LECTURE 8

Blueprint Communication

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Present Blueprint Communication
- Show how to use Direct Blueprint Communication
- Explain casting in Blueprints
- Show how to reference Actors in the Level Blueprint
- Present event dispatchers
- Explain Blueprint Interfaces

Outcomes

By the end of this lecture you will be able to

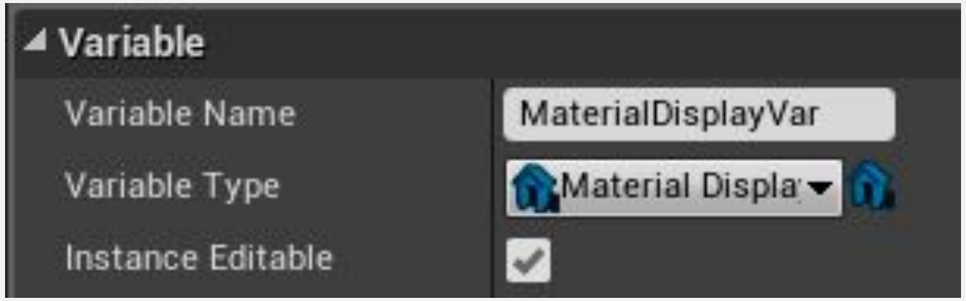
- Set up interactions between Blueprints
- Use casting to get the expected Blueprint reference
- Create Actor references in the Level Blueprint
- Create new Blueprint Interfaces

DIRECT BLUEPRINT COMMUNICATION

Direct Blueprint Communication is a simple method of communication between Blueprints. It can be used by creating a variable that will store a reference to another Blueprint.

The top image on the right shows a variable named “**MaterialDisplayVar**”. The type of this variable is “**Material Display**”, which is a custom Blueprint class.

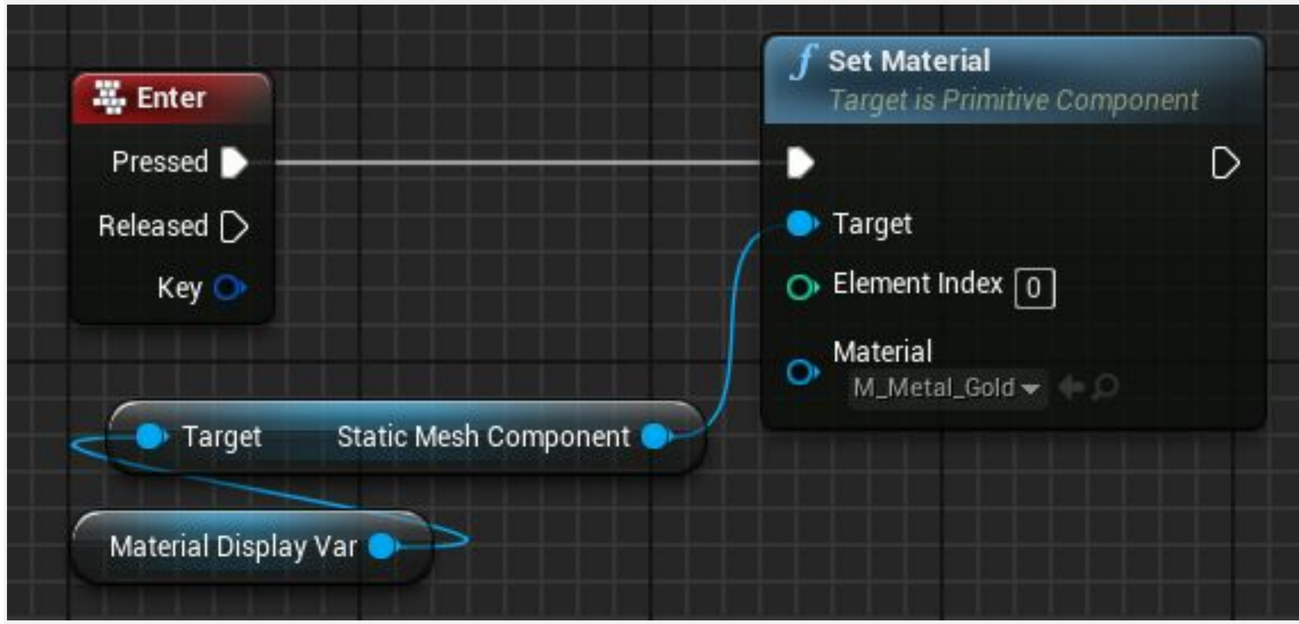
The **Instance Editable** property is checked, so this variable can be set in the Level Editor. The bottom image shows a drop-down menu where an instance of the **Material Display** class in the Level can be chosen.



DIRECT BLUEPRINT COMMUNICATION: TARGET

The image on the right shows the **Set Material** function. The target being used in the function is a Static Mesh component of the **Material Display Var** variable.

So the Blueprint that will be modified by the function is the instance of **Material Display** that was chosen in the Level Editor.



