

Gameplay Effects

Overview of Gameplay Effects within the Gameplay Ability System.



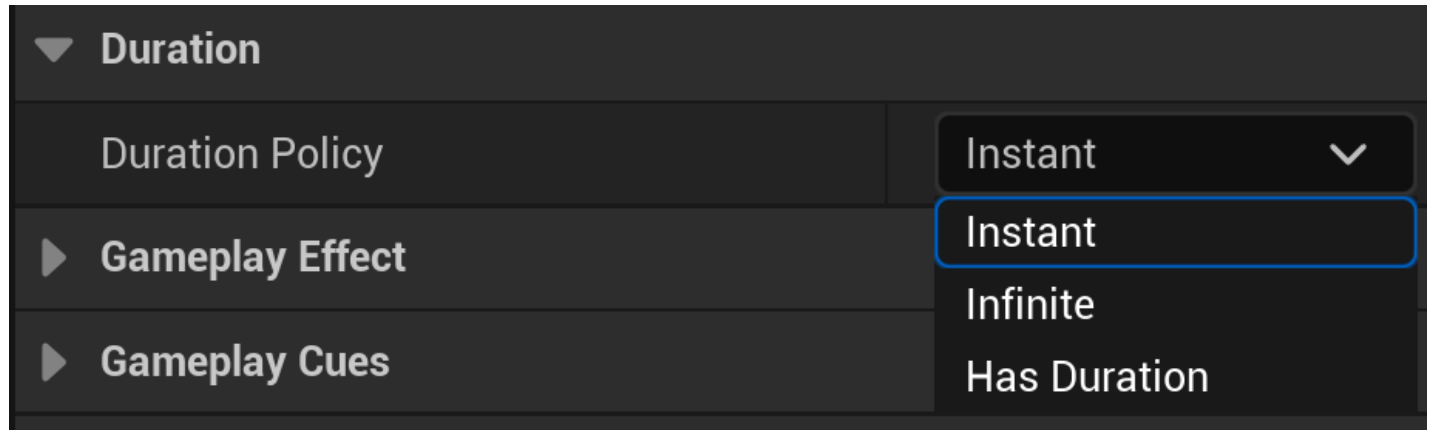
The Gameplay Ability System uses **Gameplay Effects** to change Attributes on Actors that are targeted by Gameplay Abilities. Gameplay Effects consist of function libraries that you can apply to Actor Attributes. These can be instant effects, such as applying damage, or persistent effects, such as poison, that damage a character over time.

- You can use Gameplay Effects for buffs and debuffs.
 - Make your characters stronger or weaker depending on the design of your game.
- Gameplay Effects are assets and therefore are immutable at runtime.
 - Some exceptions exist, such as when a Gameplay Effect is created at runtime, but the data is not modified once created and configured.
- **Gameplay Effect Specs** are the runtime versions of Gameplay Effects.
 - They are the instanced data wrapper around the Gameplay Effect (it is an asset).
Typically when working with Gameplay Effects at runtime, such as creating Blueprint graphs, you deal with Gameplay Effect Specs rather than Gameplay Effects. For instance, the Ability System Blueprint Library uses Gameplay Effect Specs extensively.

The Blueprint functionality concerns itself with Gameplay Effect Specs rather than Gameplay Effects, which is reflected in the Ability System Blueprint Library.

Gameplay Effect Lifetime

A Gameplay Effect has a **Duration** that can be set to **Instant**, **Infinite**, or **Has Duration**. Gameplay Effects that have durations are added to the **Active Gameplay Effects Container**. The Active Gameplay Effects Container is part of the Ability System Component.



▼ Duration

Duration Policy: Instant ▼

▶ Gameplay Effect: Instant

▶ Gameplay Cues: Infinite

Has Duration



- A Gameplay Effect that is instant is declared "Executed." It never makes its way into the Active Gameplay Effects Container.
- In the case of both instant and duration, the term used is "Applied." For example, the method `CanApplyGameplayEffect` does not consider whether it's instant or has duration.
- Periodic effects are executed at every period; therefore, it is both "Added" and "Executed".

The table below lists the properties of Gameplay Effects that you can adjust:

