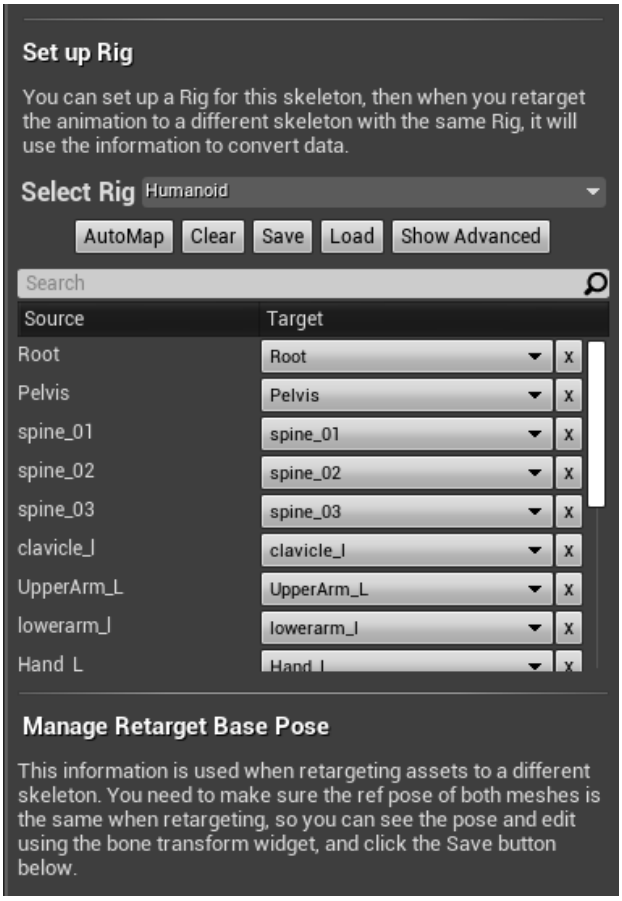
In Unreal Engine 4, to transfer animations from the product to the skeleton of your character, you will need to do a few simple actions.

As an example, for retargeting animations, we will take the standard "**UE4_Mannequin_Skeleton**" from the "**ThirdPerson**" template provided by the engine.

1. First, open the skeleton of your character and the skeleton of the character from the product.
2. Find the "**Window**" menu and enable "**Retarget Manager**" in both skeletons.



3. Go to the "**Retarget Manager**" tab and find the "**Select Rig**" drop-down menu there. Select "**Select Humanoid Rig**" in it. In the list that appears, check and configure the character's bones so that they are similar to the "**Source**" column (they should be similar not so much in names as in location and anatomy).

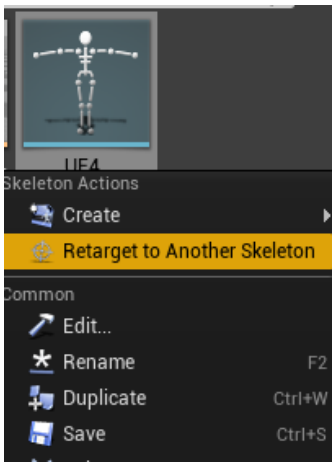


4. Save the skeleton.
5. Repeat these steps with another skeleton.

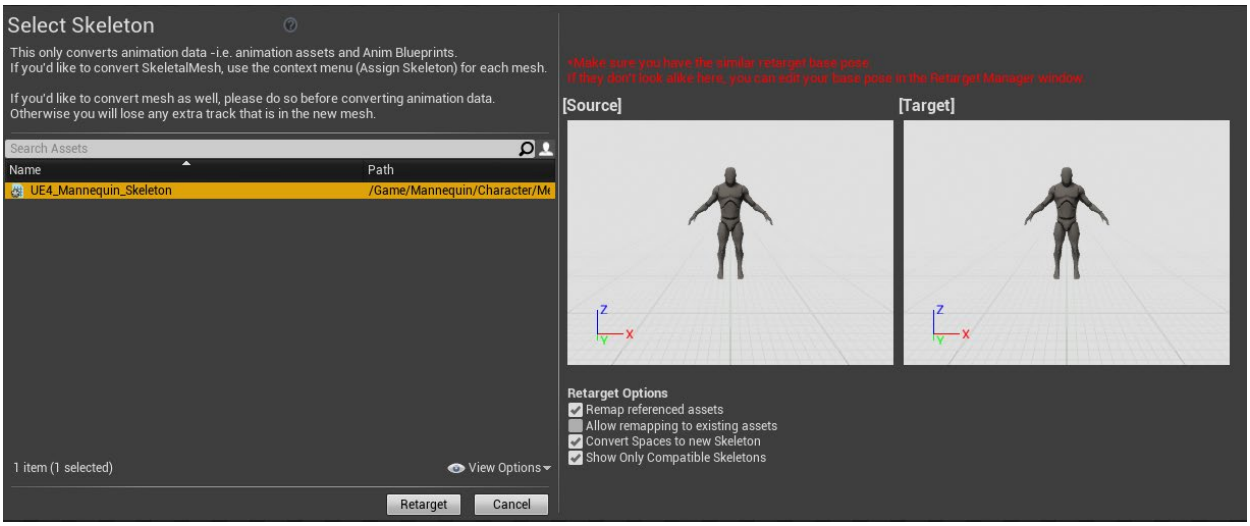
Before proceeding to the transfer of animations, be sure to check that the pose of your character and the character from the product is the same. If the poses are different, then correct them, otherwise the animation will be moved incorrectly.

Now that all the preparatory stages are completed, you can safely proceed to retargeting animations on your character's skeleton.

1. Find the skeleton of the character from the product, right-click on it and select “**Retarget to Another Skeleton**” in the drop-down menu.



2. In the window that opens, in the list on the left, find and select the skeleton of your character, then click the “**Retarget**” button.



Congratulations! Animations from the product have been transferred to your character's skeleton!

Also, please note that when retargeting one skeleton to another, sockets that were present in the skeleton of the character from the product will be created in the skeleton of your character.

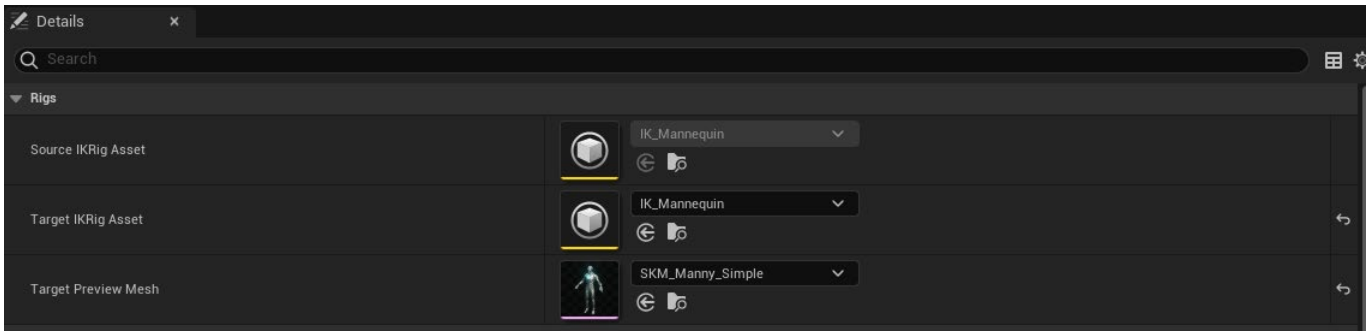
1.2 Unreal Engine 5.

If we compare the Unreal Engine 4 retargeting with Unreal Engine 5, then in the new version of the engine, the retargeting function has undergone major changes, therefore, the method from the old version in Unreal engine 5 will not work.

In order to transfer animations to the skeleton of your character, you will need to perform several actions. As an example, for retargeting animations, we will take the standard "**SK_Mannequin**" from the "**ThirdPerson**" template provided by the engine.

