Table of Contents

Note about this primer

Casting is a fairly basic functionality in Unreal blueprinting and we will go through the basics of it in this guide, but not delve into the finer details of do's and don'ts as these are very varied and simply a matter of knowing the system and the hierarchy of blueprints. Having said that, we will go through some examples of casting used in Conan Exiles and explain what they do and how. Hopefully these examples will clarify (at least to some extent) how casting works.

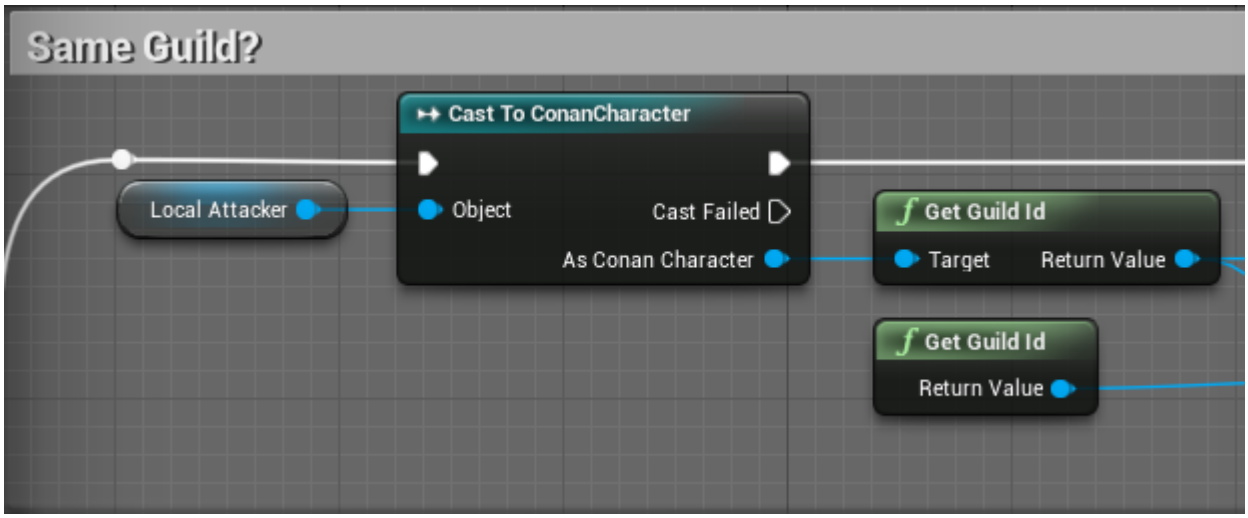
What is "Casting"?

In short - Casting is a method of checking target objects if they are the right type of object, and then (probably at least) use this new object as a basis of new function calls. This is used for when you need to access functions that do not exist in the current blueprint but rather on a target blueprint (for example a player stepping on a trap, or suchlike).

The Unreal Engine team has an excellent guide on the core functionality of casting here:

https://docs.unrealengine.com/en-us/Engine/Blueprints/UserGuide/CastNodes

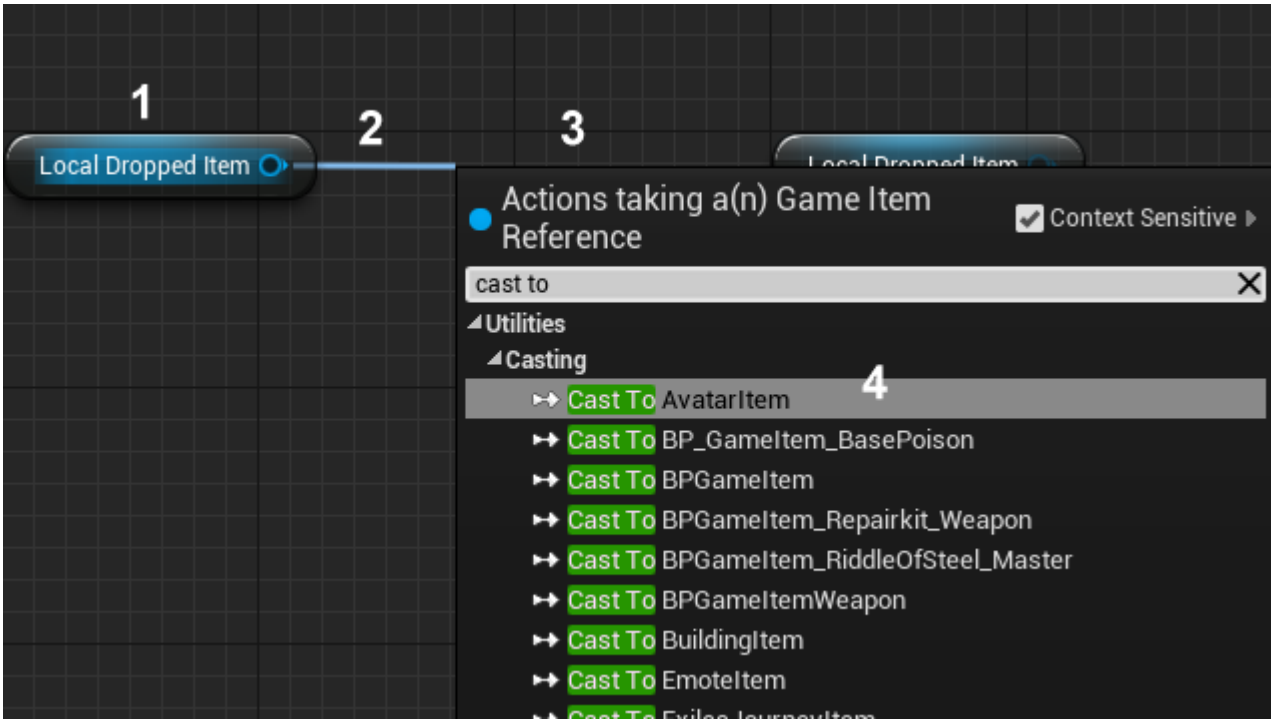
But for this guide, we will go through some practical examples and usages as well as some things one should stay away from.



