RPG Attributes

# Initialise Attributes from a Data table

So far attributes have been initialised by calling init functions. This works, but it’s not the only way. Here’s an alternative

To make things more interesting we’ll add some more attributes to the AttributeSet

A screenshot of a computer program

Description automatically generated

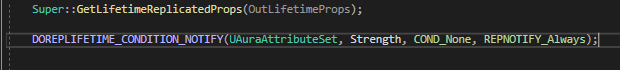
Example: Strength. Just copying the vital but changing the specifics. Note the On\_Rep for a notif that does not yet exist

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated



And then we apply for the other attributes

A screen shot of a computer program

Description automatically generated

So, the purpose of this exercise is to initialise from a datatable. To do this we need to expose the ability system component to BP, so that we can set a specific asset, the datatable, on that component

To od this we need to open the Aura Player State, where the ASC is located

A computer screen with white text

Description automatically generated

A black background with white text

Description automatically generated

The ASC properties are now exposed in the player state:

A screenshot of a computer

Description automatically generated

To use a data table to initialise some attributes, we have to add an element to the array Attribute Test called Default Starting Data

A screenshot of a black box

Description automatically generatedA screenshot of a computer

Description automatically generated

You can choose attributes

A screenshot of a computer

Description automatically generated

And a data table to use

A screenshot of a computer

Description automatically generated

But you can’t just use ANY data table – it needs the correct row structure.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Row name is specified by the Attribute Set, a dot, and the attribute name



Base value:

A black screen with white dots

Description automatically generated

And now back in the player state:

A screenshot of a computer

Description automatically generated

And now in-game:

A screenshot of a phone

Description automatically generatedA screenshot of a computer screen

Description automatically generated

A screenshot of a phone

Description automatically generated

The struct:

From the parent AttributeSet.h:

A screenshot of a computer program

Description automatically generated

Currently it’s mostly useful for setting the initial value of attributes; it implements no functionality for a minimum or maximum, it just inits the values

It’s limited in application, but this can be pretty useful for some cases; You just fill in the values and the ASC does the rest

Most people prefer to initialise using a Gameplay Effect on game start though

# Initialising Attributes with Gameplay Effects

After disabling the data table previously used, go to AuraCharacterBase.h

Protected:

A black background with white text

Description automatically generated

So, we have a DefaultPrimaryAttributes, now we need a way to apply these attributes

A screenshot of a computer

Description automatically generated

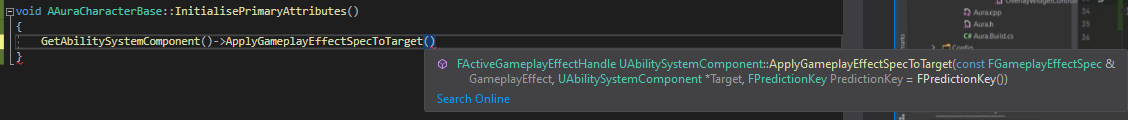
How to define.

First, if we’re going to call something like ApplyGameplayEffectToSelf() or toTarget, we’ll need a GameplayEffectSpec to apply. Which we can create because we have a class for our DefaultPrimaryAttributes

So work backwards.

Call the fn to apply the effect

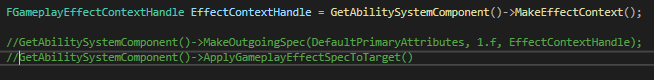
We can always get our AbilitySystemComponent, assuming that it’s valid at the time of calling, and from it call ApplyGameplayEffectToTarget()



To apply the Gamepla Effect we need a GameplayEffect Spec. Asa reminder, the function for this is MakeOutGoingSpec(), taking in a UGameplayEffect, a level and a context handle.



