Dynamic Binding[1]

Dynamic Binding provides custom Blueprints logic that picks which object to possess in the level or which to spawn.

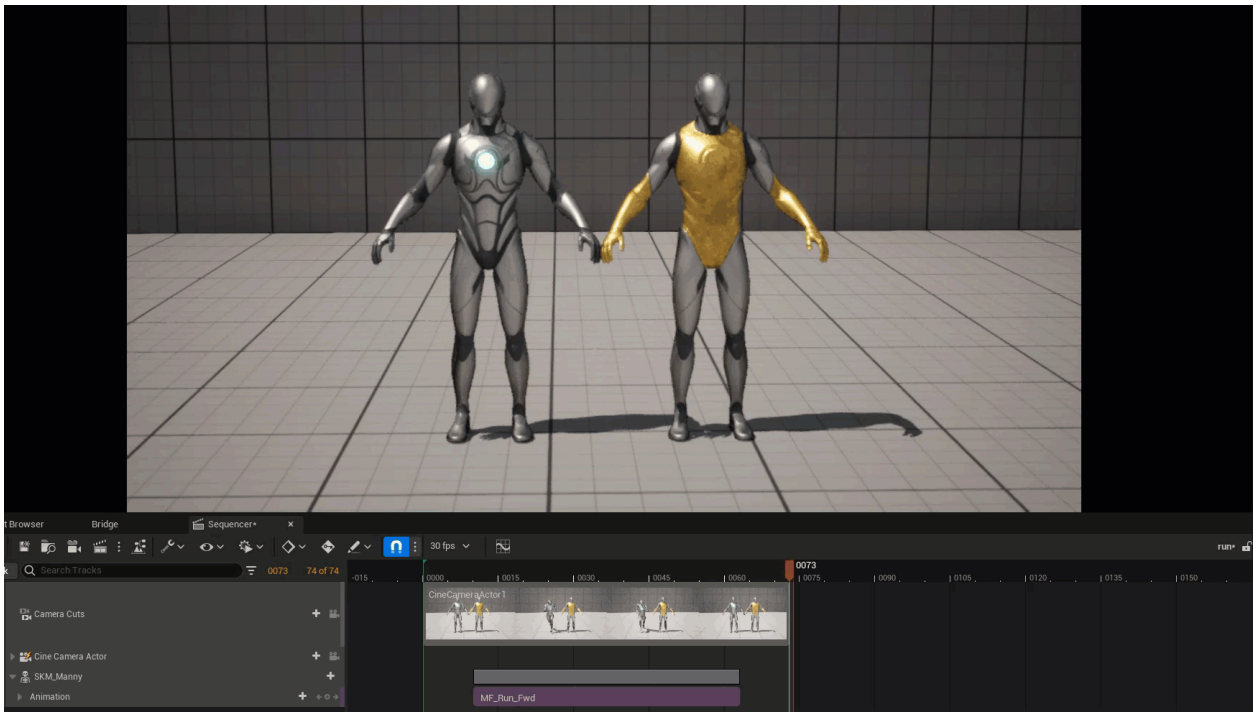
During gameplay events or UI animations, you can animate different objects that the player sees during a cinematic experience or while interacting with the UI. **Dynamic Possession** provides custom Blueprint logic that picks which object to possess in the level or which to spawn. This feature also provides a quick binding option for common resolution logic such as dynamically binding the **Player Pawn**. **Tracks** and **Properties** can be bound to the newly possessed or spawned **actor** and have the same effect as the original bound object.

With Dynamic Binding in Sequencer, you can:

- Define an object in the level to be dynamically possessed with custom **Blueprint** logic. This logic is implemented within the **Level Sequence Director Blueprint**. For UMG, it is implemented within the UMG Blueprint graph.
- Define dynamically spawned objects to be bound with custom Blueprint logic.
- Quickly define a binding to the player controller.
- Have **Tracks** and **Properties** that are compatible to affect the same tracks and properties when dynamically bound to a new object.
- Define a widget in **Unreal Motion Graphics** (**UMG**) to dynamically possess custom Blueprint logic.

In short, you can override a possessable or spawned actor using Dynamic Binding.

[1] Note: this is a static PDF from the original document. Any animated GIFs will appear static.



