

Table 1: Outlets for AffinitiesComponent

GetUnspentPoints	
► Before	<code>const uint8 OriginalPoints, uint8&ReturnedPoints</code>
► After	<code>const uint8 OriginalPoints, const uint8 ReturnedPoints</code>

SetUnspentPoints	
► Before	<code>const uint8 OriginalPoints, const uint8 InputPoints, uint8&SetPoints</code>
► After	<code>const uint8 OriginalPoints, const uint8 InputPoints, const uint8 SetPoints</code>

— • • • —

Table 2: Outlets for EffectComponent

GetStacks	
► Before	<code>const uint16 OGStacks, int32&ReturnedStacks</code>
► After	<code>const uint16 OGStacks, const int32 ReturnedStacks</code>

OnAddEffect	
► Before	<code>const EffectComponent* EffectToAdd</code>
► After	<code>const EffectComponent* AddedEffect</code>

OnRemoveEffect	
► Before	<code>const EffectComponent* EffectToRemove</code>
► After	<code>const EffectComponent* RemovedEffect</code>

— • • • —

Table 3: Outlets for LevelComponent

GetBaseExpYield	
► Before	<code>const float OriginalYield, float&ReturnedYield</code>
► After	<code>const float OriginalYield, const float ReturnedYield</code>
GetCXP	
► Before	<code>const uint32 OriginalCXP, int32&ReturnedCXP</code>
<i>Note:</i>	<code>ReturnedCXP</code> is <code>int32&</code> instead of <code>uint32&</code> for Blueprint compatability.
► After	<code>const uint32 OriginalCXP const int32 ReturnedCXP</code>
<i>Note:</i>	<code>ReturnedCXP</code> is <code>const int32</code> instead of <code>const uint32</code> for Blueprint compatability.
GetExpYield	
► Before	<code>const float OriginalYield, float&ReturnedYield, const uint16 DefeatedLevel, const uint16 VictoriousLevel</code>
<i>Note:</i>	“Defeated” and “Victorious” levels are provided for flexibility (e.g., in case you want to yield exp differently based on level difference, although technically you could always back-calculate the level difference based on the equation and <code>OriginalYield</code>).
► After	<code>const float OriginalYield, const float ReturnedYield, const uint16 DefeatedLevel, const uint16 VictoriousLevel</code>
<i>Note:</i>	“Defeated” and “Victorious” levels are provided for symmetry with respect to the <code>Before</code> delegate (since <code>ReturnedValue</code> is already calculated, I can’t think of why you would need them, but you never know!).

Continued on next page

Table 3: Outlets for `LevelComponent` (Continued)

GetMaxLevel	
► Before	<code>const uint16 DefaultMax,</code> <code>int32&AttemptedMax</code>
<i>Note:</i>	<code>DefaultMax</code> is defined in the code. It should normally be 100, but may change for certain subclasses (e.g., a <code>BossLevelComponent</code> may have a max of 200 instead). <code>AttemptedMax</code> is <code>int32&</code> instead of <code>uint16&</code> for Blueprint compatability.
► After	<code>const uint16 DefaultMax</code> <code>const int32 ReturnedMax</code>
GetMinLevel	
► Before	<code>const uint16 DefaultMin,</code> <code>int32&AttemptedMin</code>
<i>Note:</i>	<code>DefaultMin</code> is defined in the code. It should normally be 1, but may change for certain subclasses (e.g., a <code>EggLevelComponent</code> may have a min of 0 instead for whatever reason). <code>AttemptedMin</code> is <code>int32&</code> instead of <code>uint16&</code> for Blueprint compatability.
► After	<code>const uint16 DefaultMin</code> <code>const int32 ReturnedMin</code>
<i>Note:</i>	<code>ReturnedCXP</code> is <code>const int32</code> instead of <code>const uint32</code> for Blueprint compatability.
GetBaseExpYield	
► Before	<code>const float OriginalYield,</code> <code>float&ReturnedYield</code>
► After	<code>const float OriginalYield,</code> <code>const float ReturnedYield</code>
SetBaseExpYield	
► Before	<code>const float OldYield,</code> <code>const float InputYield,</code> <code>float&AttemptedYield</code>
► After	<code>const float OldYield</code> <code>const float InputYield,</code> <code>const float NewYield</code>
<i>Note:</i>	<ul style="list-style-type: none"> ▷ <code>OldYield</code> is the yield prior to calling <code>SetBaseExpYield</code>, ▷ <code>InputYield</code> is the original, unmodified input to <code>SetBaseExpYield</code>, ▷ <code>AttemptedYield</code> is the modified value that will be used to set the base exp yield.