To get a context handle:

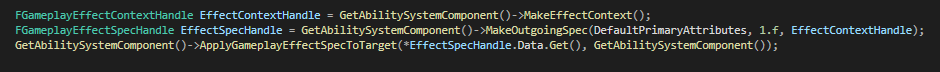


Now we have our Context we can pass it into the next line:

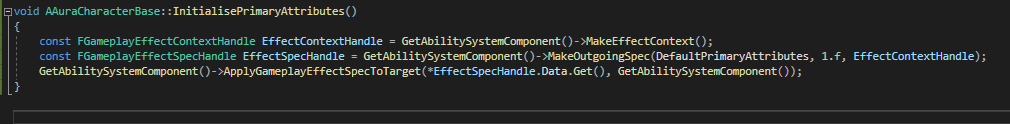


This returns a Spec Handle, which we can pass into the next time. However, it’s not enough to just pass in the handle, we need to use the internal variable Data. Data is a wrapper, so we need to then call Get(). This returns a pointer, but the function does not take in a pointer, so we have to dereference it!

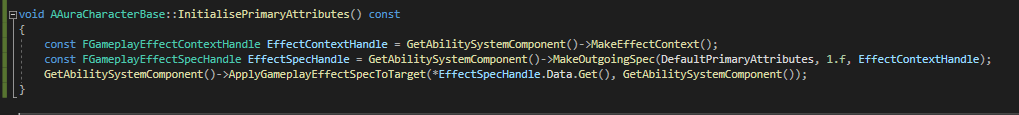
Finally the second argument required is the ASC



Finally these can be made consts



This applies a GameplayEffectSpec to a target Abilitysystemcomponent



There should be no problem as long as all these values are correct, and we can check them

A screen shot of a computer

Description automatically generated

Now we have the function, where do we call it?

It can be called in different circumstances, depending on if we are the aura character or the enemy

For the moment, we’ll do it for the character

Since we created this effect in the base character class we can inherit it in the character class

Unlike the InitAbilityActorInfo this does not need to be done on both sever and client, it only needs to be done on the server, because all of the attributes are marked to be replicated, so if we change them on the server they change on the client as well. We can do it locally as well to save waiting for replication, either is fine

If we do this in InitAbilityActorInfo, we know that the ASC is valid at this point. So we know it’s safe to call it here; lets do it after the Hud

A screen shot of a computer program

Description automatically generated

Now all that remains is to make sure we have that effect and set some default values

Run in Debug mode

Error: 1 Remember to forward declare the UGameplayEffect class

Error 2: DefaultPrimaryAttributes is not a function! Change line to:

Check(DefaultPrimaryAttributes);

Now in the editor we need a GameplayEffect to initialise the PrimaryAttribute values

This will be specific to the Aura Character, and eventually we’ll have different types of characters that start with different values

For now, we’ll create just 1 GE; an instant, applied once, with 4 modifiers, one for each primary attribute

A screenshot of a computer

Description automatically generated

Modifier opp is not add, it’s Override

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Once this is done we need to set the GE effect on the aura character(otherwise game will crash)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Attribute Based Modifiers

So far all GEs are using scalable floats for modifier magnitudes

A screenshot of a computer

Description automatically generated

But there are other options:

A screenshot of a computer

Description automatically generated

The one we’re looking at right now is Attribute Based

A screenshot of a phone

Description automatically generated

Ability to have an attribute change based on other attribute values

Easy to have an attribute that derives from other attributes

New actor that applies a test effect that uses attribute modifier magnitude calculations

So, create new actor and GE:

A screenshot of a computer

Description automatically generated

In TestActor:

Add Box collision, set collision box to visible and make the lines a bit chunker, maybe 5 thickness.

A screenshot of a computer

Description automatically generated

Set applied effect to:

A screenshot of a computer

Description automatically generated

In GE Test AttributeBased:

A screenshot of a computer

Description automatically generated

Expand Backing Attribute

A screenshot of a computer

Description automatically generated

Effect is applied from something to something, so we can capture the attribute effect from the source or from the target

A screenshot of a computer

Description automatically generated

In this case capturing Target

For later: snapshot controls when to capture the attribute, when should we use that attribute’s value? When the effect was created, or when it was applied? Snapshot determines this

Attribute to capture:  
A screenshot of a computer

Description automatically generated

We’re taking from the target (Aura) so we’ll be using one of her attributes

If we pick Vigor then we will be applying the value to health with an Add opp, so Health is now Health + vigor

In action:

A screenshot of a grey screen

Description automatically generated

Health 75 default, + 9 vigor is now Health 84

With all the options present here this can get complicated quickly, and we can add multiple modifers as well!