Dynamic Binding example. The sequence for Manny is overridden and the sequence animation is bound to "Goldie".

Use Case

Let's say you have a fixed player character that matches prepared cinematics. You want to blend between gameplay and cinematics with this character. When creating a cinematic, you typically create a spawnable but during runtime, you don't want to spawn the character as you want to give control back to the player. You would typically grab the player pawn and let the binding take over. For example, this happens in RPG games with lots of missions with a lot of NPCs that can join your mission based on the choices you made with your character. At editing time, you don't know who will join your character. This makes creating cinematics difficult.

Prerequisites

- You have an understanding of **Sequencer** and its **Interface**.
- You know how to create and use **Tracks**.
- Understanding of **Possessables and Spawnables**.
- Understanding of **Blueprints**.

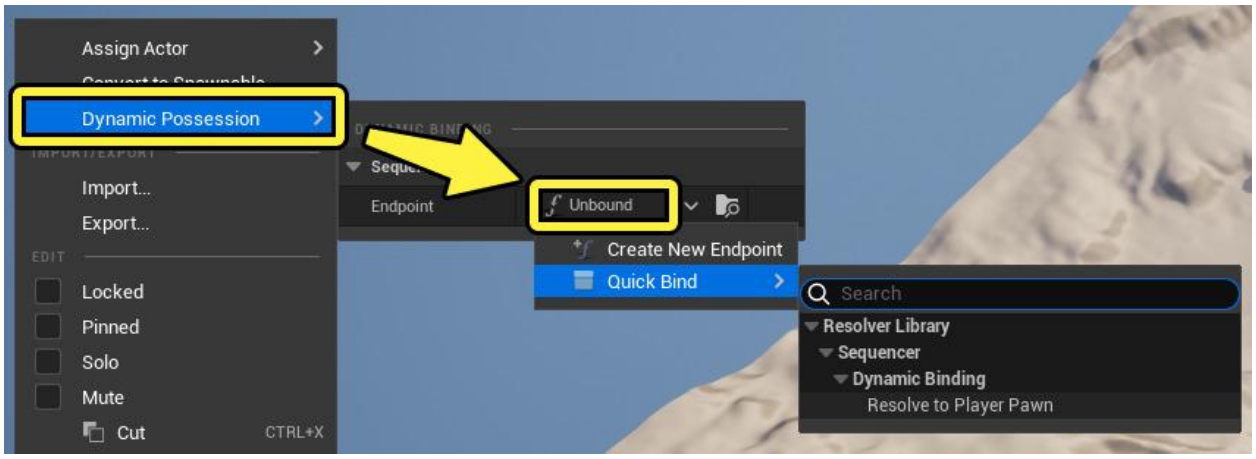
Create a Dynamic Binding

Right-clicking an actor in the **Hierarchy Tree** provides you with an option to dynamically bind it to a new endpoint, to a function that resolves to the **Player Pawn**, or to an existing dynamic binding. The latter two options, found under **Quick Bind**, are similar to how Event Track works.

Once dynamically bound, the actor icon displays a blue pip overlay in the bottom left.

1. Cube is dynamically possessed.
2. Cylinder is not possessed.

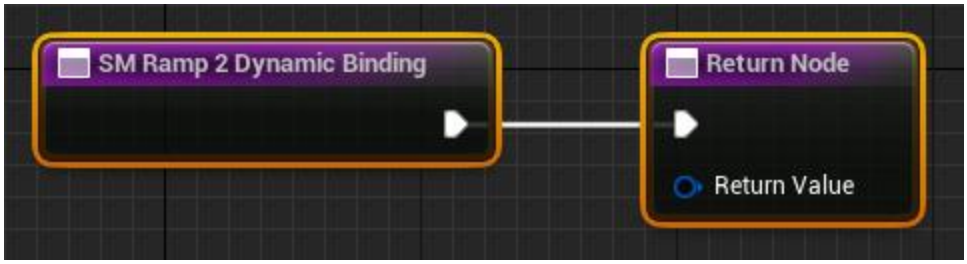
The **Dynamic Binding** menu option includes a way to **Create New Endpoint** or to bind the actor to a solver via **Quick Bind**. From the **Dynamic Possession > Endpoint (unbound)** menu, you can select to create a **New Endpoint** or use the **Quick Bind** feature. These options are described in detail below.