CASTING IN BLUEPRINTS

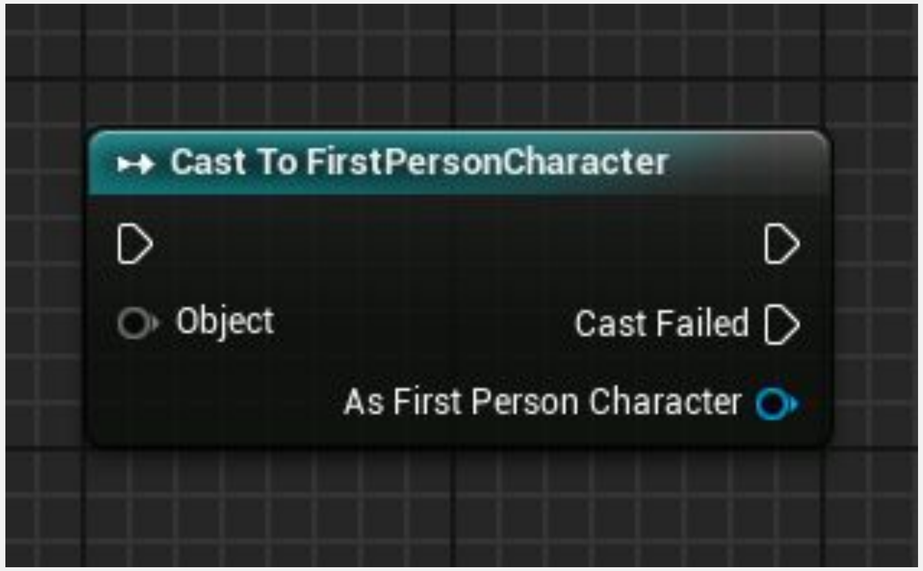
The **Cast To** node converts the reference variable type to a new specified type. This action is necessary in some situations in order to access the variables and functions of a class or Blueprint. The image on the right shows a **Cast To FirstPersonCharacter** Blueprint.

Input

- **Object**: Takes in a reference to an object.

Output

- **Cast Failed**: Execution pin that is used if the referenced object is not of the type used in the cast.
- **As [new type]**: Outputs a reference using the new type specified in the cast.

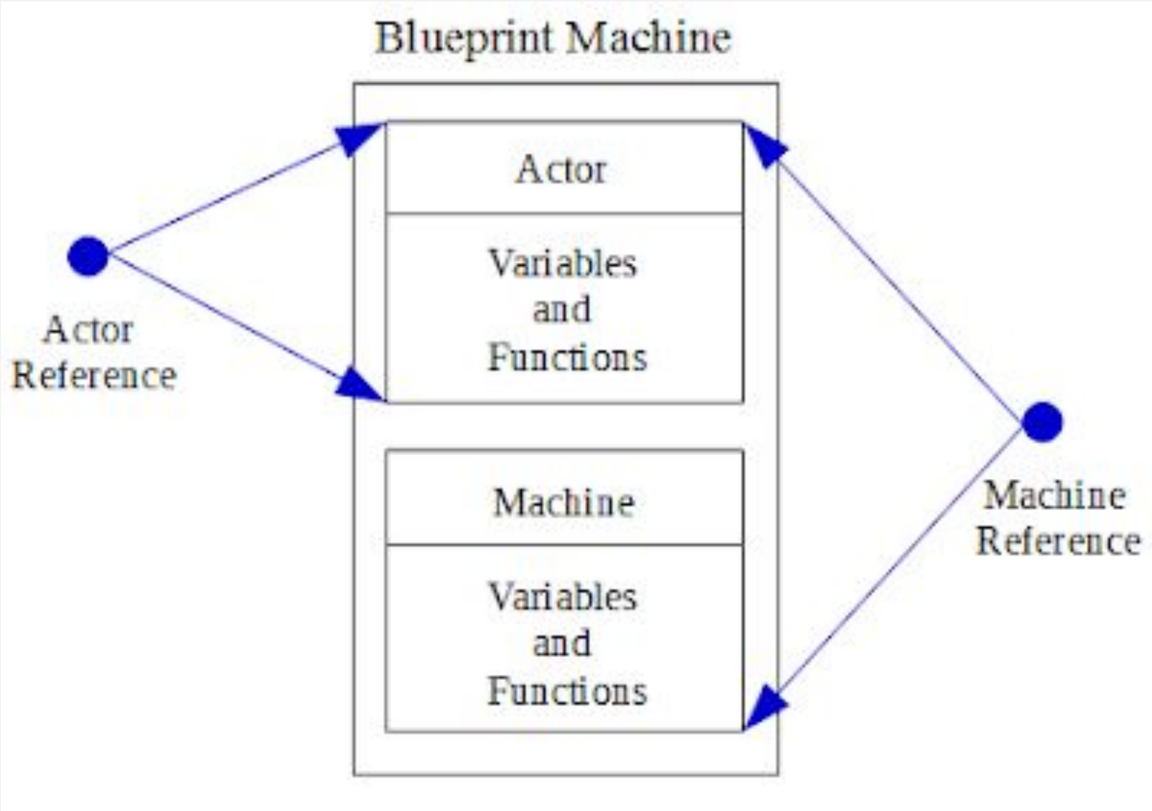


CASTING IN BLUEPRINTS: EXAMPLE 1/2

In this example, a new Blueprint class has been created called “**Machine**”. It is of the “**Actor**” type. The Blueprint has a function called “**Recharge Battery**”.