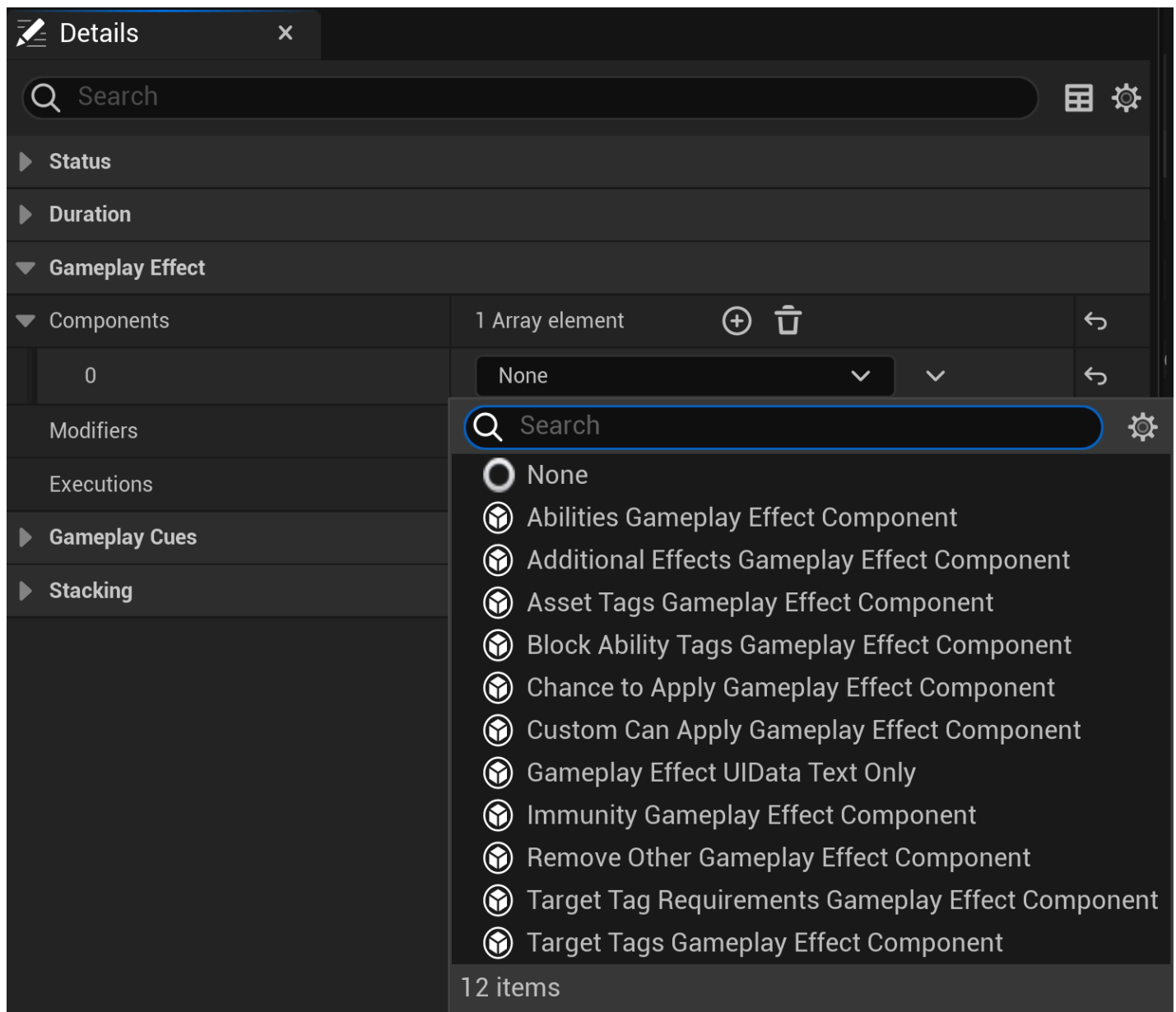
Property	Description
Duration	Gameplay Effects can apply instantly (such as Health decreasing when attacked), over a limited duration (such as a movement speed boost that lasts for a few seconds), or infinitely (such as a character naturally regenerating magic points over time). Effects that have a non-instant duration can apply themselves at different intervals. These intervals can change how the Effect works regarding gameplay and the timing of audio or visual effects.
Components	Gameplay Effect Components define how a Gameplay Effect behaves. For a complete list of available components See [#GameplayEffectComponents]

Property	Description
Modifiers	<p>Modifiers determine how the Gameplay Effect interacts with Attributes. This includes mathematical interactions with Attributes themselves, such as increasing an armor rating attribute by 5 percent of its base value, and includes Gameplay Tag requirements to execute the Effect.</p>
Executions	<p>Executions use the <code>UGameplayEffectExecutionCalculation</code> to define custom behaviors that the Gameplay Effect has when it executes. These are particularly useful for defining complex equations that aren't adequately covered by Modifiers.</p>
Gameplay Cues	<p>Gameplay Cues are a network-efficient way to manage cosmetic effects, like particles or sounds, that you can control with the Gameplay Ability System. Gameplay Abilities and Gameplay Effects can trigger Gameplay cues. Gameplay Cues act through four main functions that can be overridden in native or Blueprint code:</p> <ul style="list-style-type: none"> • On Active • While Active • Removed • Executed (used only by Gameplay Effects). <p>All Gameplay Cues must be associated with a Gameplay Tag that starts with <code>GameplayCue</code>, such as <code>GameplayCue.ElectricalSparks</code> or <code>GameplayCue.WaterSplash.Big</code>.</p>
Stacking	<p>Stacking refers to the policy of applying a buff or debuff (or Gameplay Effect) to a target that already carries it. It also covers handling overflow, where a new Gameplay Effect is applied to a target that is already fully saturated with the original Gameplay Effect (such as a poison meter that builds up, resulting in damage-over-time poison only after it overflows). The system supports a wide variety of Stacking behaviors, such as:</p> <ul style="list-style-type: none"> • Building effects until a threshold is broken. • Maintaining a "stack count" that increases with each fresh application up to a maximum limit. • Resetting or appending time on a limited-time Effect. • Applying multiple instances of the Effect independently with individual timers.