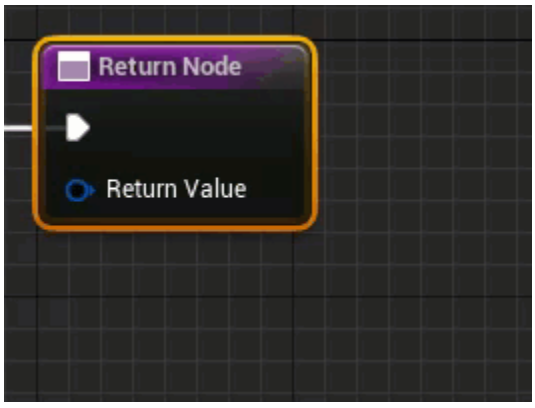
Once an actor has been dynamically bound, UE opens up the **Level Sequence Director Blueprint**.

Create New Endpoint

Creating a new dynamic endpoint creates a pair of custom Blueprint nodes (**binding node** and **Return Node**) where you can add your own logic for how the binding happens. From there, you can provide a returning object that is bound via a possessable or spawnable.

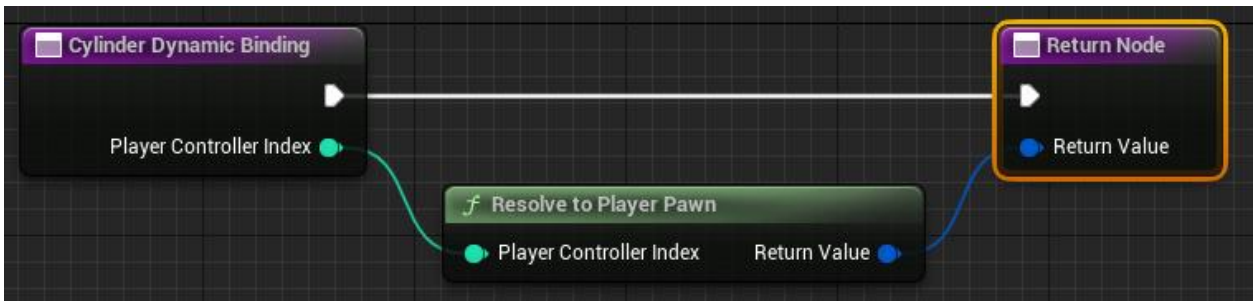


The **Return Node** contains features that a dynamically bound actor can be either possessed or spawned and what other asset to replace it. Right-click the **blue input pin** for **Return Value Object** and select **Split Struct Pin**. Once split, you have options to select which asset is returned from the **Return Value Object** dropdown menu and if that asset is possessed or spawned. By default, the value in **Return Value Is Possessed Object** is checked, which means the asset is **possessed**. Unchecking the **Return Value Is Possessed Object** returns the asset as a **spawned actor**.



Quick Bind

With the **Quick Bind** method, you can select **Resolve to Player Pawn**. This creates an endpoint to bind the assigned binding to the player pawn and adds the logic in the **Level Sequence Director Blueprint** for the dynamically bound actor to call.



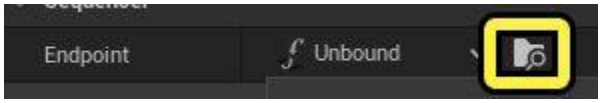
Existing Dynamic Bindings

If you have an existing dynamic binding, it can be used again to bind other actors.