Continued on next page

Table 3: Outlets for LevelComponent (Continued)

SetCXP	
► Before	<pre>const uint32 OldCXP, const int32 InputCXP, int32& AttemptedCXP</pre>
<i>Note:</i>	AttemptedCXP is int32& instead of uint32& for Blueprint compatability.
► After	<pre>const uint32 OldCXP const int32 InputCXP, const uint32 NewCXP</pre>
<i>Note:</i>	<p>StatsComponent subscribes to this in order to change stats on level change.</p> <ul style="list-style-type: none"> ▷ OldCXP is the cumulative experience points prior to calling SetCXP, ▷ InputCXP is the original, unmodified input to SetCXP, ▷ AttemptedCXP is the modified value that will be used to set the cumulative experience points.

• • •

Table 4: Outlets for StatsComponent

ApplyDamage	
► Before	<pre>float& BasePower, float& CritMultiplier, float& RandFluct, float& Stab, float& TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats</pre>
<i>Note:</i>	Other quantities, such as the Attacker's attacking Stat, may be calculated from the given quantities.
► After	<pre>const float BasePower, const float CritMultiplier, const float RandFluct, const float& Stab, const float& TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats</pre>

Continued on next page

Table 4: Outlets for StatsComponent (Continued)

CalculateDamage	
► Before	<pre>float&BasePower, float&CritMultiplier, float&RandFluct, float&Stab, float&TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats</pre>
► After	<pre>const float BasePower, const float CritMultiplier, const float RandFluct, const float&Stab, const float&TypeAdvantage, UCombatStatsComponent* Attacker, UCombatStatsComponent* OwningStats</pre>
<i>Note:</i>	The difference between calculating damage and applying damage is theoretical. For example, low-level AI might use CalculateDamage to make decisions. On the other hand, applying the damage might invoke some kind of reaction, like raising Physical Attack if hit by a move it's weak to.
GetCritMult	
► Before	<pre>float&BaseMultiplier, float&CritBonus, UCombatStatsComponent* OwningStats</pre>
<i>Note:</i>	Total crit bonus is BaseMultiplier + CritBonus ; for example, 1.5 + 0.2.
► After	<pre>const float BaseMultiplier, const float CritBonus, UCombatStatsComponent* OwningStats</pre>

Continued on next page

Table 4: Outlets for StatsComponent (Continued)

ModifyStat	
► Before	<pre>const EStatEnum TargetStat, const EStatValueType ValueType, const EModificationMode Mode, const float OriginalValue, float& AttemptedValue</pre>
► After	<pre>const EStatEnum TargetStat, const EStatValueType ValueType, const EModificationMode Mode, const float OriginalValue, const float NewValue</pre>
<i>Note:</i>	All “ModifyStat” functions from StatsComponent (such as ModifyStatsUniformly or RandomizeStats) go through ModifyStatInternal, which calls this Outlet.
RandomizeStats	
► Before	<pre>const EStatEnum TargetStat, const FStatRandParams OriginalParams, FStatRandParams& ParamsToBeUsed</pre>
► After	<pre>const EStatEnum TargetStat, const FStatRandParams OriginalParams, const FStatRandParams UsedParams</pre>
<i>Note:</i>	The EStatEnum is not the actual FStat. To get the FStat (such as FHealth), use StatsComponent::GetStat(EStatEnum)
RecalculateStats	
► Before	<pre>const EStatEnum TargetStat, const bool bResetCurrent, const float OriginalCurrent, const float OriginalPermanent</pre>
► After	<pre>const EStatEnum TargetStat, const bool bResetCurrent, const float OriginalCurrent, const float OriginalPermanent</pre>