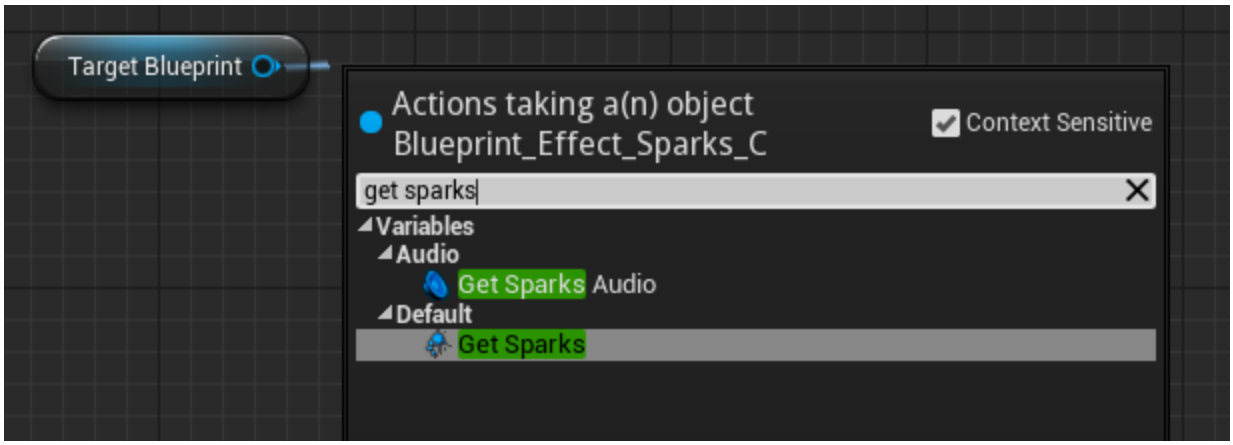
Blueprint Communication Usage

Overview of when to use different methods of Blueprint Communications.

When working with Blueprints, in order to pass or share of information between your Blueprints, you will need to use a form of **Blueprint Communication**. Based on your needs you can use several different types of communication and this page outlines the most common methods and provides links and sample usage cases.

Direct Blueprint Communication

Direct Actor communication is the most common method of sharing information between Actors in your Level. This method requires a reference to the target Actor so you can access its information from your working Actor. This communication type uses a one-to-one relationship between your working Actor and your target Actor.