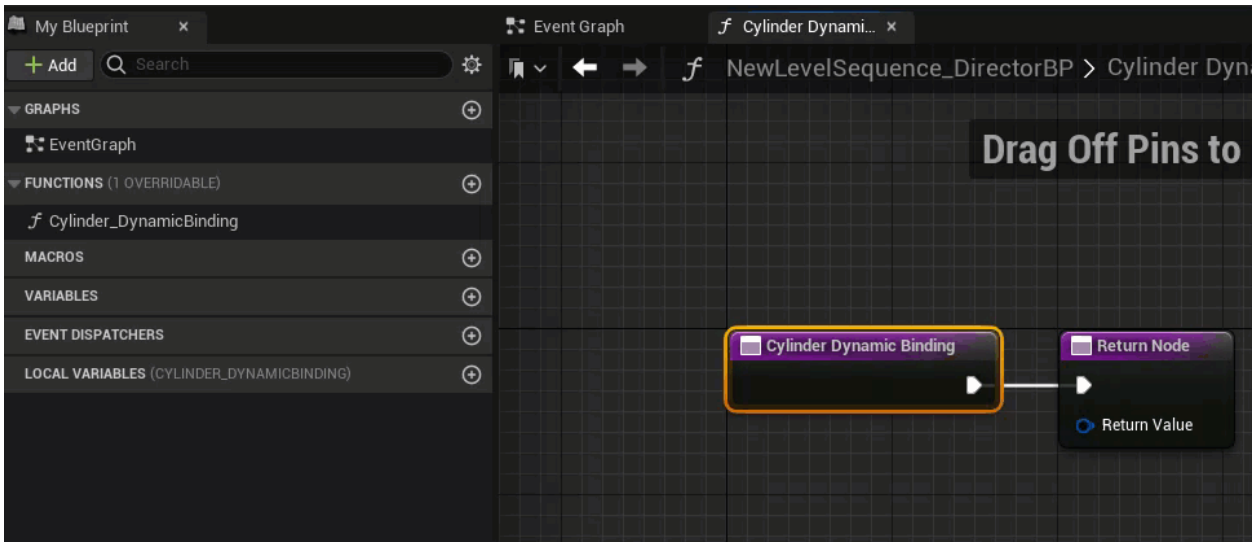
1. Right-click your actor and from the **Quick Bind menu**, select **Unbound > Quick Bind**.
2. Select from **This Sequence > Call Function** > select any listed binding.

Bindings are named after the actor they were created from and have "Dynamic Binding" added to the end of their name. For example, if you made a Dynamic Binding from a generic cube, the binding name would be "Cube Dynamic Binding".

You can use the browse icon (**Dynamic Possession > Endpoint > Browse**) to navigate to the associated Blueprint endpoint.

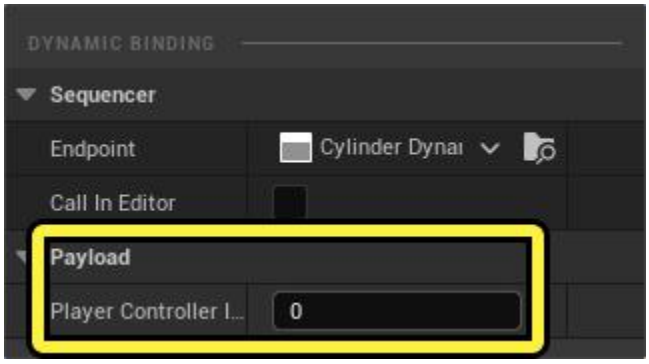


Deleting the actor in Sequencer does not delete any previously created Blueprints that Dynamic Binding created. If you want to remove any unused Blueprints that Dynamic Binding has previously created with other assets, you need to delete their functions from the associated sequence's **DirectorBP**. The associated functions are listed in the Functions section of the **My Blueprint** panel.



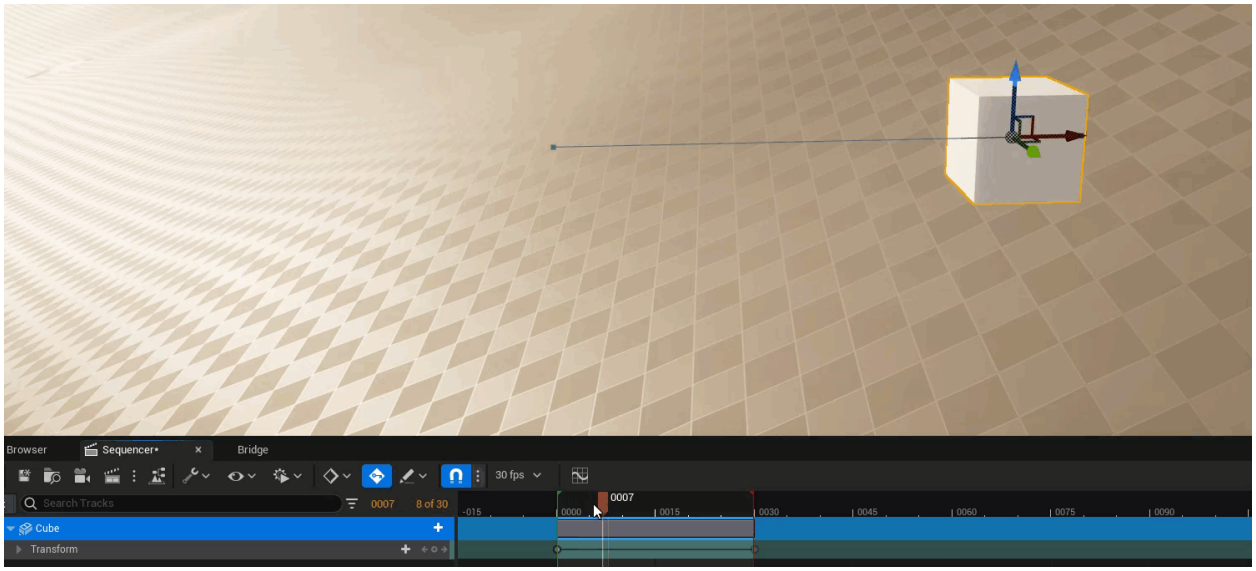
If an actor has been dynamically bound to a Player Pawn, you can select which Player Controller for the binding to use. Under **Dynamic Possession**, select the **Resolve to**

Player Pawn endpoint. When you look at the binding again, you can select which Player Pawn to use in the Player Controller Index under the **Payload** section.



Call In Editor

You can use the **Call In Editor** to debug your dynamic bindings. Once an actor has been dynamically bound, you can enable the **Call In Editor** option (right-click the actor in **Sequencer > Dynamic Possession > Call In Editor**). When this feature is enabled, the assigned endpoint is triggered in the Editor outside of Play In Editor (PIE).



> Any changes made as a result of this endpoint being called may end up being saved in the current level or asset.

Clear a Dynamic Binding

To remove a dynamic binding, follow these steps:

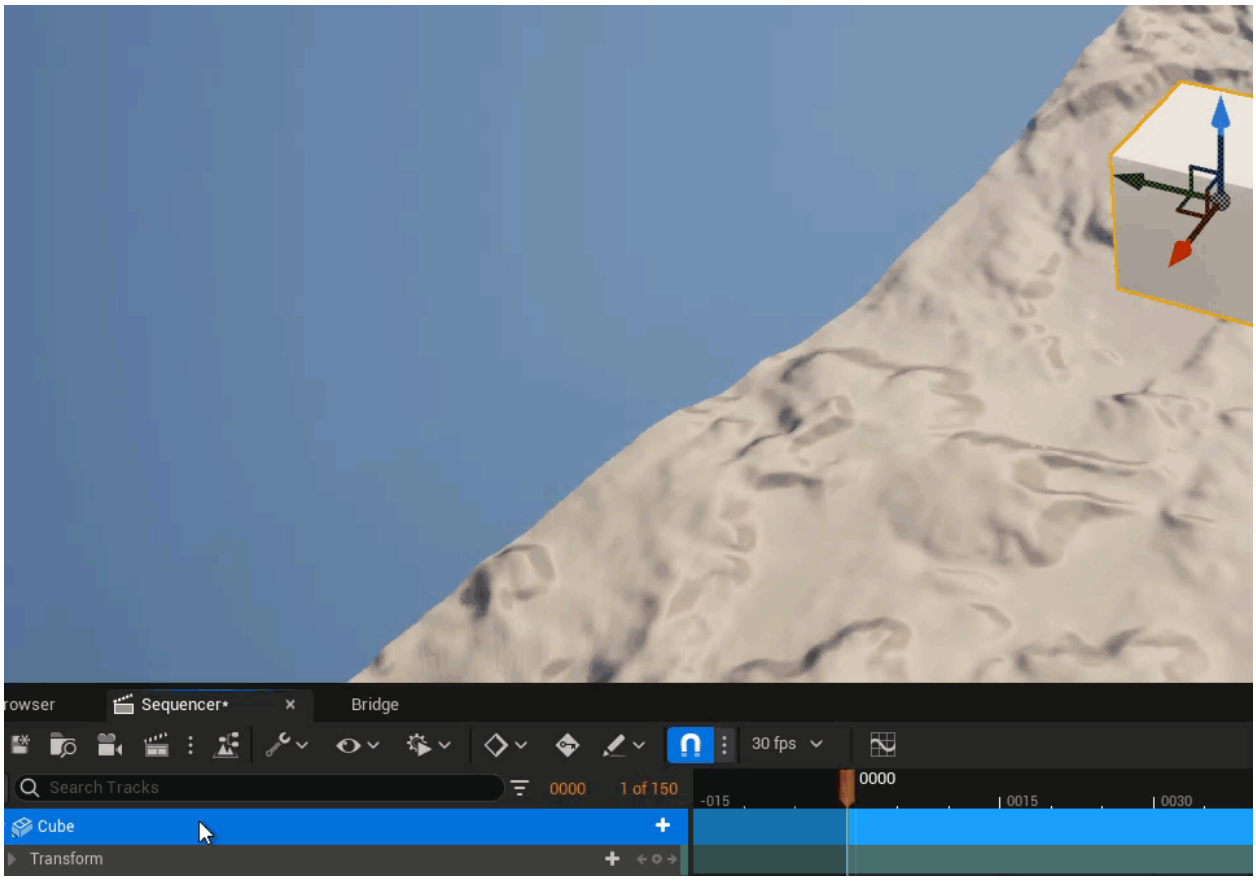
1. Right-click the dynamically bound actor.
2. Select **Dynamic Possession**.
3. From the dropdown in **Endpoint**, select **Clear**.