Gameplay Effect Components

Gameplay Effects contain **Gameplay Effect Components** (GEComponents) to determine how a Gameplay Effect behaves. Gameplay effects can:

- Alter the [Gameplay Tags](#) of the Actor to which a Gameplay Effect is applied, or remove other active Gameplay Effects based on conditions.
- You can create your own game-specific Gameplay Effect Components, which help extend the usability of Gameplay Effects. Instead of providing a larger API for all desired functionality, the implementer of a Gameplay Effect Component must read the Gameplay Effect flow carefully and register any desired callbacks to achieve the desired results. This limits the implementation of Gameplay Effect Components to native code.
- GEComponents live within a Gameplay Effect, a data-only Blueprint asset. Therefore, like Gameplay Effects, only one GEComponent exists for all applied instances.



The Table below contains a complete list of available Gameplay Effect Components:

Gameplay Effect Component	Description
UChanceToApplyGameplayEffectComponent	Probability that the Gameplay Effect is applied.
UBlockAbilityTagsGameplayEffectComponent	Handles blocking the activation of Gameplay Abilities based on Gameplay Tags for the Target Actor of the owner Gameplay Effect.
UAssetTagsGameplayEffectComponent	Tags the Gameplay Effect Asset owns. These do <i>not</i> transfer to any Actors.

Gameplay Effect Component	Description
UAdditionalEffectsGameplayEffectComponent	Add additional Gameplay Effects that attempt to activate under certain conditions (or no conditions.)
UTargetTagsGameplayEffectComponent	Grant Tags to the Target (sometimes referred to as the Owner) of the Gameplay Effect.
UTargetTagRequirementsGameplayEffectComponent	Specify tag requirements that the Target (owner of the Gameplay Effect) must have if this GE should apply or continue to execute.
URemoveOtherGameplayEffectComponent	Remove other Gameplay Effects based on certain conditions.
UCustomCanApplyGameplayEffectComponent	Handles the configuration of a CustomApplicationRequirement function to see if this GameplayEffect should apply.
UImmunityGameplayEffectComponent	Immunity is blocking the application of other GameplayEffectSpecs.

Gameplay Attributes

A **Gameplay Attribute** contains a measurement of an Actor's current state that can be described by a floating-point value, such as:

- health points
- physical strength
- movement speed
- resistance to magic,

and so on. Attributes are declared as UProperties of FGameplayAttributeData type within Attribute Sets, which both contain Attributes and supervise any attempts to modify them.



Attributes and Attribute Sets must be created in native code — they cannot be created in Blueprints.

Creating an Attribute Set

Follow the steps below to create an Attribute Set

1. Inherit your class from `UAttributeSet`, then add the Gameplay Attribute Data members tagged with `UPROPERTY`. For example, an Attribute Set containing only a "health" Attribute would look like this:
2. Once you have created an Attribute Set, you must register it with the Ability System Component. You can either add the Attribute Set as a subobject of the Ability System Component's owning Actor, or pass it to the Ability System Component's `GetOrCreateAttributeSubobject` function.

Programming Effect and Attribute Interaction

There are several functions that an Attribute Set can override to deal with the way that an Attribute responds when a Gameplay Effect attempts to modify it. As an example, the "Health" Attribute from the sample `USimpleAttributeSet` can store a floating-point value, and that value can be accessed or changed by the Gameplay Ability System. Currently, nothing actually happens when the Health value drops to zero, and there is nothing to prevent it from dropping below zero.

To make the "Health" Attribute behave the way you want it to, the Attribute Set itself can step in by overriding several virtual functions that handle attempted modifications to any of its Attributes.

The following functions are commonly overridden by Attribute Sets:

Function Name	Purpose
PreAttributeChange / PreAttributeBaseChange	These functions are called just before modifications to an Attribute. They are intended to enforce rules on the Attribute's value, such as "Health must be between 0 and MaxHealth", and should not trigger in-game reactions to Attribute changes.
PreGameplayEffectExecute	Just before modifying an Attribute's value, this function can reject or alter the proposed modification.
PostGameplayEffectExecute	Immediately after modifying an Attribute's value, this function can react to the change. This often includes clamping the final value of the Attribute or triggering an in-game reaction to the new value, like dying when the "health" Attribute falls to zero.