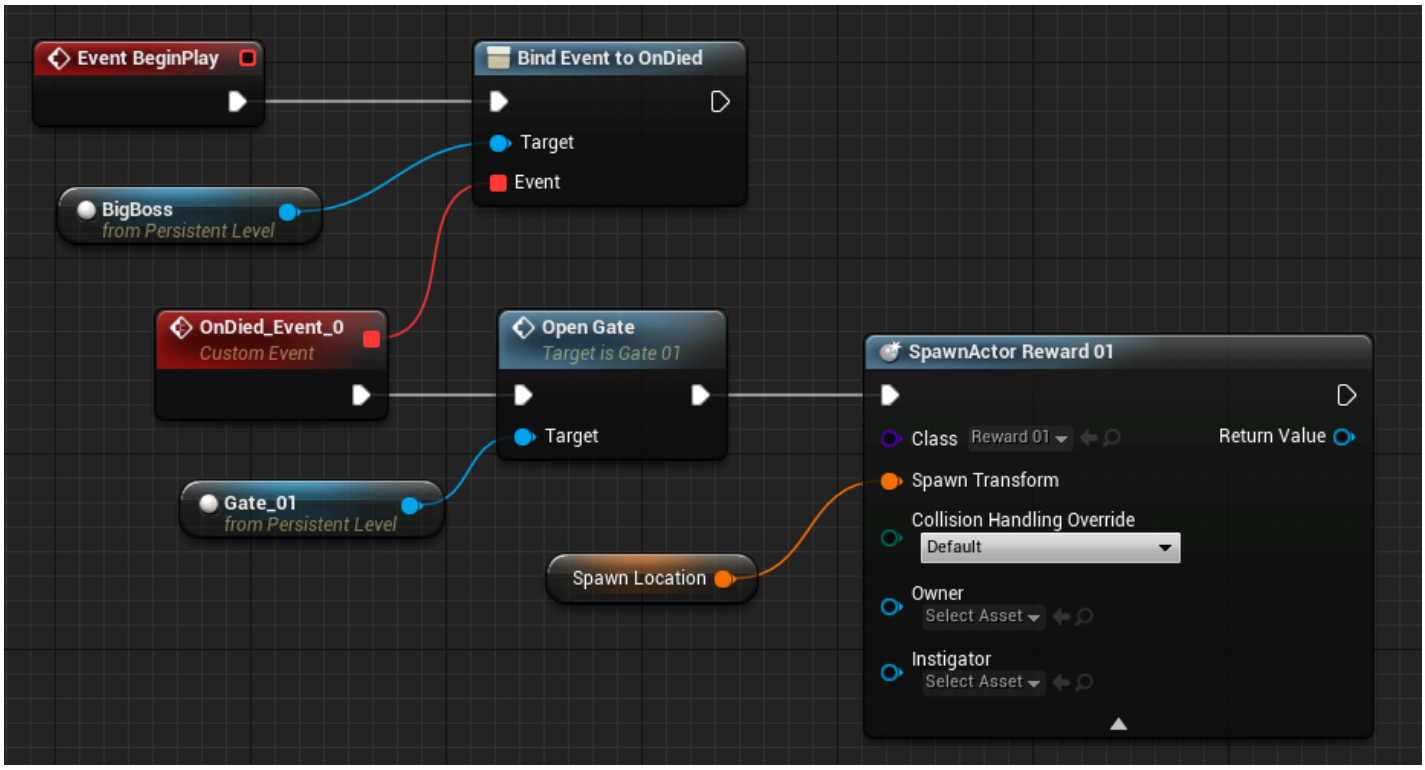
When to Use It

Below is an example of when you might use **Direct Blueprint Communication**:

- You have two Actors in your level that you want to communicate with each other.
- Interacting with a switch in your level opens a specific door or turns on a specific light (each of which are separate Blueprints).

> Refer to Setting Up Direct Blueprint Communications for a tutorial on how to setup Blueprint Communication between Actors.

Event Dispatchers



Event Dispatchers are best suited for telling other "listening" Blueprints that an event has happened. When that event occurs, those listening Blueprints can then react and do whatever you want them to do independently of each other.

For example, suppose you had a Boss in your game that when killed calls an "OnDied" Event Dispatcher. You could Bind the "OnDied" event inside of any number of other Blueprints such as your Character (who performs a celebration), a door in your level (which opens) or a HUD (which flashes some UI message) all of which are executed when getting the "OnDied" notification that the Boss was killed.

> Please see Event Dispatchers for more information.

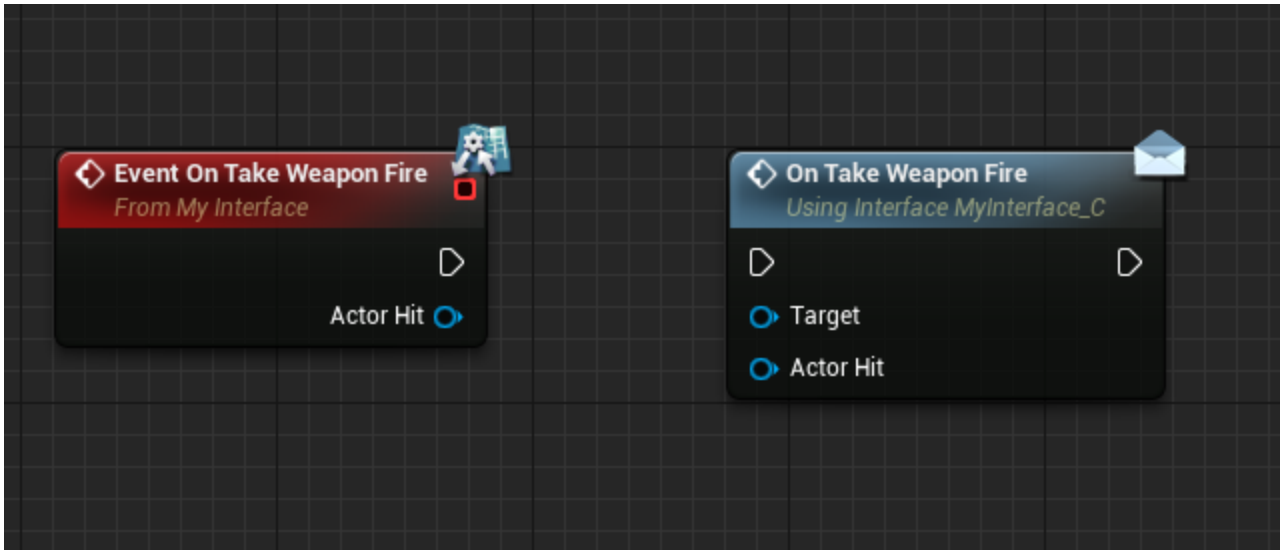
When to Use It...