This removes the dynamic binding but does not remove any associated Blueprints.

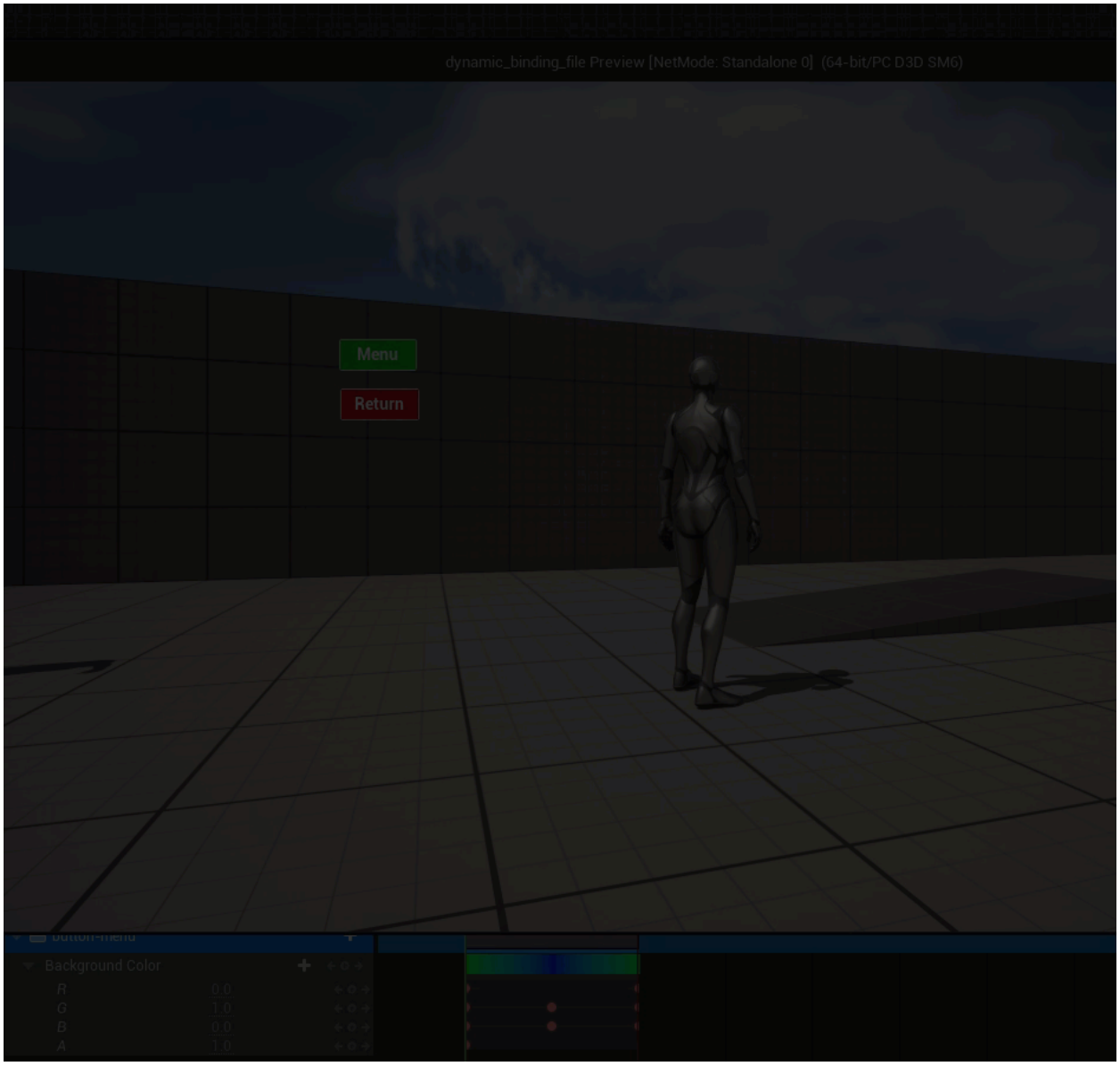
Rebinding Dynamic Bindings

To can rebind a dynamically bound actor to another endpoint, follow these steps:

1. Right-click an actor.
2. Select **Dynamical Possession**.
3. From the dropdown in **Endpoint**, select **Rebind To** and select which endpoint you want to use.



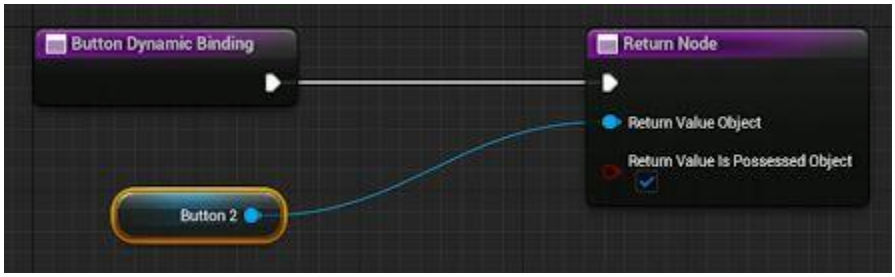
UMG and Dynamic Binding



Creating a dynamic binding for buttons. Note: The Blueprints were created ahead of time and you will need to modify your Blueprint nodes as needed.

Creating a dynamic endpoint for an **Unreal Motion Graphics UI Designer** (**UMG**) asset is similar to any other actor binding with one difference: creating a Dynamic Binding with an UMG asset doesn't open the **Level Sequence Blueprint Director**. This process creates a function inside the UMG asset, since there is a **UI Blueprint Graph** already associated with your asset. You should also note that, unlike non-UMG assets,

the process of creating a dynamic binding does not automatically open the associated Blueprint.



Dynamic Binding Exercise

This tutorial is an example of how you can dynamically bind an actor to another actor and use the **Call In Editor** to see dynamic binding happen in the **Viewport**.

1. Create a new empty level.
2. Create a **Cube** and a **Sphere** from the Shapes menu: **Quickly add to the project**: **Shapes > Cube** and **Shapes > Sphere**.
3. Create a **Level Sequence** and add the Cube to it:
    a. Right-click in an open space in **Sequencer**
    b. Select **Actor to Sequencer**
    c. Find the **Cube** is the available actors to add to your sequence.
4. Select the **Sphere** from the **Outliner** and add an **Actor Tag** to the **Sphere**.
    a. In the **Details panel** click the **filter by Actor** option.
    b. In the **Tags** section (under Advanced), click the **Add (+)** button to add an **Array element**.
    c. Enter the tag name of 'sphere'. Don't confuse this actor tag with a Sequence Tag. This process uses the actor Tag later that is picked by a Blueprint node.