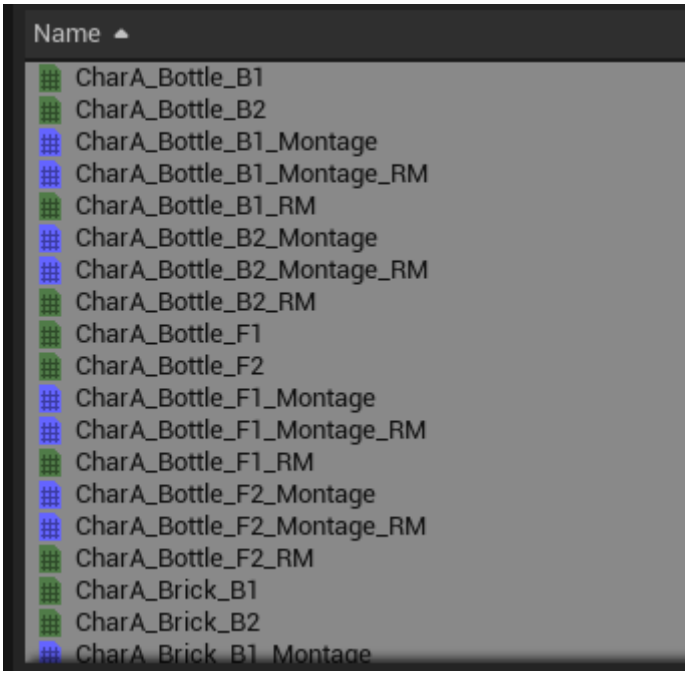
1. In the product folder "**Takedown Animations**" go to "**Example – Mannequins – Rigs**" and open "**RTG_Mannequin**".

2. Open the "**Details**" panel and insert your character's IKRig into the "**Target IKRig Asset**", then select your character's Mesh in the "**Target Preview Mesh**".



3. Go to the "**Chain Mapping**" tab and check "**Target Chain**" with "**Source Chain**".

4. Check the pose of your character and the character from the product, they should be the same. If the poses are different, then correct them, otherwise the animation will be transferred incorrectly.
5. Open the "**Asset Browser**" tab and select all animations and montages from the product.



6. Click the "**Export Selected Animations**" button, in the window that appears, select the folder to which you want to export everything and click "**OK**".

Congratulations! Animations from the product have been transferred to your character's skeleton!

Please note that unlike the retargeting in Unreal Engine 4, in the new version of the engine you will have to create and configure all the sockets in your character's skeleton that can be found in the skeleton of the character from the product. In addition, you will also have to manually sort all exported animations into your folders, as well as rename them if necessary.

2. Transferring test logic to another blueprint.

2.1. About test logic.

Before you want to transfer all or part of the logic to your Blueprint, then you should know that during the creation of the test logic, our plans did not include the task of writing flexible code that could be easily implemented into any project. First of all, the test logic was created to demonstrate the capabilities of the product, as well as so that the buyer could take something useful for himself in future developments.

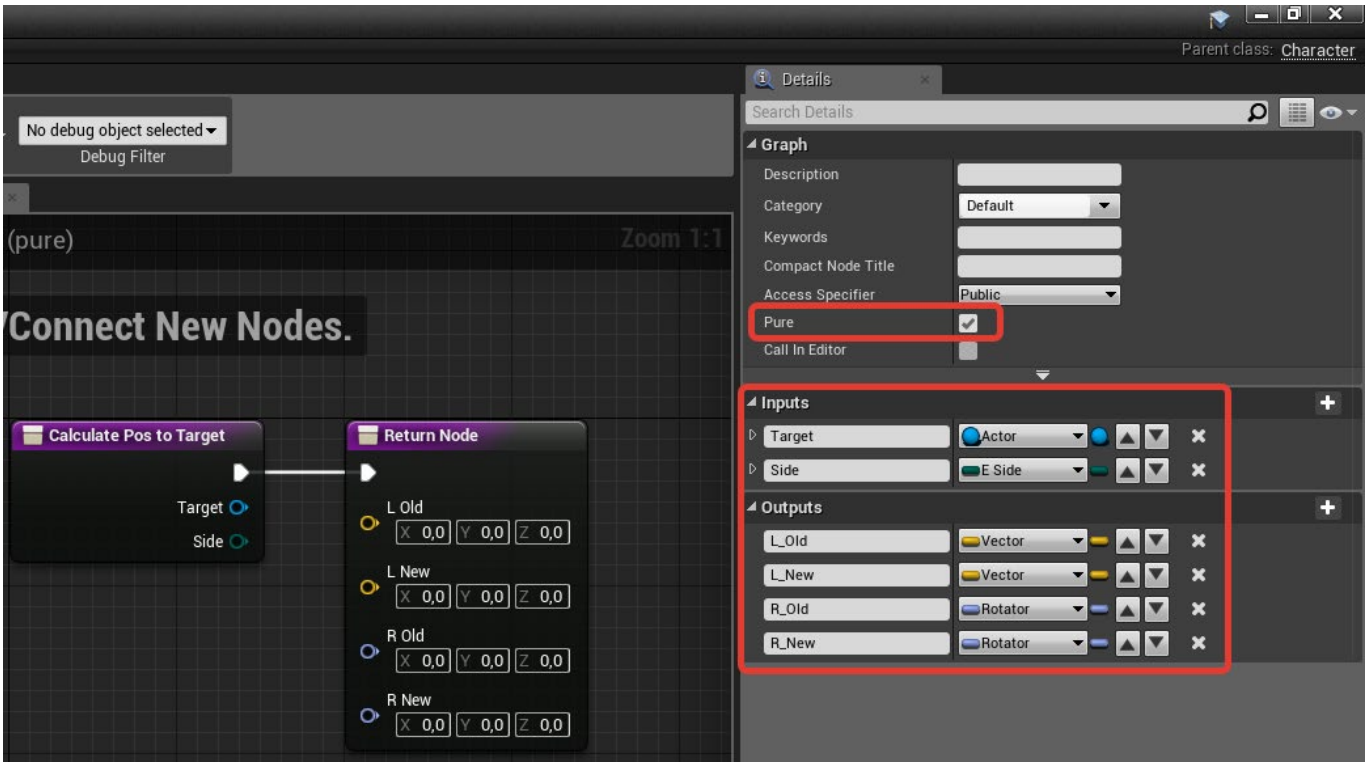
2.2. Transferring logic.

