Universal Stats & Attributes System

Table of Contents

For more, visit https://www.youtube.com/playlist?list=PLgRbu73Lib5n9_xg-FbaZRlbCSLujmK1h.

The Universal Stats & Attributes System (USAS) is a versatile plugin that allows you to create and manage your own stats and attributes systems.

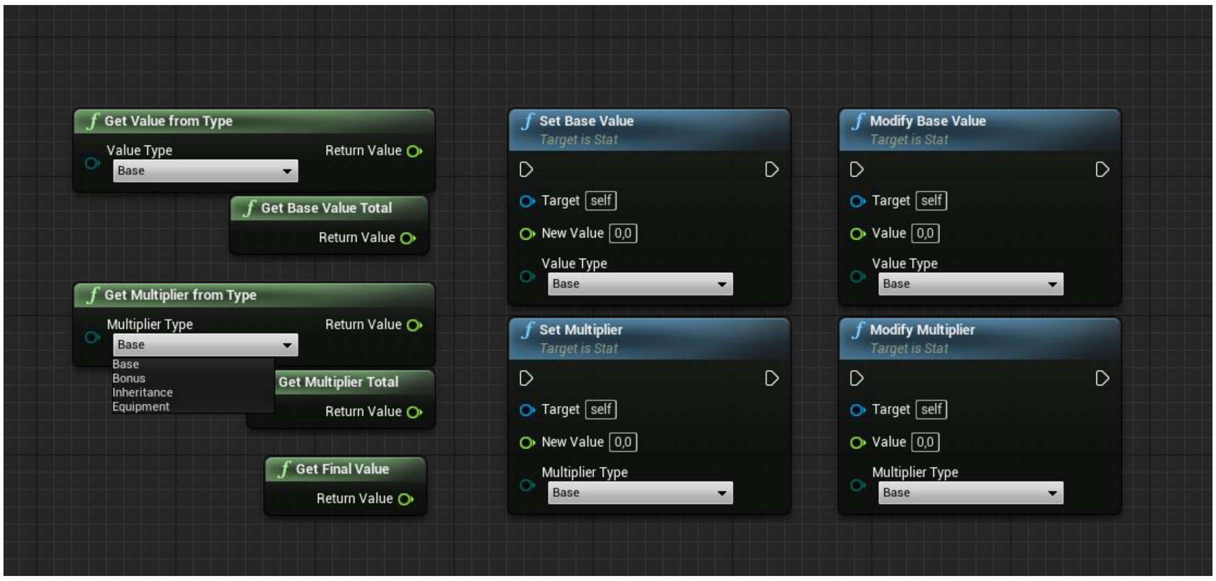
I – Stats

1) Working with Stats

To create a new stat, create a blueprint derived from the class « Stat ».
Each stat has 4 types of different values, both for their Base Value and Multiplier :

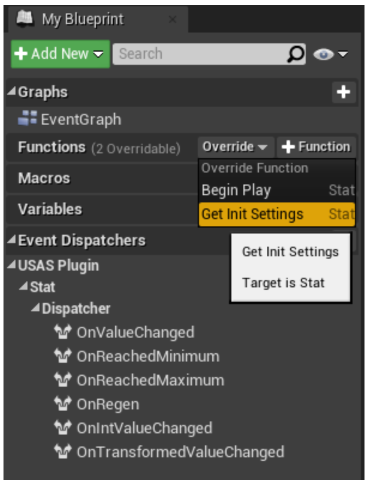
- **Base** : The base value of your stat which you can set manually
- **Bonus** : The bonus value of your stat which you can also set manually, but is distinct from the Base
- **Inheritance** : The value inherited from the « Inheritances » option
- **Equipment** : The value inherited from the StatsComponent's equipment

You can get, set, or modify these values with the following functions :



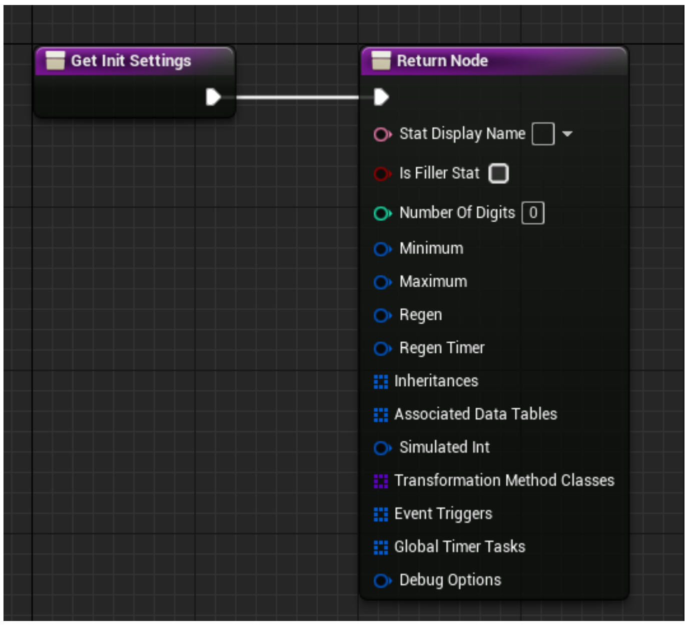
The **Base Value Total** and the **Multiplier Total** are the sum of their 4 respective types: **Base**, **Bonus**, **Inheritance** and **Equipment**.
The **Final Value** is usually the function you'll use to get the value of your Stat, and is the product of the **Base Value Total** and the **Multiplier Total**.