What if we add a second modifier to the effect that also adds Str to health?

A screenshot of a computer

Description automatically generated

How it’s 94!

A blurry image of two blue balls on a checkered floor

Description automatically generated

It’s pretty intuitive when the mods are all Add, you just add them all together. But it’s important to understand how they work and how the order is taken into account!

# Modifier Order of Operations

Adding a third additive Attribute mod based on Resilience boosts the Health over 100; pretty predictable

75 +9 +12 +10 = 106, over the 100 cap

I’ve dropped default health to 10 for the sake of simplicity

So, initialising health to 10. We can add a modifier to a GE effect, choosing an attribute to affect, choose an operation like Add and for Backing attributes we choose vigor itha value of 9. So This means Health will be 10+9 19.

A second modifier targeting health, adding backing attribute strength is another 10, so 29

Similarly adding Resilience is +12, so 41

A screenshot of a computer

Description automatically generated

These happen in order, so the first modifier is applied, then the second then the third

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

But what if we change the Str modifier to multiply?

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

(Health is clamped)

If we comment out all the clamping lines:

A screenshot of a computer

Description automatically generated

What if we now change the Resilience, third modifier to divide?

A screenshot of a computer

Description automatically generated

A screenshot of a calculator

Description automatically generated

A screenshot of a computer

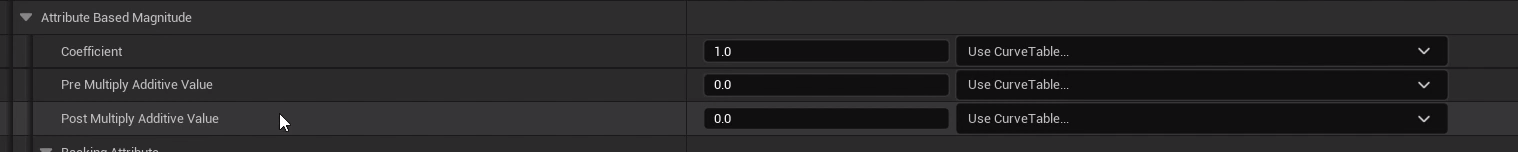
Description automatically generated

Let’s add a 4th attribute, Max Health

A screenshot of a computer

Description automatically generated

So, order of operations is fairly straight forward. What about the effects of these additional values?



# Modifier Coefficients

Just being able to multiple, add, divide or subtract an attribute from another attribute isn’t quite enough!

What if you wanted to Add to health a value of Str \* 10? Or Str \* 0.1? Or Str \* 0.1 + 12?

This is where we do it

So we have a Coefficient, a Pre multiply additive value and a post multiple additive value.

A screenshot of a computer

Description automatically generated

So 10 + 9 is actually 10 + ( Coefficient \* ( 9 + Pre)) + Post !

Pre and Post default to 0 and C defaults to 1

A screenshot of a computer

Description automatically generated

Applying numbers to ALL the modifiers might result in this:

A screenshot of a computer game

Description automatically generated

For a final result of:

A screenshot of a computer game

Description automatically generated

This allows for a nice level of complexity and flexibiilty, but we can’t use arbitrarily complex and arcane calculations (other functions in GAS do allow this)

# Derived Attributes

Now that we know how to apply modifiers to affect attributes, and that those modifications can be backed by other attributes we can create derived attributes: attributes that depend partially or entirely on other attributes.

From a design perspective it’s important to lay out what all the attributes in the game should be doing.

So far we have categorised some attributes as Primary that do no depend on others and are standalone. We can increase or decrease them based on game mechanics but other attributes in some way or another depend on them. Decided at the Design level

A screenshot of a computer

Description automatically generated

As you can see, some secondary attributes are dependent on other secondary attributes.

Also, Max Health and Max Mana are now secondary instead of Vital

To do:

A black and white text with yellow text

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screen shot of text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated