Damage

# Meta Attributes

So far the gameplay effects that modify Health are reaching into the attribute set and changing the value of Health directly. Typically in RPG games with GAS this is not how it is done, because applying damage to an enemy there are a number of calculations that need to be made, and those calculations have to do with attributes on the attacker and victim. For this reason, to keep the mathematics simple, we often use a placeholder attribute that behaves as an intermediary that allows us to perform all the necessary maths before setting the value of the health, in the case of damage

These intermediary values are called **Meta Attributes**

Normal attributes:

* Often Replicated
* Changed on server and replicated to clients

Meta attributes:

* Temporary
* Not replicated
* Only used on server
* Allows for calculations

Damage:

* Meta Attribute would be *Incoming Damage*

How Incoming Damage works:

A GE causes 11 damage to a target. Instead of subtracting 11 health, the GE instead adds 11 to Incoming Damage

There are ways t respond to attribute changes in the attribute set eg PostGameplayEffectExecute is used to check if the attribute that has changed is the Incoming Damage Meta Attribute. If so, we then calculate if there was a Block, a Crit, is there an Intelligence/Strength Bonus. Whatever, we perform those calculations to determine the final real amount of damage actually inflicted and then subtract that number from Health

A screenshot of a computer

Description automatically generated

The Gameplay Effect is not the arbiter of Health subtracted from the attribute, it’s the rules set by those equations that determines the final value

The Incoming Damage is set by the GE, and the Attribute Set performs the calculations it needs to and then zero out the Incoming Damage Meta, Set the Value of Health and then respond in any other ways needed eg floating text (which may be a different colour depending on bock or crit), apply debuff effects, damage over time, stuns, hit react anim

Meta attributes are therefore often the way that GAS projects handle damage

# Damage Meta Attribute

In GE\_Damage:

A black and white striped background

Description automatically generated

This is an Instant Gameplay Effect that adds a negative value directly to Health. This is really a placeholder Gameplay Effect – it hardcodes a Negative 10, doesn’t use a curve table

Instead we’ll modify a meta attribute, then allow the Ability Set to process the data. We’ll also use the damage meta in the custom calculations as a custom calculation class as Damage will be one of the more complicated types of modifiers

We’ll add the Meta to the Attribute Set

A screen shot of a computer program

Description automatically generated

Easy enough. Note that the UPROPERTY includes a category and BP Readonly, like the other attributes, but there is NO ReplicatedUsing, because this is not a replicated attribute

So how do we actually **use** the meta attribute?

In the CPP:

A computer screen with text

Description automatically generated

Here we can see it the attribute data has the attribute IncomingDamage, in exactly the same way we check for GetHealthAttribute and GetManaAttribute

A screen shot of a computer program

Description automatically generated

So, inside the if check we know we are getting a change to the IncomingDamage Meta Attribute

As IncomingDamage is a Meta, it should be used for it’s value and then reset to zero – we will consume that data!

To do this we will make a local float variable that has the value IncomingDamage

A black screen with white text

Description automatically generated

So we’re caching that value locally and then we can set that meta attribute to zero

A computer screen with white text

Description automatically generated

IncomingDamage is now ready to take a new value, because we have received the data into this function

Now we have the Local incoming damage we can decide what to do with it, and we should only do something with it if its value is greater than zero

So far we have nothing to do with the Meta Attribute other than set the health value, so we can do that for now and handle any additional clamping

A screen shot of a computer program

Description automatically generated

At this point, we can tell certain things about the damage that has been done; eg if NewHealth is 0 then we know that the damage done is fatal. So we can do something like this:

A screen shot of a computer

Description automatically generated

If NewHealth is equal to or less than 0 we know that bFatal will be true - enough damage was done to kill the owner of this attribute set

So this is basically how it should work – we use the meta instead of changing health and the gameplay effects that currently affect health will be changed to affect the Meta on the target, and they will be adding to it, not subtracting, because the subtraction is done here in NewHealth – LocalIncomingDamage

In GE\_Damage:

A screenshot of a computer

Description automatically generated

The Attribute is now set to Incoming Damage and we’ll increase the damage a bit for testing

A video game with a firework

Description automatically generated with medium confidence

The bar is going down; we’re doing damage with a meta attribute, which allows us to do all kinds of things; we can check the damage done, perform custom calculations

There’s no hit react or death at zero, but we’ll worry about that later

GE looks better but we’re still hard-coding 25 for the damage – would be better if this were linked to the Gameplay Ability somehow, so if the Gameplay Ability has it’s own damage variable, how do we make sure that the Gameplay Effect is using the Damage from the Gameplay Ability?

That would be the 4th Magnitude type:

A screen shot of a message

Description automatically generated

# Set By Caller Magnitude

A Gameplay Ability may have a base damage value that we’d like to set on the Gameplay Effect that the ability is causing

Currently the Effect Spec when launching a projectile is handled like this:



We are creating a Gameplay Effect Spec with MakeOutgoingSpec and then taking the SpecHandle and setting that on the projectile – the projectile has a Damage Effect Spec Handle variable that we are setting

So we’re setting the Damage Gameplay Effect for the projectile spell, but as soon as we set the Spec Handle for the Projectile – which will apply it later – that Spec Handle should carry the information for how much damage to cause. We should be able to set that here from our ability, if the ability wants to have some value – the ability itself should determine how much damage to do, not the effect

To do this we can use the Set By Caller option on the Gameplay Effect

Set By Caller Magnitudes are Key/Value pairs; where the key is a gameplay tag, so we need a gameplay tag to identify the Set By Caller magnitude

First we’ll make a Gameplay Tag for Damage in the AuraGameplayTags singleton

A screen shot of a computer

Description automatically generatedA black screen with blue and green text

Description automatically generated

Now we have a native Gameplay Tag for Damage and we can use it in code

In AuraProjectileSpell.cpp:



We’ll Get the gameplay tags right before applying the Spec Handle:

A screen shot of a computer

Description automatically generated

And now we can use a Set By Caller magnitude

This is done using the AbilitySystemBlueprintLibrary

A screenshot of a computer

Description automatically generated

Assign Tag set By Caller Magnitude requires a Gameplay Effect Spec Handle, a Tag and a Magnitude

We’ll use SpecHandle for this.

