

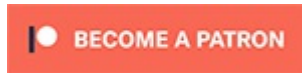
Table of Contents

Introduction	1.1
API	1.2
Attributes	1.2.1
Modifiers	1.2.2
Installation	1.3

Attributes Extension

Attributes Extension is a lightweight plugin that adds a very **intuitive, easy and flexible attribute system** to UE4.

If you like our plugins, consider becoming a Patron. It will go a long way in helping me create more awesome tech!



Why Attributes?

This plugin is not a damage system, or a spells system, not even a combat system. It only, strictly, provides **variable, non-destructive and efficient attributes** to UE4. This is its only task, and it does it very well.

It can be very easily integrated into custom game systems like buffs, areas, spells, abilities, or any other kind of gameplay mechanic.

What can it be used for?

Almost every game needs attributes. They are floats that can be changed, modifiers can be added and removed while ensuring the value is not lost.

Values like *Damage*, *Max Health*, *Speed* or *Attack Rate* usually need to be attributes in games like Skyrim, TBOI, World Of Warcraft... well a LOT of games.

What I want to reach with this is, that every game is different, gameplay systems like "Spells" or "Effects" are never the same, but attributes are. They share a common start point.

Has it been used before, is it stable?

We like to share the tools we create for our own.

Attributes have been used for more than 2 years internally and between different projects.

Is it performant?

It really is. Attributes avoid copies or extra operations, values are calculated once, so no matter how many you use they will be performant friendly.

Networking?

Yes! Attributes are super optimized and only take a maximum of 12 bytes while replicating.

This basically means you can replicate thousands at the same time!

Test Project

You can download a **Test project** from the marketplace page to see and test the Plugin.

Attributes

Types

There are two supported types:

Int32 Attr

Attributes using Int32 as the value type.

Float Attr

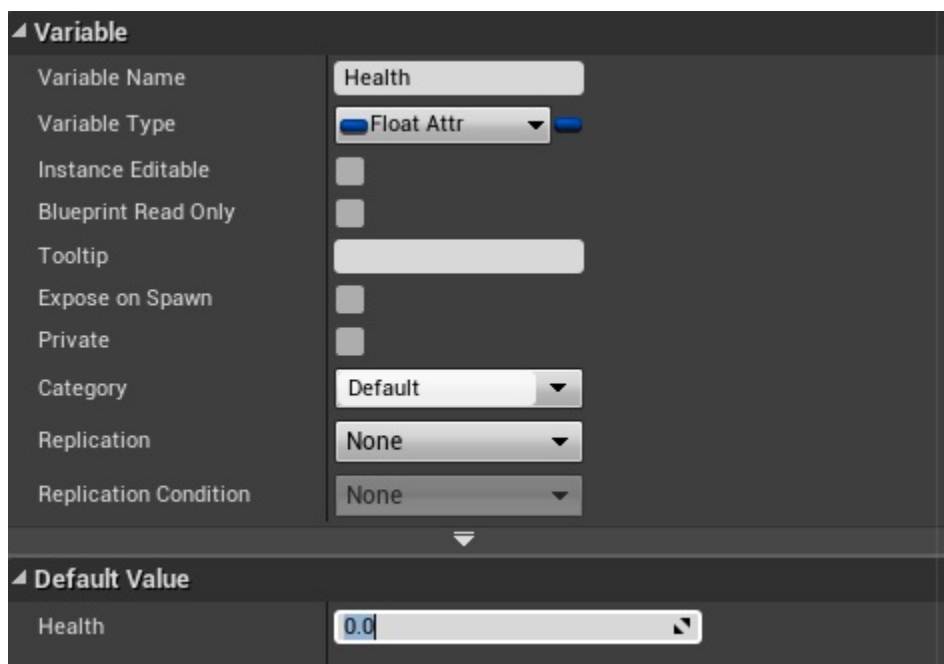
Attributes using Float as the value type.

Usage

Note that Int32 and Float share the exact same API.