An example of a cast used to grab the Guild ID from a player attacking a target

How to cast

You can start a cast in two ways - either you have an object reference that you can use as a source, or you can start a cast and then hook up the Object of the cast after the fact. However - the easiest method is the first one, since invalid target objects simply won't be part of the list of objects to cast to, thus lowering the risk of a miscast.



When to cast and when not to

Casting is not inherently a bad thing to do but it does require the system to load the target blueprint if it isn't already, which can detract from performance. Thus, operations that are run very often or are expensive in other ways (For-loops, recursive checks, etc) should not be cast if it can at all be avoided.

Do cast when

- You need to access functions that are not available in any other way, typically because the target blueprint is not a parent of the current blueprint.

Don't cast when

- Your blueprint is inheriting data from the target you are attempting to cast to.
- You have already cast to the same target prior to the current cast in the execution line.
- You have an alternative to casting that is less expensive, such as an interface call

Cast Failed >

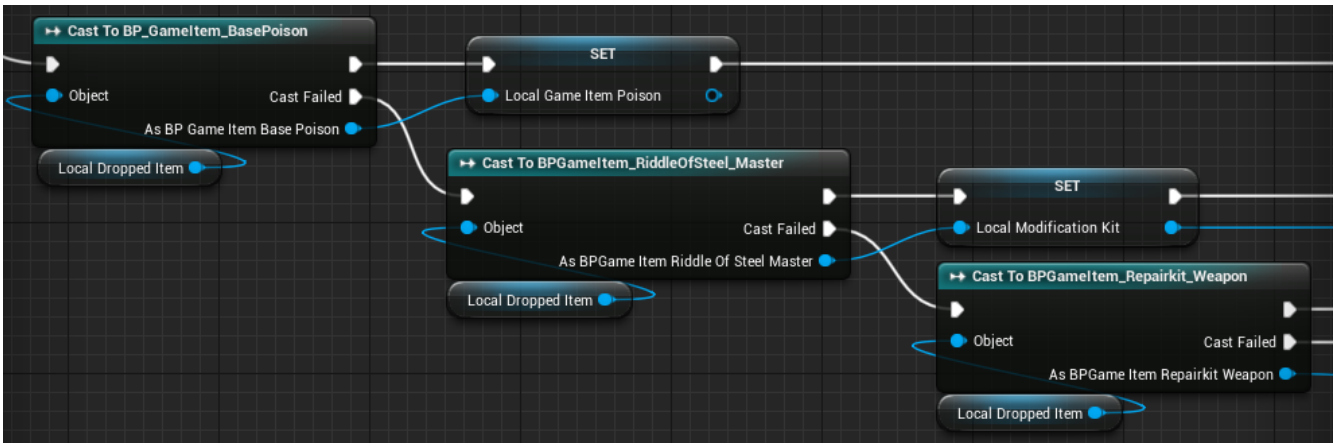
You will notice that when you drag in a Cast node, it has two execution paths - one for if the cast was successful and one for when the cast was not successful.

You might also notice that some of the blueprints in Conan Exiles do not have an execution path for the Cast Failed. This is because these casts have been deemed fool-proof or in no need of a backup for the failed cast.

If you are certain the cast will succeed, you typically won't need a Cast Failed execution path - although you should always double-check this.

Example - Using cast to check for valid blueprints

The example we will use here is from the blueprint "**BPGameItemWeapon"**, which has a function called "**On Item Dropped on Item**". This function checks if the item dragged on to another item in the UI is a specific type of blueprint (and thus has special functions).



When the function is initialized, we get the "Local Dropped Item" variable. We then use this variable as a target for Casts to check if it belongs to the BasePoison class, RiddleOfSteelMaster or RepairKit type blueprints. Each of these casts, if resulting in true, allows us to then grab properties of the target blueprint for use later in the execution path.

As mentioned, once we have the Local Dropped Item, we use that to first make a cast to **BP_GameItem_BasePoison**. If that fails, we cast to **BPGameItem_RiddleOfSteel_Master**, and if that fails, we finally cast to **BP_GameItem_RepairKit_Weapon**. If that fails, we return a "Can't use this" message to the player.

So - this casts three times and all the targets are items. So - why not do it this way instead:

1. Implement an ItemType variable in the BP_GameItem blueprint
2. Set up that variable differently for the different gameitems (as you can see, they are all items that ultimately share the same parent; BP_GameItem)
3. Implement a *single* cast to BP_GameItem blueprint and then check against that variable to find what type of item is being used

The answer is that all the different children (BasePoison, RiddleOfSteel, Repairkit) have custom variables and functions in them, and if we cast to the parent, the cast would never know about these variables or be able to access them in subsequent functions related to these. If all those functions did exist on the parent, however, that is how we would do it.