

TL; DR

TODO:

0.1 Structure

- **EffectableComponents** are **ActorComponents** that allow for delegation (effects). They have predefined places that allow for code modification.
 - Let's use **StatsComponent** as an example. Say we want a Pokémon-style “Adamant” nature (+10% PhA/−10%SpA). One such place for modification is in the function **RecalculateStats**.

```
void UStatsComponent::RecalculateStats(const bool bResetCurrent)
{
    for(FStat* Stat : StatsArray)
    {
        ExecuteBeforeRecalculateStats(Stat, bResetCurrent);
        Stat->Update(GetLevel(), bResetCurrent);
        ExecuteAfterRecalculateStats(Stat, bResetCurrent);
    }
}
```

- **Delegate arrays** are variables inside of **EffectableComponents**. They hold functions that execute when needed.
 - Let's use **StatsComponent**'s **AfterRecalculateStatsArray** in our example. In this case, after stats are recalculated (say, on level-up), the base PhA would increase by 10% and the base SpA would decrease by 10% (additively):

```

// Define "adamant" delegate (+10% PhA/-10% SpA)
UStatsComponent::FRecalculateStatsDelegate AdamantRecalculateDelegate;
AdamantRecalculateDelegate.BindLambda(InFuncor: [StatsComponent](FStat* Stat, bool bResetCurrent) -> void
{
    // +10% PhA
    if ( Stat->Name() == StatsComponent->PhysicalAttack.Name())
    {
        Stat->ModifyValue( Modifier: 10, EStatValueType::Permanent, EModificationMode::AddPercentage);
        if (bResetCurrent)
            Stat->ModifyValue( Modifier: 10, EStatValueType::Current, EModificationMode::AddPercentage);
    }

    // -10% SpA
    if ( Stat->Name() == StatsComponent->SpecialAttack.Name())
    {
        Stat->ModifyValue( Modifier: -10, EStatValueType::Permanent, EModificationMode::AddPercentage);
        if (bResetCurrent)
            Stat->ModifyValue( Modifier: -10, EStatValueType::Current, EModificationMode::AddPercentage);
    }
});
StatsComponent->AfterRecalculateStatsArray.Add(AdamantRecalculateDelegate);

```

0.2 List of EffectableComponents and Delegate Arrays

The following tables show all implemented **EffectableComponents** and their delegate arrays. Note the “base name” indicates existence of:

1. the delegate type **FBaseNameDelegate**;
2. the before/after arrays of delegates:
TArray<FBaseNameDelegate> BeforeBaseNameArray; and
3. a function for each before/after to execute the arrays: **ExecuteBeforeBaseName (...)**.

Note that the philosophy applies to what is *probable* rather than what is *possible*. Hence the list meant to be practical rather than exhaustive.

Table 1: Delegate Arrays for **AffinitiesComponent**

Delegate Array Base Name	Parameters	Note
GetUnspentPoints	int& Unspent points	
SetUnspentPoints	int& Current unspent points	
	int& Attempted value being set	

Table 2: Delegate Arrays for `LevelComponent`

Delegate Array Base Name	Parameters	Note
GetBaseExpYield	<code>int</code> Unaltered base exp yield	
SetBaseExpYield	<code>int</code> Unaltered base exp yield	
	<code>int&</code> Attempted value being set	
GetExpYield	<code>UStatsComponent*</code> Victorious Monster	
	<code>float&</code> Awarded exp	
GetCumulativeExp	<code>int&</code> Current CXP	
SetCumulativeExp	<code>int</code> Current CXP	All other level/exp setters go through here!
	<code>int&</code> Attempted CXP	
AddExp	<code>int</code> Current exp	No GetExp (=GetLevel)
	<code>int&</code> Added exp	
SetLevel	<code>int</code> Current level	No GetLevel
	<code>int&</code> Attempted level	
MaxLevel	<code>int&</code> The maximum level	This is a getter function only
MinLevel	<code>int&</code> The minimum level	This is a getter function only

Table 3: Delegate Arrays for `StatsComponent`

Delegate Array Base Name	Parameters	Note
RandomizeStats	<code>int&</code> Min base stat	This is the one with four parameters, but is called by all others
	<code>int&</code> Max base stat	
	<code>int&</code> Min base pairs	
	<code>int&</code> Max base pairs	

Continued on next page

Table 3: Delegate Arrays for `StatsComponent` (Continued)

RecalculateStats	<code>FStat*</code> Each stat in the loop	Rather than make each individual stat an <code>EffectableComponent</code> , you can go stat-by-stat here
	<code>bool</code> If true, reset the current stats to match the newly-calculated permanent stats	
ModifyStat	<code>FStat*</code> The stat being modified	
	<code>float&</code> The value of modification	
	<code>EStatValueType&</code> The value type (e.g., current or permanent)	
	<code>EModificationMode&</code> E.g., additive or multiplicative	

0.3 Making Your Own Effects

Suppose you want to make your own effect from scratch. **TODO: Lay this out!**