Creating an attribute

To create a float attribute we just need to create a variable of type *"Float Attr"* or *"Int32 Attr"*.



The screenshot shows the 'Variable' and 'Default Value' sections of the Unreal Engine editor. The 'Variable' section is expanded, showing the following settings:

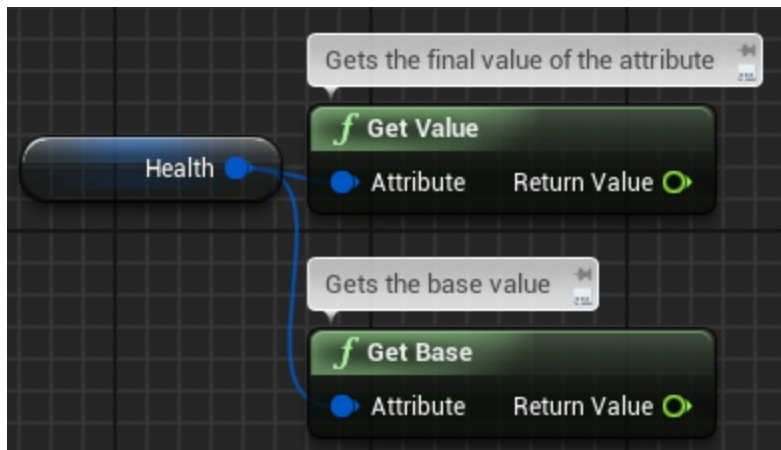
- Variable Name: Health
- Variable Type: Float Attr (selected from a dropdown)
- Instance Editable: ☐
- Blueprint Read Only: ☐
- Tooltip: (empty text field)
- Expose on Spawn: ☐
- Private: ☐
- Category: Default (selected from a dropdown)
- Replication: None (selected from a dropdown)
- Replication Condition: None (selected from a dropdown)

Below the 'Variable' section is the 'Default Value' section, which is also expanded. It shows a single entry for the 'Health' variable with a value of '0.0' in a text field.

We can now edit the base value of the attribute.

Read base and final values

The next two functions expose this values:



Modifiers

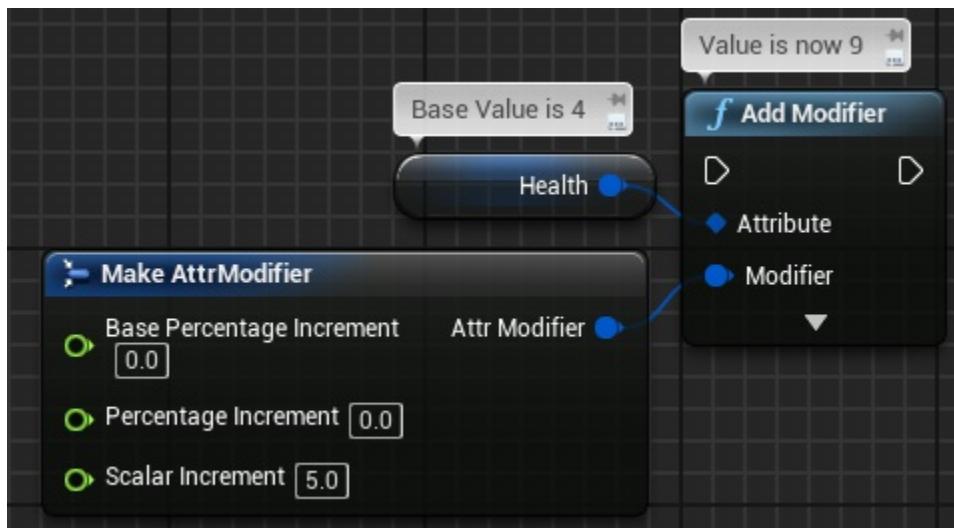
Modifiers change the base value of an attribute depending on 3 different factors.