2.2.1. Transferring logic in a character.

In order to transfer the test logic to your character's blueprint, you will need to perform several actions. As an example, for transferring logic, we will take the standard "**ThirdPersonCharacter**" from the "**ThirdPerson**" template provided by the engine.

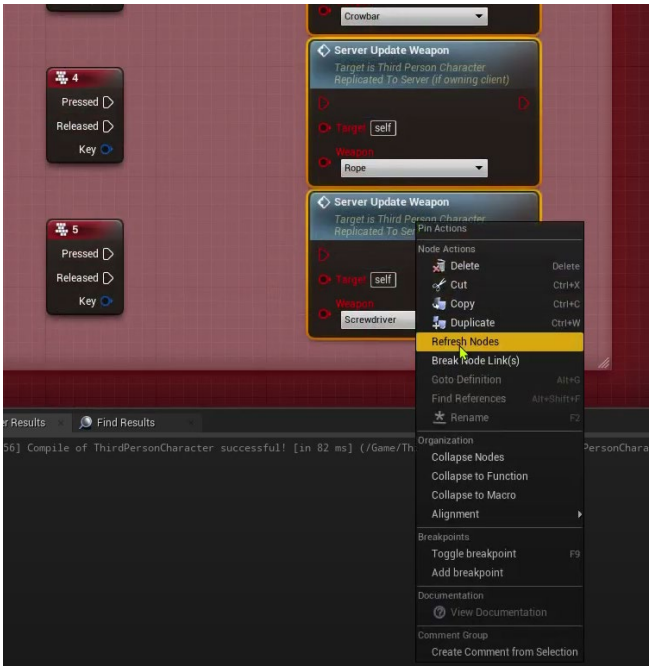
1. Open the blueprint of your character and the character from the product.
2. Go to your character's blueprint and select Mesh in the "**Components**" tab.
3. Open the "**Details**" tab and find the "**Collision**" list. In the "**Collision Presets**", select "**Custom...**" and in the "**Object Type**" line, set "**World Dynamic**" (This is necessary so that the weapon stuck in the target remains in it. You can learn more about this in the "**W_Screwdriver**" blueprint).
4. Transfer all variables from the product character (from the Non Edit and Edit tabs) to your character's blueprint. Configure them the same way as it is done in the test character. Change the "**Target**" variable type to your character's blueprint.
5. In your character's blueprint, click on "**Class Settings**", and find the "**Details**" tab, find the "**Interfaces**" list in it, click "**Add**" and add the "**BI_TDA**" interface. Compile and save the blueprint of your character.