2) Stats Behaviors



To set the behavior of your Stat, which will determine its internal functioning, go into **Functions** -> **Override** -> **GetInitSettings** inside your Stat blueprint panel.

You will then find the **GetInitSettings** function node, which is the core of your stat. (See below)



You can drag any structure of this node and use the functions under the « TEMPLATES » section to fill it.

Setting : Description. (*Related functions*)

Stat Display Name : The name of your stat. (*GetStatName*)

Is Filler Stat : Check this box if your stat is evolving between bounds and has a regen value.

- Only has a **Base** value (no bonus, inheritance or equipment)
- Doesn't have a multiplier
- The **Final Value** is always equal to the **Base** value.

Number of Digits : The number of digits displayed when you use the function *GetValueAsText*.

Minimum : The Minimum value of your Stat. (*GetMinValue*)
Dispatcher: *OnReachedMinimum.*

Maximum : The Maximum value of your Stat. (*GetMaxValue*, *GetMaxRef*)
Dispatcher: *OnReachedMaximum.*

Regen : The Regen value of your Stat. (*GetRegenValue*, *GetRegenRef*, *ApplyRegen*, *SetRegenPaused*)
Dispatcher: *OnRegen.*

Regen Timer : Gives you 3 options for your regen :

- **Leave it blank** : You can then apply the regen manually with the function *ApplyRegen*
- **Use Personal Timer**
- **Use Global Timer System** (see the Global Timer System section).

Inheritances : Allows to inherit values from other Stats. Use the “Mod” option to restrict inheritances to certain StatsComponents. If you want your inheritance to work for any StatsComponent, type in “All”.
You can use a Basic Inheritance with a ratio or a Curved Inheritance, for which you can create a Curve Float where the X axis will be the value of the Parent Stat and the Y axis the value inherited.

Associated Data Tables : Allows you to link a Stat to a Data Table, which will trigger Stats modifiers and events whenever certain values are reached.
To create a Data Table for this option, use the “**StatsModifiers**” structure.
The row name of the Data Table corresponds to the value of your linked Stat.

Simulated Int : Simulates an Integer value. (*GetIntValue*)
Dispatcher: *OnIntValueChanged.*

Transformation Method Classes : Adds calculation methods for your Stat. (*GetTransformedValue()*)
Dispatcher: *OnTransformedValueChanged.*

- ➔ Create a class from the "*StatTransformationMethod*" class and override its function "*GetTransformedValue()".*
- ➔ You can then add it in the GetInitSettings() function of your Stat.
- ➔ You can access its value with the function "GetTransformedValue" from your Stat and use the dispatcher "OnTransformedValueChanged" which is called whenever a TransformedValue is updated. The index asked is the one used in GetInitSettings(), so keep that in mind if you want to use multiple TransformationMethods for the same Stat.

It can be used if you want to easily have access to a value derived from your stat with a formula, and don't want to calculate it every time you need to access it.
A good example could be a "Critical Rate" Stat with a certain value, to which you want to apply a formula to get the actual rate, depending on multiple other variables.
You could apply the formula every time you want to get the usable rate, or use a TransformationMethod that will keep it updated for you.

Event Triggers : Triggers effects whenever a certain value range is reached. Can also be set manually with the functions *AddEventTrigger* and *RemoveEventTrigger*.

You can create a class from the “*EventTriggerEffect*” class and override “*EventTriggerIn()”* which is triggered whenever the value enters the range, and “*EventTriggerOut()”* whenever it leaves the range.