Modifier Factors

My Modifier	
Base Percentage Increment	0.0
Percentage Increment	0.0
Scalar Increment	0.0

Scalar Increment

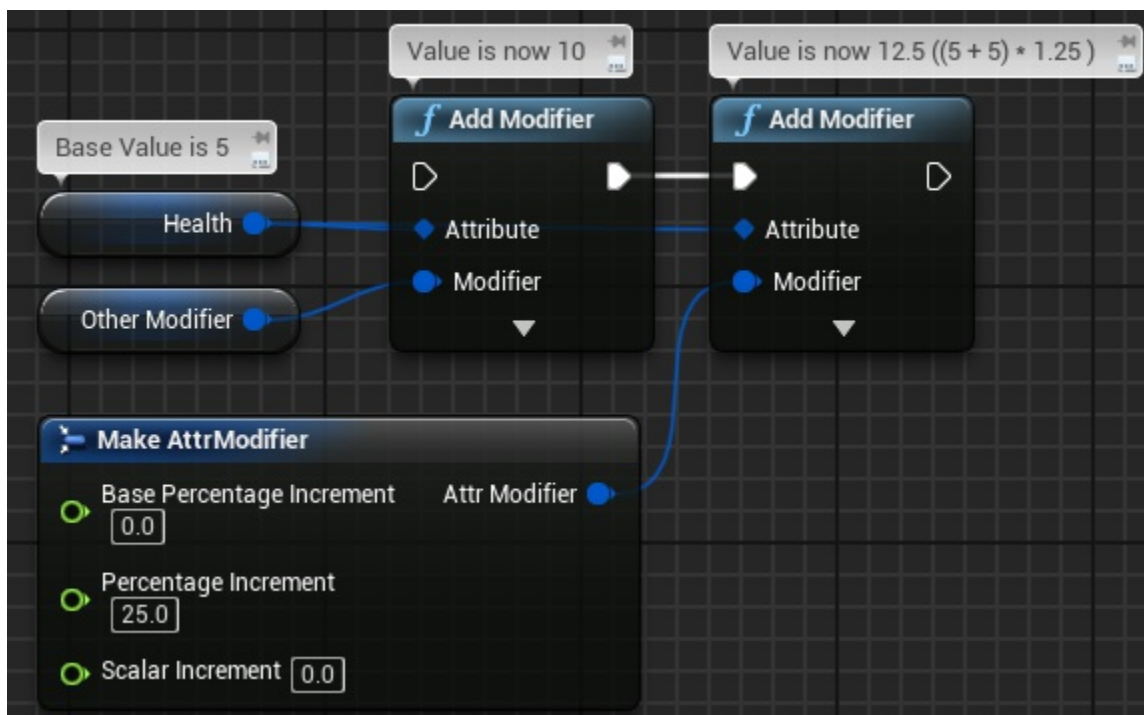
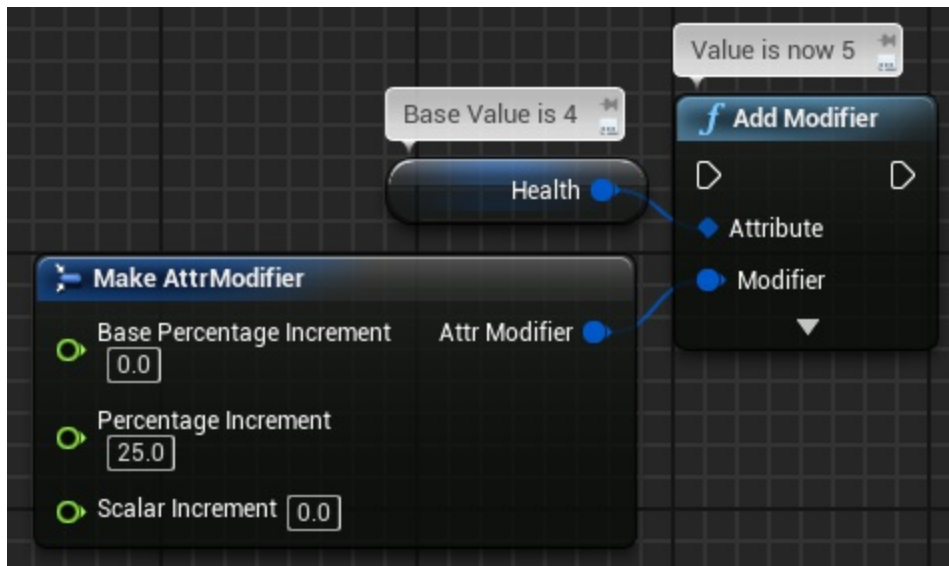
Adds a value directly to the attribute.