A reference to the **Actor** class can be used to manipulate the **Machine** Blueprint class, but it would not have access to the **Recharge Battery** function because it can only access the variables and functions that were created in the **Actor** type.

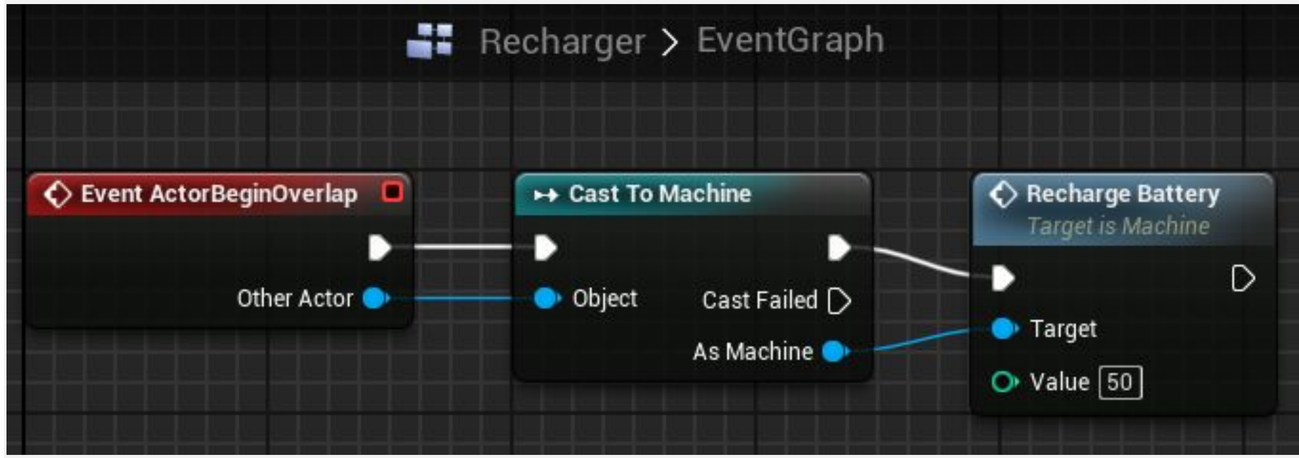
To access the **Recharge Battery** function, a reference to the **Machine** class is needed.



CASTING IN BLUEPRINTS: EXAMPLE 2/2

In another Blueprint class called “**Recharger**”, there is an Overlap event.

If the Actor who overlaps is of the **Machine** class, then the **Recharge Battery** function is called using the reference of the **Machine** class provided in the **Cast To** node.

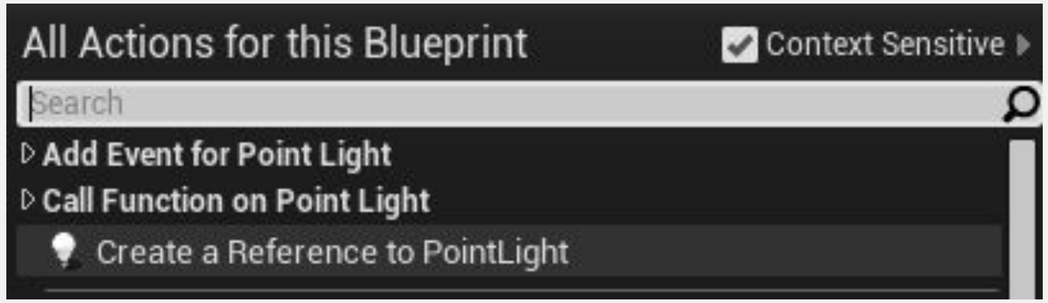


LEVEL BLUEPRINT COMMUNICATION 1/2

It is easy to add references to Level Actors to a Level Blueprint, enabling the Level Blueprint to directly communicate with them.

To add a reference, follow these steps:

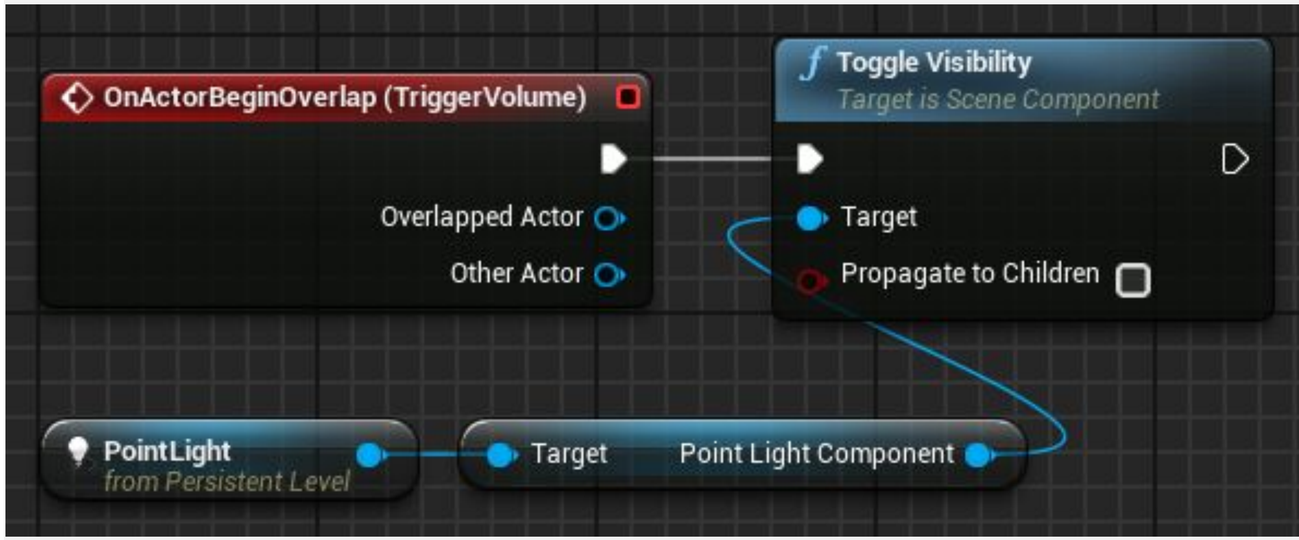
- First select the **Actor** in the **Level Editor**.
- Open the **Level Blueprint**.
- Right-click in the **Event Graph** and choose the option “**Create a Reference to [Actor name]**”.