Some examples where an **Event Dispatcher** could be used:

- **You want to communicate from your character Blueprint to your Level Blueprint.**
  - Your player character levels up and you want to open a previously locked area.
  - Your player character presses an action button that does something in your level.
- **You want to have Events fired when a spawned Actor has something done to it.**

- You spawn a boss and an Event is fired when the boss is killed which spawns a reward in the world.
- You spawn an item in your level (weapon, health, etc.) and notify the item and character when it is picked up.

Blueprint Interfaces



Blueprint Interfaces (or **Interface** for short) allow for a common method of interacting with multiple types of objects that all share some specific functionality. For example, you can have completely different types of objects like a car and a tree both of which share one specific thing; they can both be shot by weapon fire and take damage.

By creating a Blueprint Interface that contains an **OnTakeWeaponFire** function and having both the car and the tree implement that Blueprint Interface, you can treat the car and the tree as the same and simply call the **OnTakeWeaponFire** function when either of them is shot and have them respond differently.

The best way to think of Blueprint Interfaces is that they are sort of like a contract. This contract says, "if you implement this interface, you promise to implement these specific functions, and respond to them when I call." Multiple Blueprints can implement an Interface, but only those that care will respond (those that don't will simply ignore the call). Going back to the weapon fire example, you may not want your walls to react to damage, so they just wouldn't implement the interface with **OnTakeWeaponFire** and ignore the call.

Please see Implement Interface in Blueprint for more information.

When to Use It...

Each of the examples below are scenarios where a **Blueprint Interface** might be a good method of communication:

- **You have some functionality that is similar in several Blueprints but execute differently when called.**
  - Player has a flamethrower that when used, fires an Event called **ElementalDamage**. A Tree Blueprint gets the ElementalDamage call and burns the tree down while a Snowman Blueprint melts the snowman.
  - Player has a "use" button, pressing the use button opens doors, turns off lights, picks up items, etc.
  - Enemy has a special ability that changes and performs differently based on player health.

Blueprint Casting

Another common form of Blueprint Communication is through the use of **Cast** nodes. When using a Cast node, put simply you are asking the object "are you a special version of that object" and if so let me access you, if not then disregard my request.

For example, suppose you created a special Character Blueprint called "Flying Character Blueprint" that allows a player character to fly in your game. You could use a **Get Player**

Character node to access the player character's movement component and affect the player in general ways such as setting its location, rotation, etc. You would not have access to the ability to fly as the Character Blueprint doesn't know about flying. Only the "Flying Character Blueprint" knows about flying. In this instance, you would get the player character and Cast To Flying Blueprint which would then grant you access to the flying function.

If the player character is not using the Flying Character Blueprint but instead is using a different type of Character Blueprint entirely, when we perform a Cast To Flying Blueprint it would fail since we are not using the Flying Blueprint and therefore would not be able to access and execute the flying command.

When to Use It...

Here are a few examples when you could use **Blueprint Casting**:

- **You want to access a specialized version of another Blueprint.**
  - Character walks into fire causing a Health Value to deplete.
    - Cast to your special Character Blueprint to access and change the Health Value.
  - Your character dies and you need to respawn them.
    - Cast to your special Game Mode Blueprint to execute a respawn script.
- **You want to access multiple Blueprints of the same class and modify them all the same way.**
  - You have several lights in your level and you want them all to turn on or off when an event occurs.
    - Cast to your Light Blueprint and execute a function that turns the light off.
- **You want to access a specific Child Blueprint.**
  - You have several Blueprints based off an Animal Blueprint (Cat, Dog, Bird) and you want to access one of the animals.
    - Cast to Cat, Cast to Dog or Cast to Bird to access their respective Blueprints and unique functionality.