Modifiers should usually be used from a variable (of type *Attr Modifier*) if you want to be able to remove them from Attributes

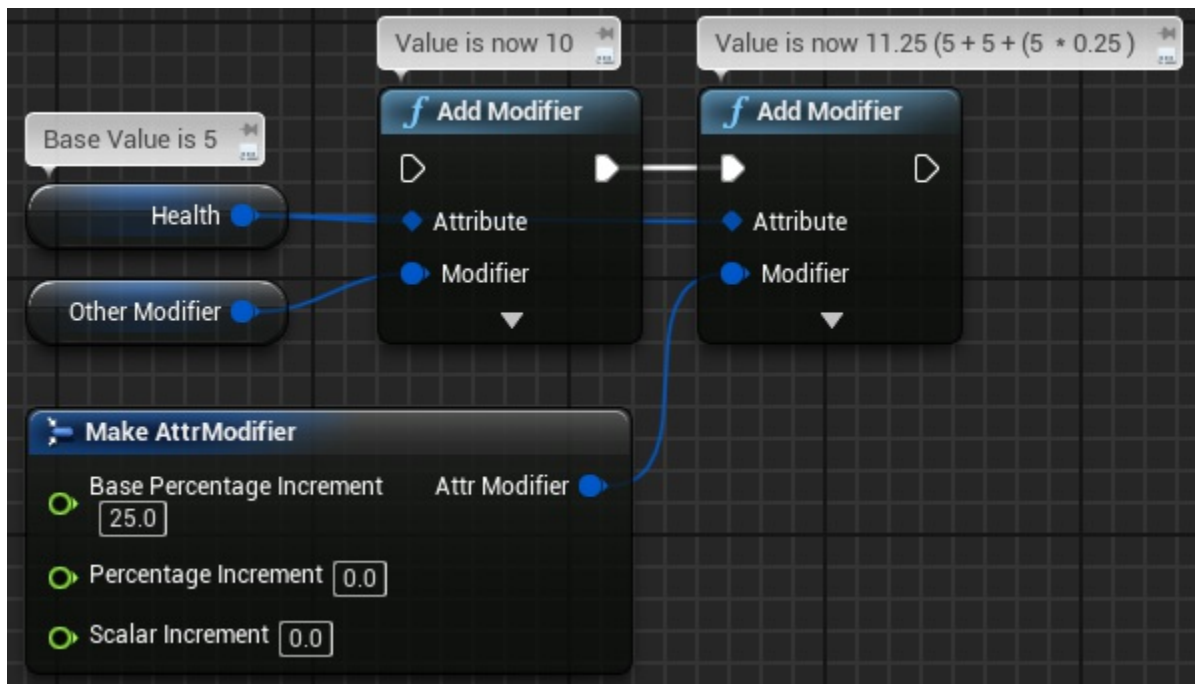
Percentage Increment

Adds a percentage of the last value of the attribute.



Base Percentage Increment

Similar to Percentage except that this percentage is based on the original value.



Application order

Modifiers are applied into an attribute following the next rules of priority:

1. **Modifier Category** - Check [Modifier Categories](#)
2. **Order** - The order at which modifiers are applied.

Adding "ModA" and "ModB" to the same category will result in "ModA" being applied before.