LEVEL BLUEPRINT COMMUNICATION 2/2

In the image on the right, an Overlap event of a Trigger Volume has been placed in a Level Blueprint.

When an overlap occurs, the **Toggle Visibility** function is called on the Point Light component of a Point Light Actor.



EVENT DISPATCHER

An **event dispatcher** allows a type of communication between Blueprint classes and the Level Blueprint. It is created in a Blueprint class and can be implemented in the Level Blueprint.

Event dispatchers are created in the My Blueprint panel. The image on the right shows an event dispatcher that was created with the name “**ButtonPressed**”.

EVENT DISPATCHER: EXAMPLE 1/3

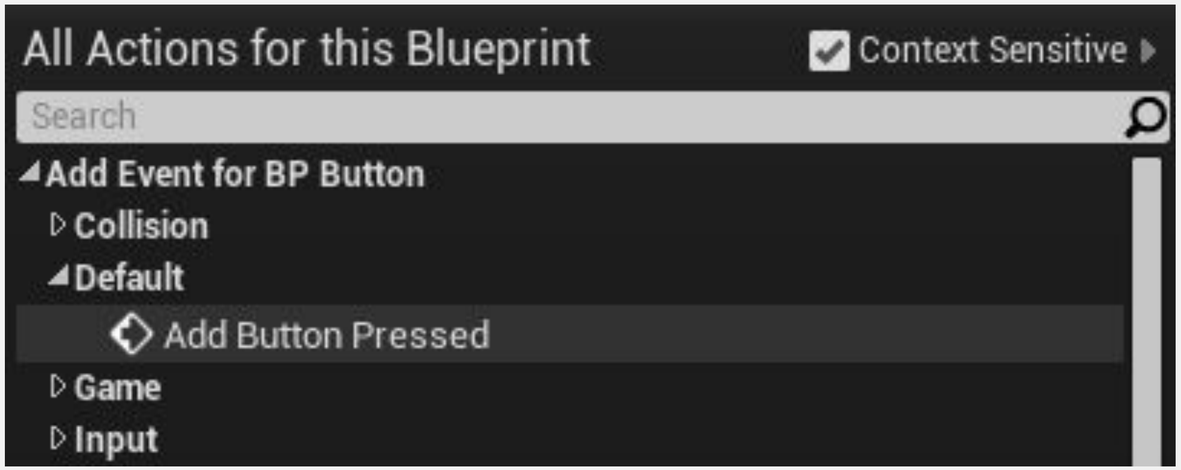
In this example, a Blueprint class named “**BP Button**” represents a button that can be pressed. The only purpose of this Blueprint is to communicate when the button is pressed. As a result, the button can be used in several different situations. The actions that will occur when the button is pressed will be set in the Level Blueprint.

The **BP_Button** Blueprint contains an **Enter** keyboard event. When the **Enter** key is pressed, the **ButtonPressed** event dispatcher will be called, as seen in the figure on the right.

