Ability System Component And Attributes

Using the Ability System Component with Gameplay Attributes and Attribute Sets

The **Ability System Component** (`UAbilitySystemComponent`) is the bridge between Actors and the **Gameplay Ability System**. Any Actor that intends to interact with the Gameplay Ability System needs its own Ability System Component, or access to an Ability System Component on a PlayerState or Pawn.

Make sure that your project is set up to use the [Gameplay Ability System Plugin][making-interactive-experiences\GameplayAbilitySystem] before attempting to use the Ability System Component.

Basic Requirements

To set up your `AActor` subclass to use the Gameplay Ability System, implement the `IAbilitySystemInterface` interface and override the `GetAbilitySystemComponent` function. This function must return the Ability System Component associated with your Actor. In most cases, the Actor class will have a variable, tagged with `UPROPERTY`, that stores a pointer to the Ability System Component, similar to any built-in Component on any Actor type.

While it is common for an Actor to have its own Ability System Component, there are cases in which you might want an Actor, such as a player's Pawn or Character, to use an Ability System Component owned by another Actor, like a Player State or, in rarer circumstances, Player Controller. Reasons for this may include things like a player's score, or long-lasting ability cooldown timers that do not reset when the player's Pawn or Character is destroyed and respawned, or when the player possesses a new Pawn or Character. The Gameplay Ability System supports this behavior; to implement it, write the Actor's `GetAbilitySystemComponent` function so that it returns the Ability System Component you want to use.

Setup Example

The following procedure will help you to get started with a simple but common pattern that utilizes the Ability System Component.

1. Declare your class as a child of `AActor` or a subclass (`APawn` and `ACharacter` are common base classes), and add the `IAbilitySystemInterface` to its definition in the header file, like this:

```
class AMyActor : public AActor, public IAbilitySystemInterface
```

Copy full snippet

2. The `IAbilitySystemInterface` has a single function which you must override, `GetAbilitySystemComponent`, so declare that function in your class definition.

```
1 //~ Begin IAbilitySystemInterface
2 /** Returns our Ability System Component. */
3 virtual UAbilitySystemComponent* GetAbilitySystemComponent() const
  override;
4 //~ End IAbilitySystemInterface
```

Copy full snippet

3. In some cases, especially those where an Actor can be destroyed and respawned, you may want to keep the Ability System Component elsewhere, such as the Player State. For simplicity, this example will store it on the Actor.

```
/** Ability System Component. Required to use Gameplay Attributes and
Gameplay Abilities. */
UPROPERTY(VisibleDefaultsOnly, BlueprintReadOnly, Category = "Abilities")
UAbilitySystemComponent* AbilitySystemComponent;
```

Copy full snippet

4. In the source file for your Actor, write the `GetAbilitySystemComponent` function. Since the Ability System Component is stored on the Actor, the function will be very short, like this:

```
UAbilitySystemComponent* AMyActor::GetAbilitySystemComponent() const
{
return AbilitySystemComponent;
}
```

Copy full snippet

Advanced Usage Scenarios

You can set up an Actor to use an Ability System Component that belongs to another Actor, such as a Pawn that uses an Ability System Component that its Player State owns. To do this, the Actor's `GetAbilitySystemComponent` function must retrieve the Ability System Component from the owner, or you must cache it on the Actor in advance. This happens most often in projects where player-controlled Actors can be destroyed and respawned, and the player needs certain Gameplay Ability System information, such as money, points, or long ability cooldowns, to persist. It can also be used in projects where Actors attach other Actors to themselves to represent equipment or modular machine or body parts; in these cases, Gameplay Ability System interactions with the attached Actors can route to the main Actor's Ability System Component. A simple way to accomplish this to have the attached Actor's `GetAbilitySystemComponent` function pass through to the the main Actor; for better performance, consider maintaining a cached pointer while your Actor is attached to (or possessed by) another Actor.

Although the Gameplay Ability System supports multiple Actors sharing a single Ability System Component, it does not support a single Actor having multiple Ability System

Components. Doing this would result in ambiguity when querying, applying changes to an Actor's Ability System Component, or even retrieving the Component from the Actor.