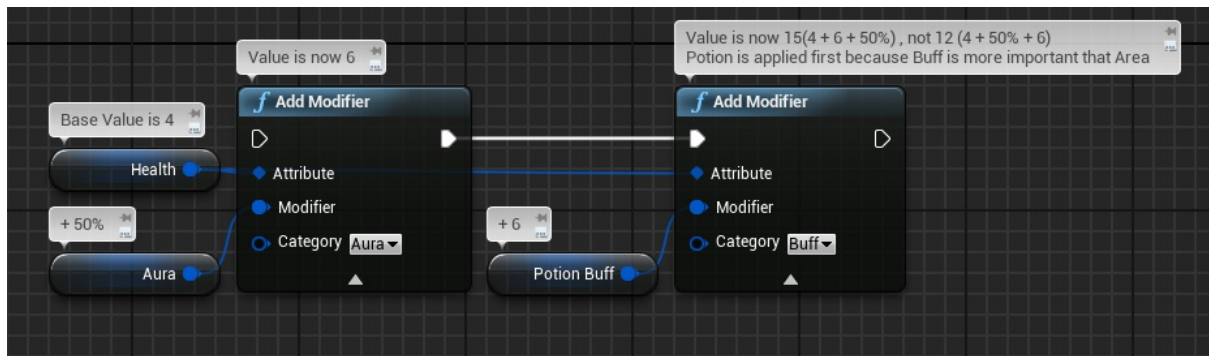
In a future release: Categories may specify if attributes should apply first last mods on the same category.

Modifier Categories

Modifier categories are used to specify **modifier application order**. Depending on the genre of a game this can be a key feature that we didn't want to miss.

With a configuration where "Buff" is more important than "Aura", a "Buff" attribute will be applied before an "Aura" modifier.

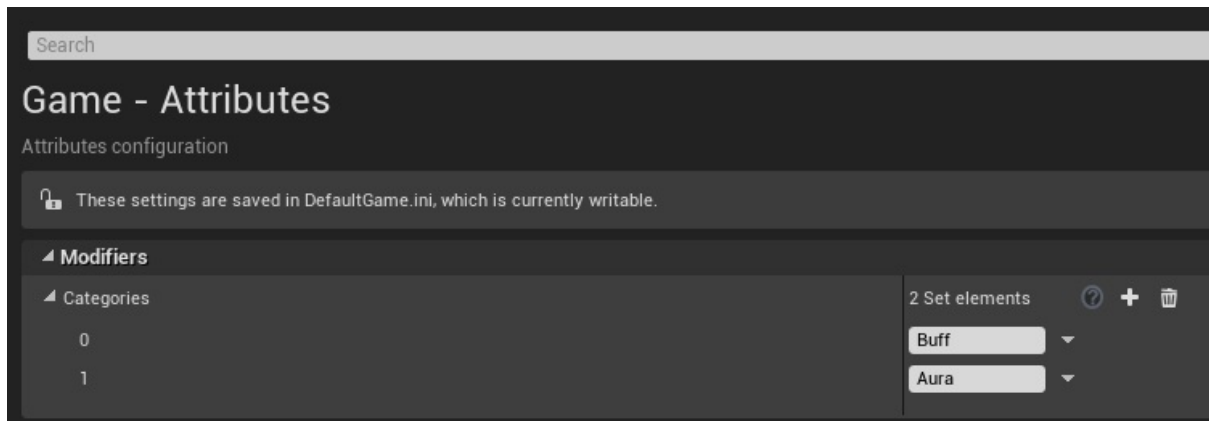
For example:



Categories can also be stored as variables of type "Attr Category"

Adding & Removing Categories

Categories can be edited from **Project Settings -> Game -> Attributes**. Remember, their order matters. First categories are applied first on attributes.



Categories **can't** be modified in runtime.

Installation

From Marketplace

1. Install from the launcher: [AVAILABLE HERE](#)
2. Enable the plugin from your project's plugin manager

Manually

This are the general steps for installing the plugin directly into your project:

1. Install from the launcher: [AVAILABLE HERE](#)
2. Copy the folder "*AttributesExtension*" from your engine's plugins folder into the **plugins folder** of your existing project (e.g "*MyProject/Plugins*")
3. Done! You can now open the project