Next it requires a Gameplay Tag called DataTag – this is how we identify the magnitude value, because you can assign multiple SetByCaller magnitudes to a gameplay effect spec.

We’ll set a Key/Value pair, where the Key is the Damage tag

And finally the magnitude is whatever we set here; the projectile spell could have it’s own Damage variable.

For the moment, as a proof of concept, we’ll hardcode 50



So now the spec handle is carrying along a key/value pair – the Damage gameplay tag and the value 50

Now we just need to access that key/value pair from within the Gameplay Effect when applying a modifier

In GE\_Damage:

A black and white background

Description automatically generated with medium confidence

Once Set By Caller is applied there’s a new dropdown. Because we can have multiple Set By Caller magnitudes on a gameplay effect spec we need to choose which one, and they are identified by the gameplay tag key

A black and white screen

Description automatically generated

As soon as we pick this tag, this modifier will do 50 damage, because that’s the value we associated with this key when we called AssignTagSetByCallerMagnitude

A screenshot of a video game

Description automatically generated

So the damage is now controlled by the Gameplay Ability, not the Gameplay Effect (even if we’re still hardcoding for the moment). We can set a base damage property based on the Gameplay Ability’s level. So we can add a member variable to the AuraGameplayAbility class to handle damage

# Ability Damage

We’re currently hardcoding 50.f for the set by caller magnitude in the effect spec handle set on the projectile

Aura projectile spell is going to control the damage and rather than hardcode this we can have a variable. We know that abilities have the concept of level, so if the ability level increases maybe we want the damage to increase as well, and we’ve seen a good way to handle increasing values based on levels – curve tables

We’ve also seen that when we select a curve table in a gameplay effect, for example

A screenshot of a computer

Description automatically generated

Here we’re using a scalable float magnitude where we have selected a curve table (CT\_Potion), choosing a curve from that table (HealingCurve) we can enter a value and the scalable float magnitude will be scaled by the value in that curve based on a level

We can do something like this in our gameplay ability – the scalable float mag is an FScalableFloat, and when you have a member variable of this type you get these properties – Ability to have magnitude, a curve table dropdown and curve and a preview slider

It would be good to have one of these useful properties in the Gameplay Ability class

It’s true that we could have abilities that do not do damage, and we could subclass AuraGameplayAbility to have an AuraDamageAbility from which all damaging abilities inherit, but for simplicity we’ll just give this to AuraGameplayAbility and then any abilities which do not do damage we simply won’t use that property – the memory it takes up is negligible

A computer screen shot of a program

Description automatically generated

We’re adding the property here for damage for the ability, which will be a scalable float and we can create curve tables and set them per ability

A screen shot of a computer

Description automatically generated

In editor GA\_FireBolt now has an FScalableFloat  
  
A screenshot of a computer

Description automatically generated

Entering a value here will scale by the curve table selected. We could enter a base value of 10 and then scale it, or set it to 1 and then set the curve for values for levels

Now we can set a damage curve table to scale Firebolt, and create different curve tables for different abilities so they scale differently when level increases!

A black and white text

Description automatically generated

A screenshot of a computer

Description automatically generated A graph with a line going up

Description automatically generated

A screenshot of a computer

Description automatically generated

In AuraProjectileSpell.cpp:



We will replace the 50.f with a query for the value at level so:

A screenshot of a game

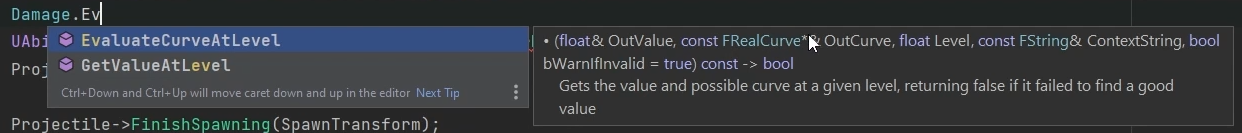
Description automatically generated

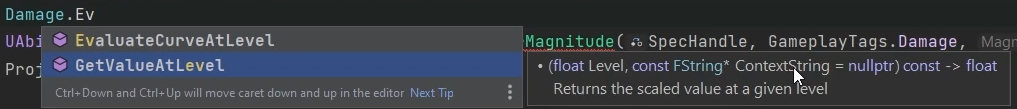
This returns an Int32

For an Integer we could use:



Or possibly:



This gets the value and possible curve; We would need to pass in an FRealCurve pointer by reference; or better for this:  




This \*does\* return an int32 instead of a float, but it performs an implicit conversion so we can just pass in this expression



However it’s better if we make a const float and pass that in



For visibility will add a screen message at this point:

A blue sky with red text

Description automatically generated

Nb We see the damage printed immediately, before it hits, because we’re printing it in the DamageEffectSpec

If we change the value 

We get more damage

A blue sky with red text

Description automatically generated