6. In the “**My Blueprint**” tab, you have a list of “**Interfaces**”, fill it in the same way as it was done in the test character.
7. Transfer all the functions from the product character to your character's blueprint (You don't need to transfer the code from the functions, because now it will cause a lot of errors. Create the necessary inputs and outputs in each function as in the test character and check whether this function should be “**Pure**”).

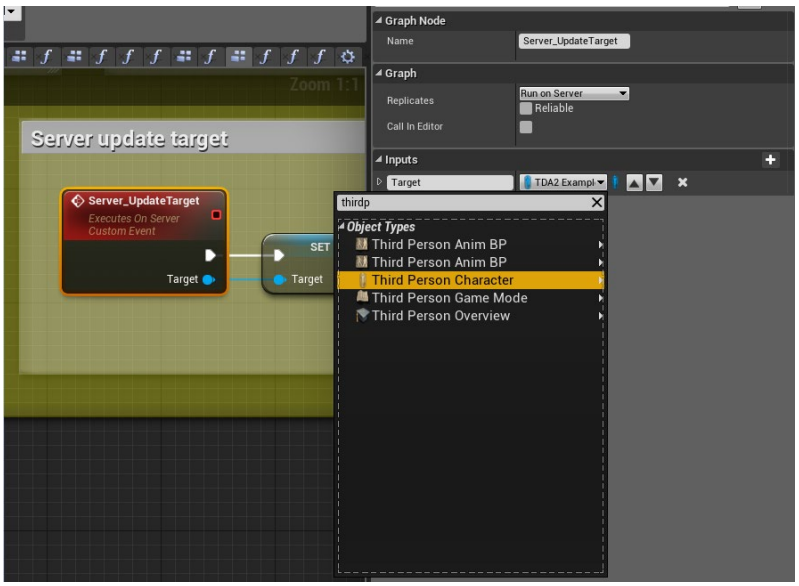


8. Transfer and fill in all macros from the product character to your character's blueprint (In the “**M_BlockInput**” macro, do not transfer the Event “**Client Show Message**”, because it does not exist in your character's blueprint yet).
9. Open the blueprint of the test character from the product and go to “**EventGraph**”. Select all logic except “**Default Input Events**”, copy it and transfer it to your character's Blueprint. Compile and save the blueprint of your character (now you can transfer the Event “**Client Show Message**” to the macro “**M_BlockInput**”). A large number of errors have appeared in your character's blueprint. It is ok. Now your task is to fix them.

10. Update the event links by right-clicking on it and selecting "**Refresh Nodes**".



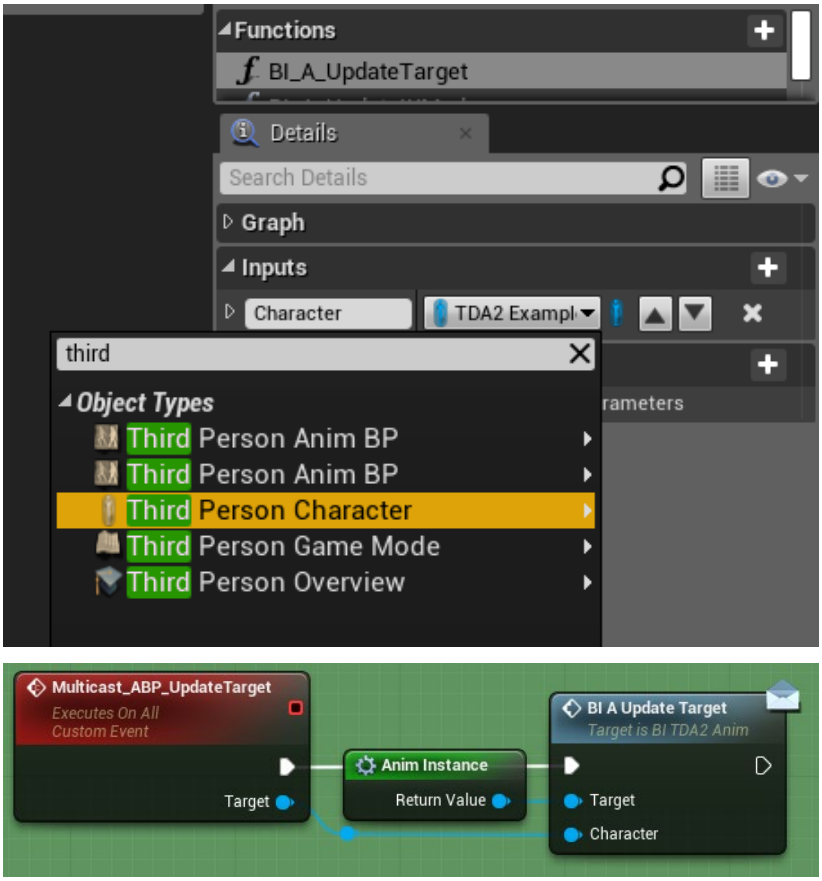
11. Since macros cannot be moved to another blueprint, you will have to manually add them there, as is done in the test character's blueprint.
12. In all events, change the variable type to your character's blueprint and fix the errors with the "**Target**" variables connected.



13. In the "**EventGrpaph**" of your character, find all the "**CollapsedGraph**", go to them and fix the errors (reconnect the nodes or recreate them. In TimeLine, you will have

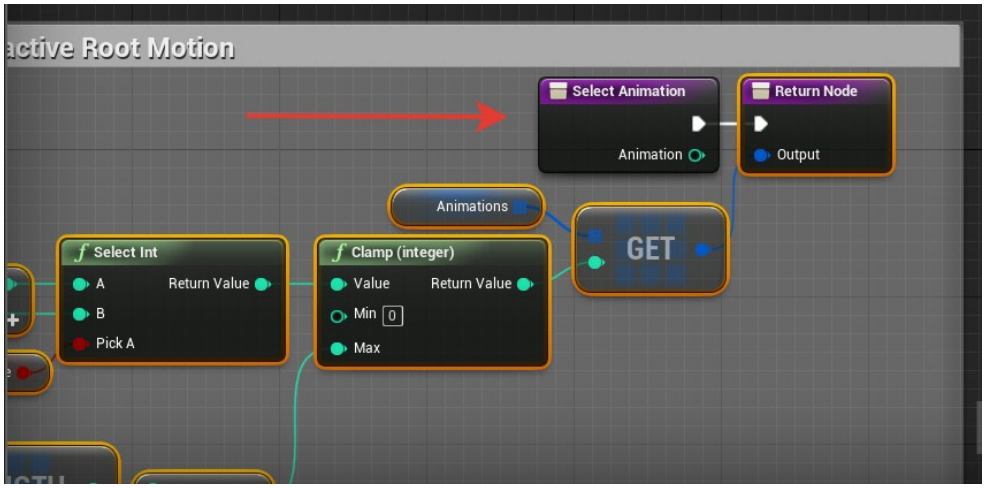
to manually create a curve and adjust it the way it was done in the test character's blueprint).

14. When compiling the project, you will have one error left, which will be associated with a type mismatch in the "**Multicast_ABP_UpdateTarget**" event. To solve it, collapse the blueprint of your character and find the interface "**BI_TDA_Anim**" in the explorer. Open it, and in the "**BI_A_UpdateTarget**" function, change the type of the variable to the type of your character. Compile and save the interface. Go back to your character's blueprint, compile and reconnect the "**Target**" link in "**Multicast_ABP_UpdateTarget**" to the "**Character**" link in "**BI_A_UpdateTarget**". Compile and save your blueprint.

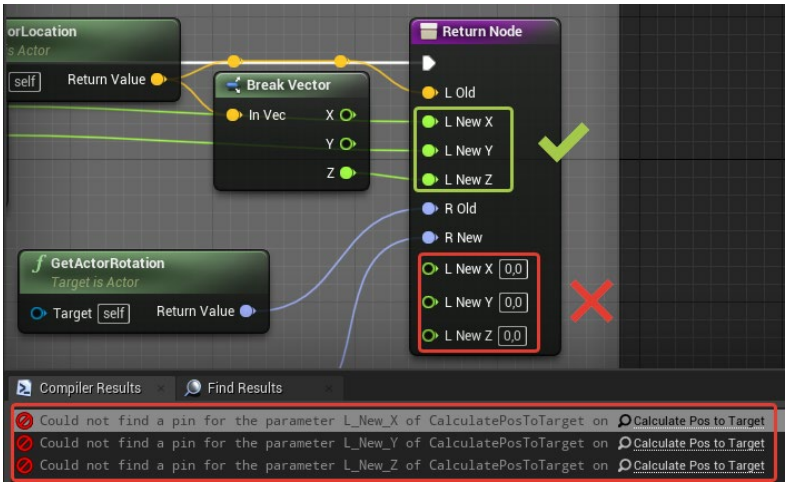


15. Transfer the logic of each function from the test character of the product to the corresponding functions in your character, and connect it the same way as it is done

in the test character from the product (during the selection of logic, do not highlight the beginning of the function, so as not to create errors when copying in the future).



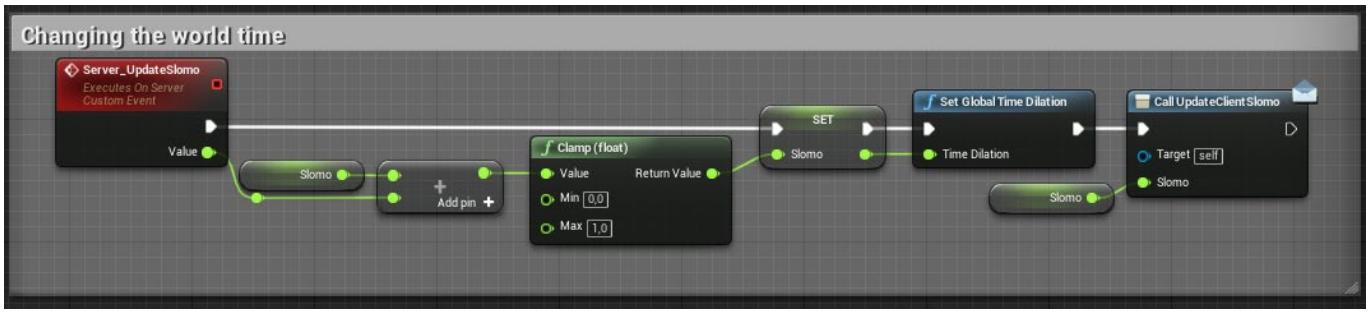
16. Compile and save your blueprint.
17. Fix the errors in the functions that occurred when compiling your blueprint. During the correction, rely on the test character from the product (reconnect nodes, update them, change casts to your character, create local variables, etc.). In the "**CalculatePoseToTarget**" function, you may have a problem with three newly created Float outputs. Remove them and reconnect the nodes to the correct outputs again. At the end of the correction process, you should not have errors during compilation.



2.2.2. Transferring logic in game mode.

Transferring logic to GameMode is only responsible for Slow-motion during the game. If you don't need this feature, then you can just skip this point.

1. Open the blueprint of your "**GameMode**", add and configure variables as in the test "**GameMode**" from the product.
2. Add a dispatcher and configure it the same way as in the test "**GameMode**" from the product.
3. Copy the logic from the test "**GameMode**" to your "**GameMode**".



4. Make sure that your Game mode is selected in the "**GameMode**" variable type in your character's blueprint. Also check that the server events that are responsible for installing "**GameMode**" have the correct casts and links to your Game mode.

2.2.3. Transferring logic in UI.

1. Go to the UI folder and open "**W_UI**".
2. Change the type of the "**Owner**" variable to the type of your character.
3. Find the "**Event Construct**" and change the castes in it to your character.
4. Compile the widget.
5. Fix the errors that appeared during compilation (change the references to objects, because the old ones are invalid due to the change of the variable type to the blueprint of your character).