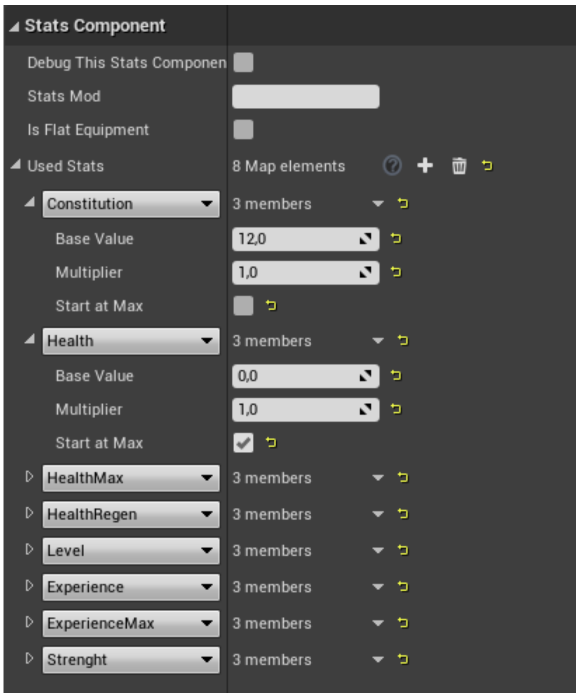
Global Timer Tasks : Triggers effects whenever a Global Timer ticks. (See the Global Timer System section).

Debug Options : Prints to screen the value of your Stat. You will also need to check the “Debug This StatsComponent” option inside your StatsComponent.

StatsComponent

1) Settings

To create a StatsComponent, create a new blueprint derived from the class “StatsComponent”.
You will then have different options inside its class defaults :



Stats Component	
Debug This Stats Componen	☐
Stats Mod	
Is Flat Equipment	☐
Used Stats	8 Map elements
Constitution	3 members
Base Value	12,0
Multiplier	1,0
Start at Max	☐
Health	3 members
Base Value	0,0
Multiplier	1,0
Start at Max	✓
HealthMax	3 members
HealthRegen	3 members
Level	3 members
Experience	3 members
ExperienceMax	3 members
Strenght	3 members

Debug This Stats Component : In pair with the DebugOptions of your Stat behavior, it allows to print to screen the value of your stat.

Stats Mod : A mod allows your stats to have different behaviors inside the same StatsComponent (Inheritances and Associated Data Tables).

Is Flat Equipment : If checked, this StatsComponent’s Stats won’t have a FinalValue. Instead, they will add their BaseValue and Multiplier to their owner. **If you don’t want to increase the owner stat, set the Multiplier to 0.**

Used Stats : This is where you can add the Stats used by this StatsComponent and their values. The “Start at Max” box makes this Stat start at its maximum value if it has one.

2) Equipment

Any StatsComponent can equip and unequip other StatsComponents with the functions “EquipStatsComponent()” and “UnequipStatsComponent()”.
The equiped StatsComponent will find all the matching Stats and add their values to its owner StatsComponent. Everytime the values of the equipped StatsComponent change, the values of the owner will be updated.
Checking “IsFlatEquipment” will disable the behavior of the StatsComponent, and allow you to equip its multiplier.

More

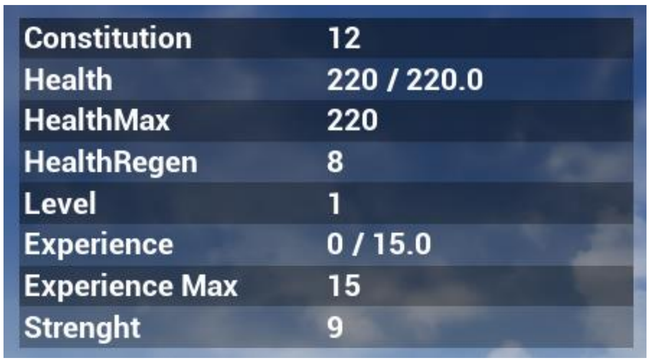
1) Global Timer System

The Global Timer System allows you to bind functions to a same timer.
Inside your GameState, add the GlobalTimerSystem component. You will then be able to add Global Timers to the array in its default settings, with their name and tick rate.

To bind and unbind events from a Global Timer, use the functions *AddGlobalTimerTask* and *RemoveGlobalTimerTask*.

2) Debug Widget

You can use the Debug Widget to quickly see all of your stats in-game.



Constitution	12
Health	220 / 220.0
HealthMax	220
HealthRegen	8
Level	1
Experience	0 / 15.0
Experience Max	15
Strenght	9

3) Functions

There are a lot of functions available inside your Stats or StatsComponents inside the “USAS Plugin” section. You can also call a lot of functions directly from your actors with the QuickFunctions library.

4) Logs

This plugin displays a lot of information inside the Output Log. Make sure the “LogStatsComponent” is checked inside your filters. At least one message must have been displayed for it to show, so if it doesn’t appear, play the level once.

5) Load Default Values

In your StatsComponent, you can override the function *GetStartingValues()* to load its values from wherever you want. If you are implementing an equipment system, this could, for example, allow you to pull default values from a Data Table into your equipable StatsComponent.

Starting Values Method:

- **Use Defaults**: this won’t affect the StatsComponent default values.
- **Override all matching Stats**: Search for the matching Stats and override their default value

- **Override all**: Only the specified Stats from this function will be present in the StatsComponent.