5. Add a short animation to the cube (for example, animate the cube moving on Location X over 30 frames). This can be any movement so you can observe dynamic binding in action later.
6. Create a Dynamic Binding for the cube:

a. Right-click on the Cube in Sequencer and select **Dynamic Possession**.

b. Select **Endpoint > Create New Endpoint**. This opens the **Level Sequence Director Blueprint** with two Blueprint nodes: **Cube Dynamic Binding** and **Return Node**.

7. Modify the newly created Blueprint to get all Actors with a select tag (‘sphere') so that the Sphere can replace the Cube.

    a. Add **Get All Actors with Tag** node and **Get (a copy)** node.

    b. Connect the **Exec out pin** from the **Cube Dynamic Binding node** to the **Exec in pin** on the **Get All Actors with Tag node**.

    c. In the **Tag field** of the **Get All Actors with Tag node**, type in ‘**sphere**'.

    d. Connect the **Out Actors pin** of **Get All Actors with Tag node** to the **array pin** on the **Get (a copy) node**. This returns the first index (0) of any actor with the actorTag of ‘sphere'.

    e. Connect the **Exec pin** of **Get All Actors with Tag node** to the **Exec pin** of **Return Node node**.

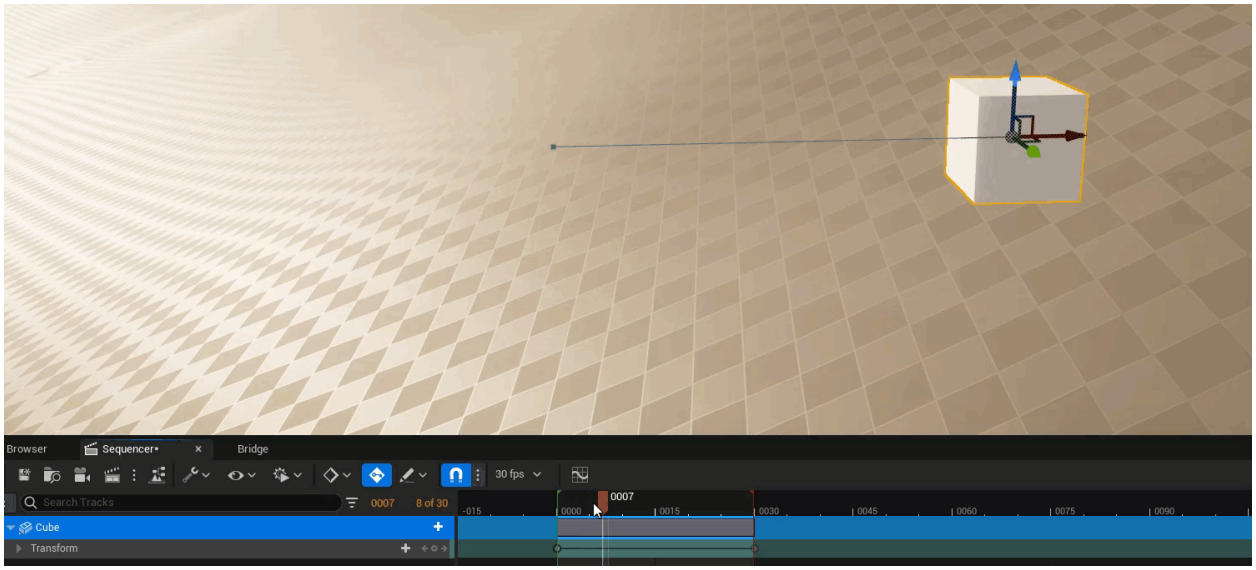
    f. In the **Return Node**, you need to **Split the Struct**. Right-click the **Return Value pin** and select **Split Struct Pin**. This exposes the **Return Value Is Possessed Object** which provides you with the possession of the sphere.

    g. Finally, connect the **Output pin** of **Get (a copy) node** to the **Return Value Object pin** of the **Return Node node**.

8. **Compile and Save** your customized Blueprint.

9. Back in **Sequencer**, right-click the Cube, go to **Dynamic Possession**, and enable **Call In Editor**.

10. Scrub the playback head in Sequencer and observe that the cube has been replaced with the sphere and takes its animation as well.


Browser
Sequencer*
Bridge
30 fps
Search Tracks
0007
8 of 30
-015
0000
0007
0015
0030
0045
0060
0075
0090
Cube
Transform