2.2.4. Transferring logic in weapons.

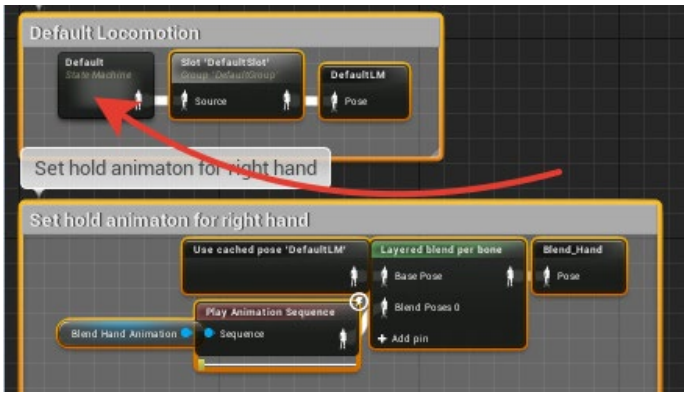
1. Go to the Weapons folder and open "**BP_ParentWeapon**".
2. Change the type of the variable **"C_Owner**" to the type of your character.
3. Find "**Event BeginPlay**" and change the cast in it to your character.
4. Compile the blueprint.
5. Fix the errors that appeared during compilation (reconnect the nodes, update them, change the references to objects, because the old ones are invalid due to the change of the variable type to the blueprint of your character).
6. Repeat steps 4 and 5 for the remaining blueprints in the Weapons folder that contain compilation errors.

2.2.5. Transferring logic in Animation Blueprint.

1. Open the Animation Blueprint for the test character from the product, and also open the Animation Blueprint of your character.
2. In your character's blueprint, click on "**Class Settings**", and find the "**Details**" tab, find the "**Interfaces**" list in it, click "**Add**" and add the "**BI_TDA_Anim**" interface. Compile and save your character's Animation Blueprint.
3. Transfer all variables from the Animation Blueprint of the product character (from the Non Edit tab) to the Animation Blueprint of your character. Configure them the same way as it is done in the test character. Change the variable type "**Target**" and "**Owner**" to your character's blueprint.
4. In the "**My Blueprint**" tab, you have a list of "**Interfaces**", fill it in the same way as it was done in the test character.
5. Transfer and fill in all macros from the Animation Blueprint of the product character to the Animation Blueprint of your character.
6. Transfer and fill in all the functions from the Animation Blueprint of the product character to the Animation Blueprint of your character (during the selection of logic,

do not highlight the beginning of the function, so as not to create errors when copying in the future).

7. Compile the Animation Blueprint and fix the errors that appeared during compilation (reconnect the nodes, update them, insert macros, change the references to objects, because the old ones are invalid due to the change of the variable type to the blueprint of your character).
8. Open the "**EventGraph**" in the Animation Blueprint of your character and the test character from the product.
9. Transfer all the logic from the Animation Blueprint of the test character to the Animation Blueprint of your character.
10. Compile the Animation Blueprint and fix the errors that appeared during compilation (reconnect the nodes, update them, change the casts to your character, etc.).
11. Open the "**AnimGraph**" in the Animation Blueprint of your character and the test character from the product.
12. Select and copy all the logic in the "**AnimGraph**" in the Animation Blueprint of the test character except for your "**StateMachine**" and paste it into the Animation Blueprint of your character.



13. Connect your "**StateMachine**" to the "**Slot 'DefaultSlot'**" and the end of the logic to the "**FinalAnimationPose**" in the same way as it is done in the Animation Blueprint of the test character.
14. Compile and save the project.

2.3. Correction of errors when running the simulation.

During the start of the project simulation, you will see the "**Blueprint Compilation Errors**" window.

To correct these errors, you will need to perform a few simple steps.

1. Open the blueprint of the test character from the product and go to "**EventGraph**".
2. Find the "**Server_SlomoUpdate**" event and remove the logic in it that causes the error (If you migrated Slow-motion).
3. Find the "**Multicast_ABP_UpdateTarget**" event and disable the execution thread in it, as well as the "**Target**" link.
4. Compile the blueprint.
5. Open the Animation Blueprint of the test character from the product and go to "**EventGraph**".
6. Find the "**BI_A_UpdateTarget**" event and disable the execution thread in it, as well as the "**Character**" link.
7. Compile the blueprint.

2.4. End of logic transfer.

Congratulations! You have successfully transferred all the test logic into your character!

If you still have any compilation errors, then you may have missed something and you should check all the migration steps again according to this documentation.

If the problem persists, and there are no missed steps in the documentation, then perhaps your problem is unique, and you will have to find a solution to this problem yourself.

Once again, we would like to repeat that during the creation of the test logic, our plans did not include the task of writing flexible code that could be easily implemented into any project. First of all, the test logic was created to demonstrate the capabilities of the product, as well as so that the buyer could take something useful for himself in future developments.