EVENT DISPATCHER: EXAMPLE 2/3

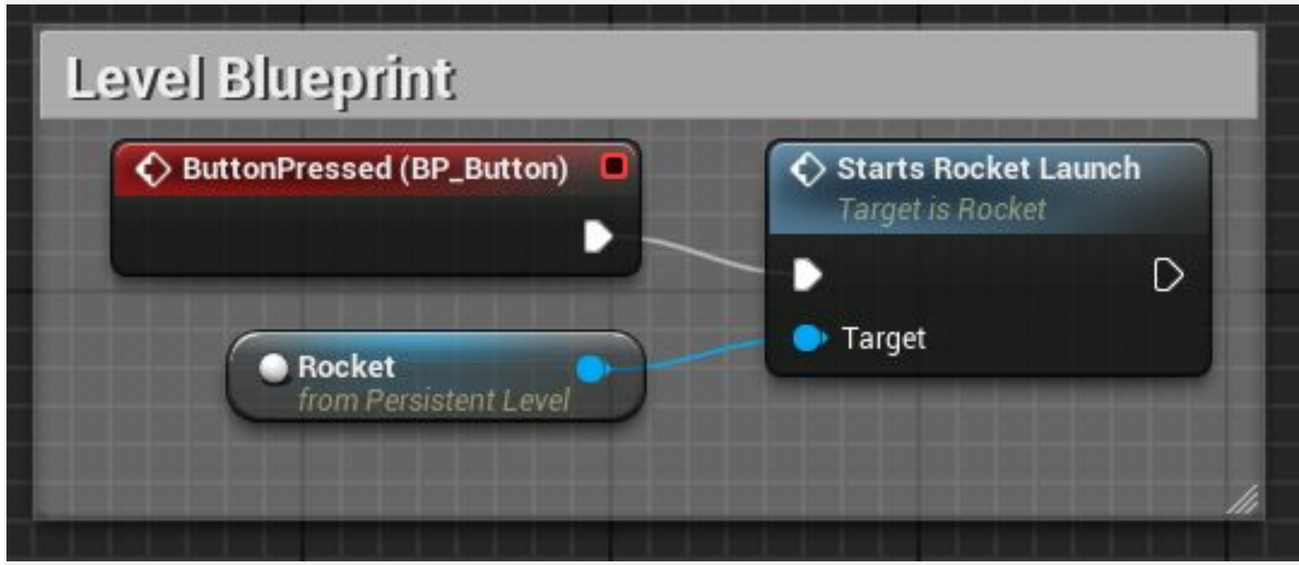
Add a **BP_Button** Actor to the Level and select it.

Open the **Level Blueprint** and right-click in the **Event Graph** to add the **ButtonPressed** event related to the event dispatcher that was created.



EVENT DISPATCHER: EXAMPLE 3/3

Assume there is a Blueprint called “**Rocket**” in the Level and that it contains a function called “**Starts Rocket Launch**”. The Level Blueprint is responsible for calling the **Starts Rocket Launch** function when the specified button is pressed.

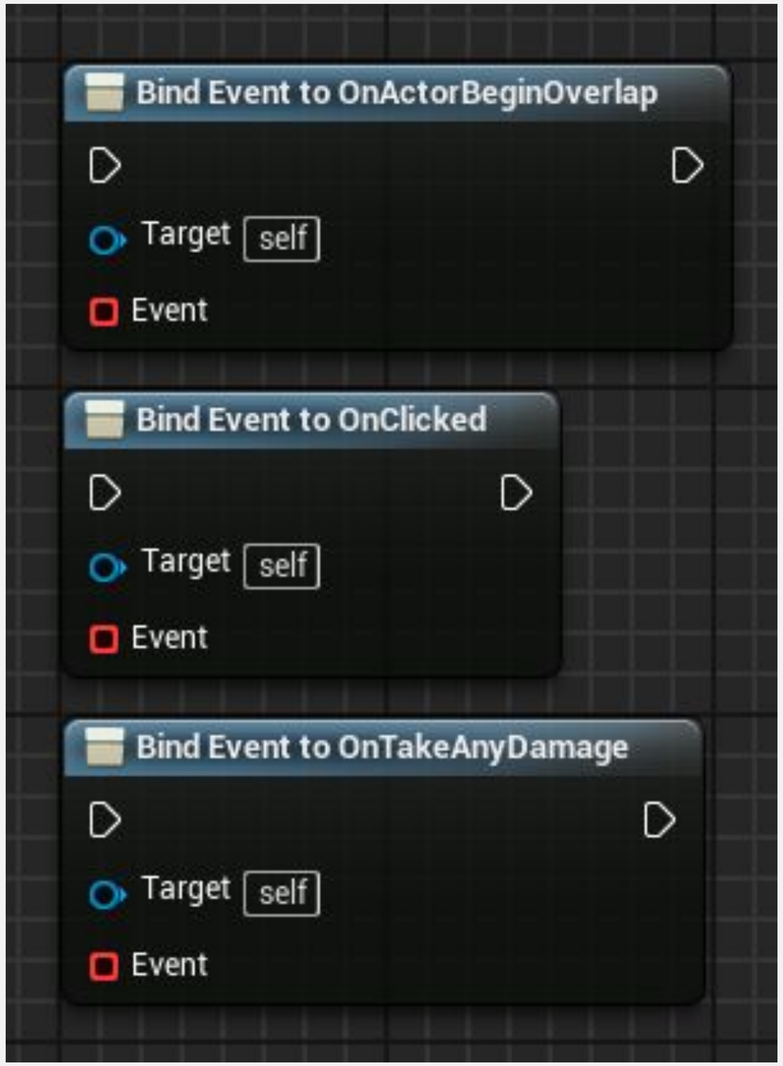


BIND EVENT NODE

The **Bind Event** node binds one event to another event or to an event dispatcher, which can be in another Blueprint. When an event is called, all the other events that are bound to it are also called. The binding is done using the event delegate (the red pin on an event node).

Input

- **Target**: Object that has the event that receives the binding.
- **Event**: Reference to an event that will be bound to another event, to be called later.

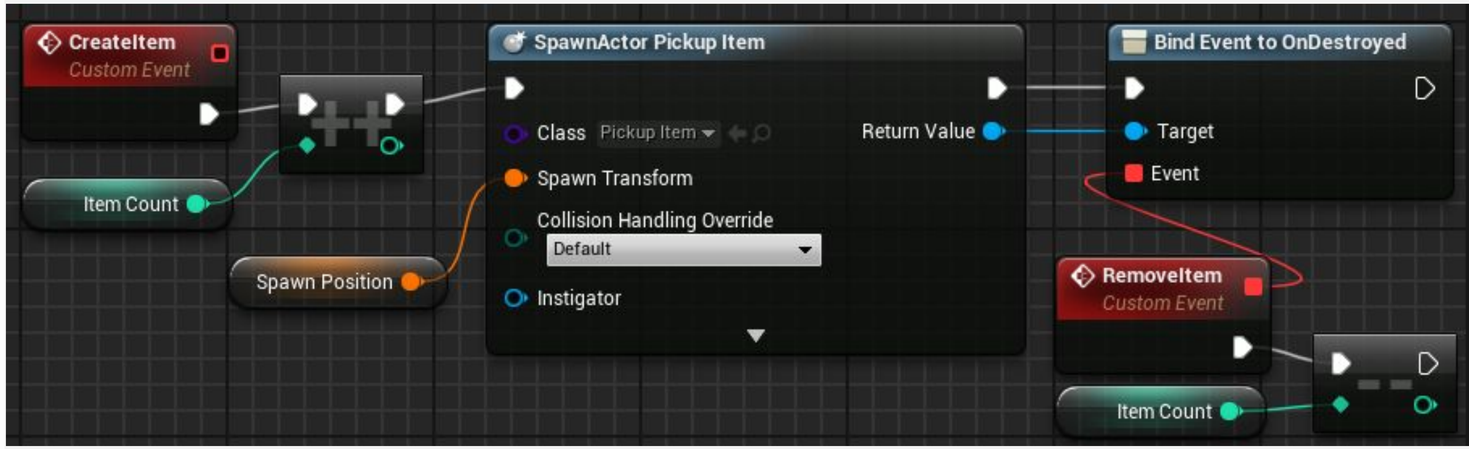


BIND EVENT NODE: EXAMPLE

In this example, there is a Blueprint class called “**PickupManager**” that is responsible for spawning **Pickup Item** Actors and contains a variable called “**Item Count**” that keeps track of the number of **Pickup Item** Actors that exist in the Level.

A custom event called “**RemoveItem**” decreases by “1” the value of the **Item Count** variable whenever a spawned **Pickup Item** Actor no longer exists in the Level.

The **RemoveItem** event is bound to the **OnDestroyed** event of the **Pickup Item** Actor that was spawned. Thus when the **Pickup Item** Actor is destroyed in the game, the **RemoveItem** event will be called.

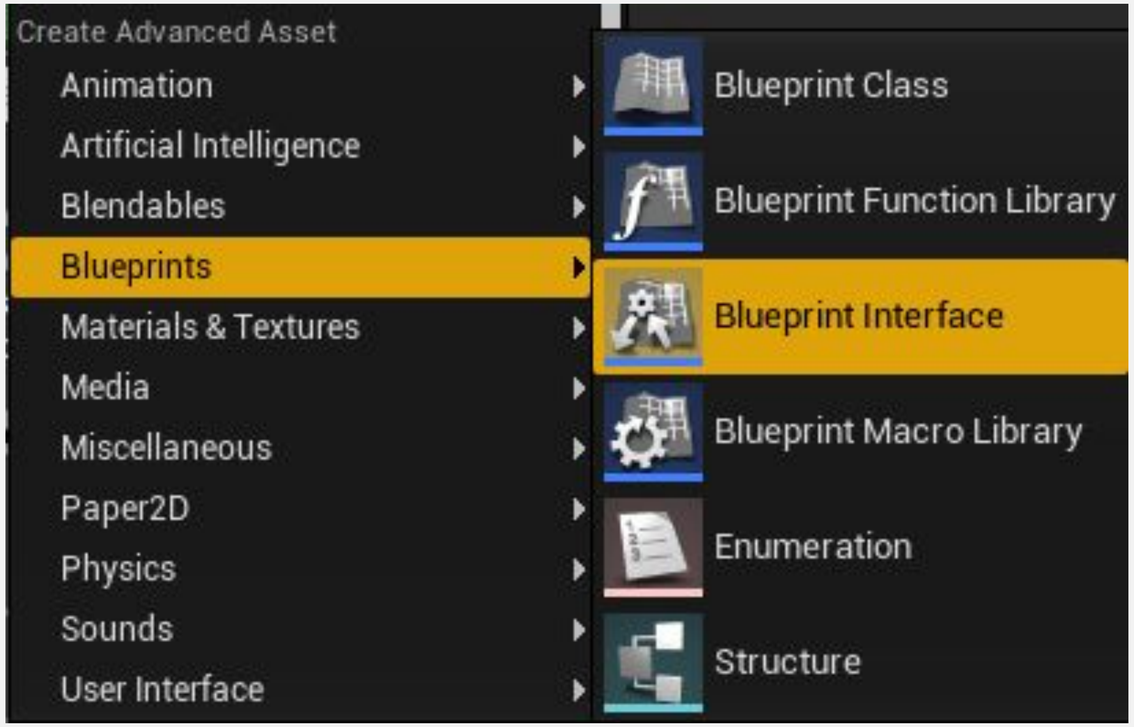


BLUEPRINT INTERFACE

A **Blueprint Interface** (**BPI**) contains only the definitions of functions, no implementation.

If a Blueprint class implements a BPI, it uses the definition provided and then implements its own logic for that function.

To create a new Blueprint Interface, click the green **Add New** button in the **Content Browser**, and in the **Blueprints** submenu select “**Blueprint Interface**”.

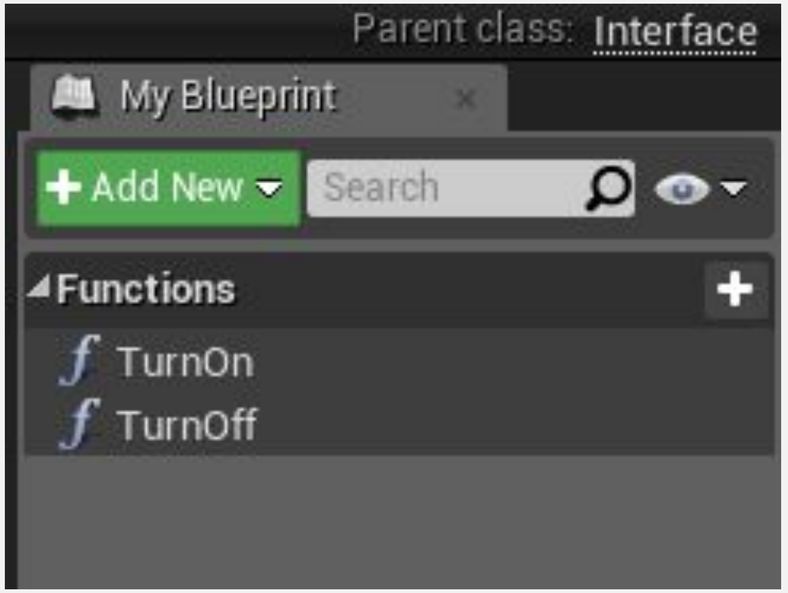


BLUEPRINT INTERFACE: CREATING THE FUNCTIONS

In this example, a Blueprint Interface named “**BP_Interface_TurnOnOff**” has been created.

When editing a BPI, it is possible to name the functions and create input and output parameters, but it is not possible to implement logic in the interface.

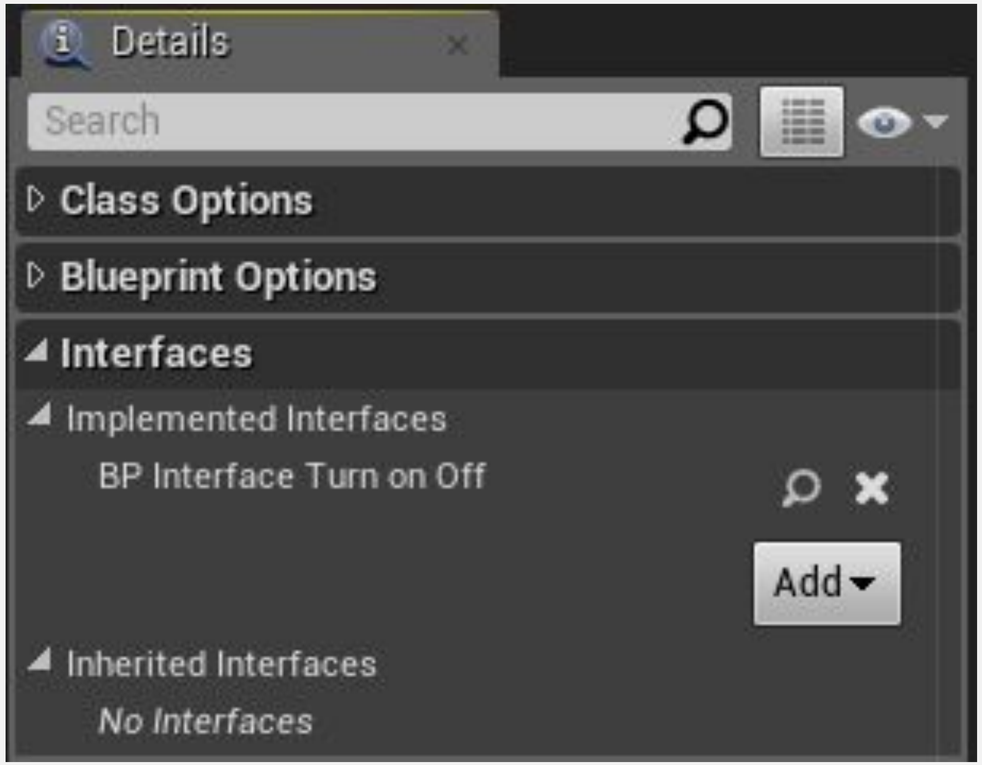
This BPI has two functions: **TurnOn** and **TurnOff**, as seen in the image on the right.



BLUEPRINT INTERFACE: ADDING TO A BLUEPRINT

To add a Blueprint Interface to a Blueprint class, click the **Class Settings** button on the **Toolbar** in the **Blueprint Editor**. In the **Interfaces** section in the **Detail** panel, click the **Add** button and choose a Blueprint Interface class.

In the image on the right, the **BP Interface Turn On Off** Blueprint Interface was added.

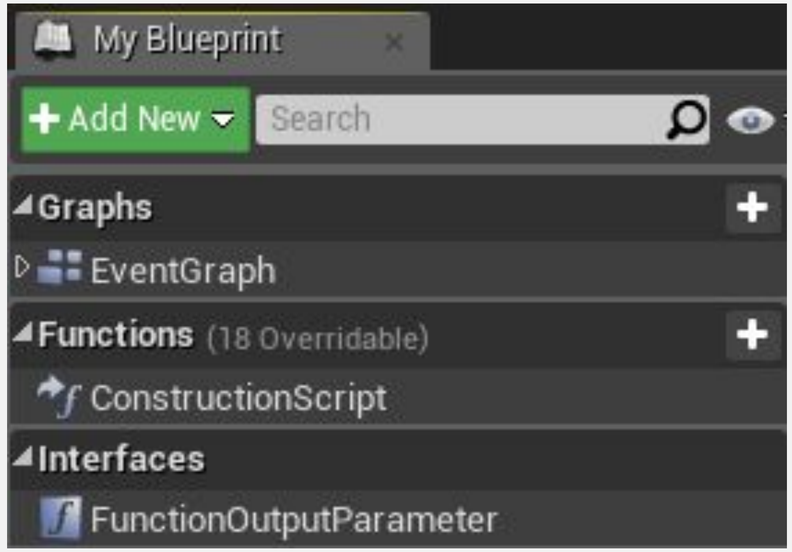


BLUEPRINT INTERFACE: IMPLEMENTING THE FUNCTIONS

Functions of a Blueprint Interface that have no output parameters appear as events in the Blueprint that implements the BPI.

Functions of a Blueprint Interface that have an output parameter appear in the **My Blueprint** panel in the **Interfaces** section.

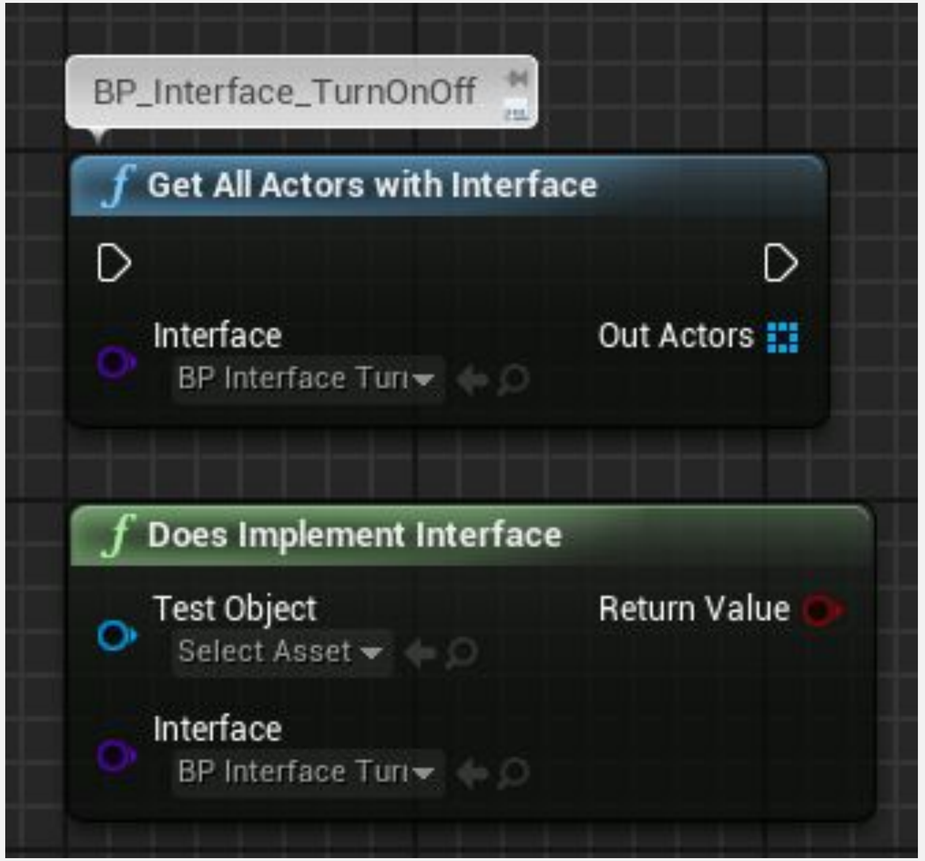
The bottom image on the right shows a function named “**FunctionOutputParameter**”. The function is implemented by double-clicking it to open the function for editing.



BLUEPRINT INTERFACE: UTILITY FUNCTIONS

There are some utility functions related to Blueprint Interfaces. Listed below are two examples :

- **Get All Actors with Interface**: Finds all Actors in the current Level that implement the BPI specified.
- **Does Implement Interface**: Tests if one specific object implements the BPI.



SUMMARY

This lecture presented Blueprint Communication and showed how to use Direct Blueprint Communication, Blueprint Interfaces, and event dispatchers.

It also explained how to cast, bind events, and reference Actors in the Level Blueprint.