Easy Stats Kit_v1.0

Thank you for purchasing!

Easy Stats Kit is a powerful and flexible attribute management plugin designed to help game developers easily define and manage various attribute values and their modifiers. Whether it's a simple tower defense game or a complex RPG attribute system, Easy Stats Kit can meet your needs and reduce the hassle of writing complex mechanisms from scratch.

Easy Stats Kit provides an intuitive editor window that allows developers to quickly integrate it into existing projects through simple configurations and a C# API, achieving highly customizable attribute management.

Features

Attribute Definition: Easily define various attributes for any object, such as health, attack power, skill cooldown time, etc.

Attribute Groups: Achieve high reusability of attributes by creating attribute groups, supporting multi-level inheritance and combination.

Modifier System: Support adding and removing modifiers to achieve effects like buffs and debuffs, supporting various calculation types and algorithm priority sorting.

Initialization Methods: Support automatic and manual initialization, flexibly adapting to different development needs.

Data Saving and Loading: Support saving attribute data as JSON format and loading shared data across different scenes.

Stats Manager: Built-in powerful Stats Manager to help developers manage all attributes of an object, providing comprehensive attribute operation interfaces.

Intuitive Editor Attribute Interface: Provides a convenient editor attribute interface, allowing management and viewing of runtime attribute status information.

Whether you need attributes to be affected by items, spells, skills, or other factors, Easy Stats Kit can manage and accurately calculate attribute values in an organized manner. Simply declare an MKStats variable, set its initial value, and then add modifiers as needed. The final attribute value can be obtained from the relevant attribute.

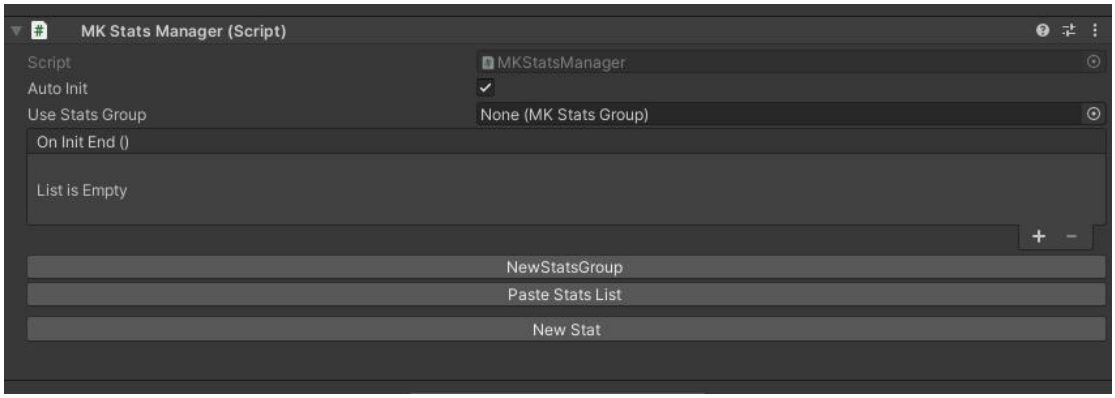
Easy Stats Kit plugin includes full source code, allowing you to customize and extend the source code as needed.

With the Easy Stats Kit plugin, you can efficiently manage and dynamically adjust various attributes in the game, providing solid support and convenience for your game development.

If convenient, please leave a review on the store page. It is very important to me and motivates me to continue developing more new features!

Quick Start

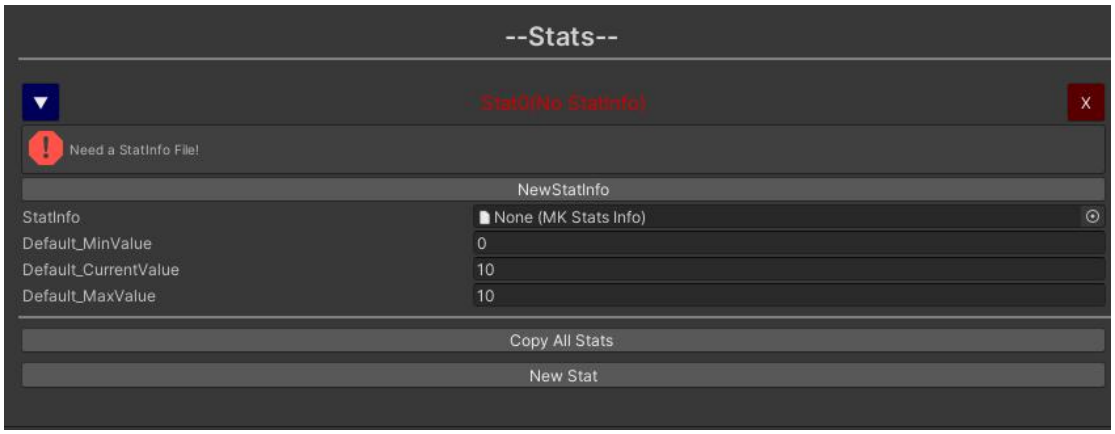
1. Set up MKStatsManager



Add the MKStatsManager script to any game object (it could be your player object, a monster, or any other object that needs dynamically changing attributes).

2. Create Stat

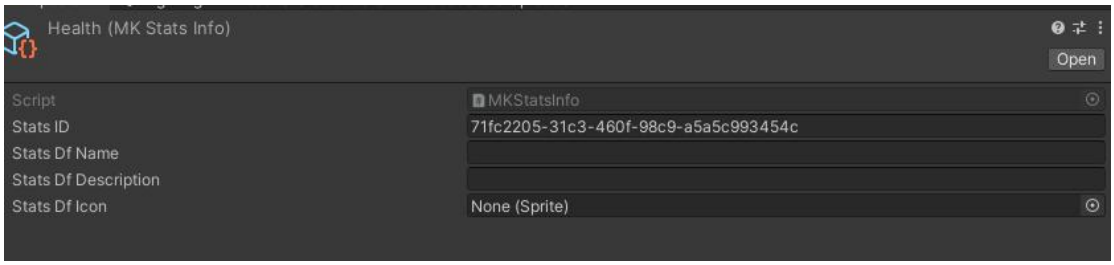
Click the 'New Stat' button on the MKStatsManager component to create a new attribute.

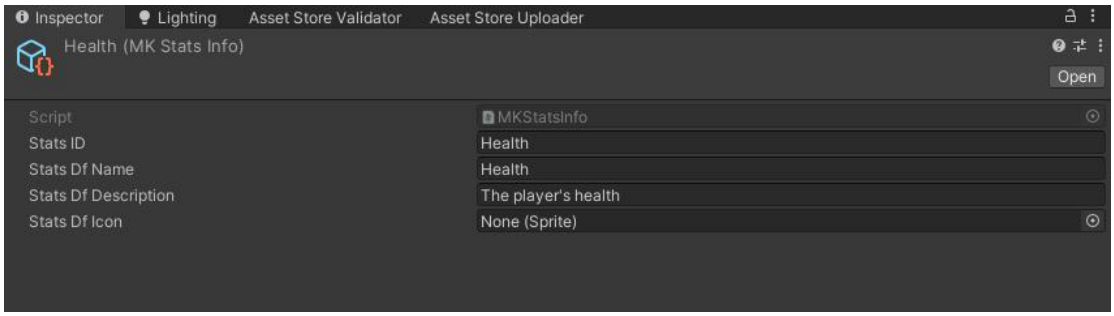


3. Create StatsInfo

You will be prompted that there is no configured Stat info. Don't worry, click the 'New StatInfo' button to create a new attribute information configuration file. Let's take health points as an example.

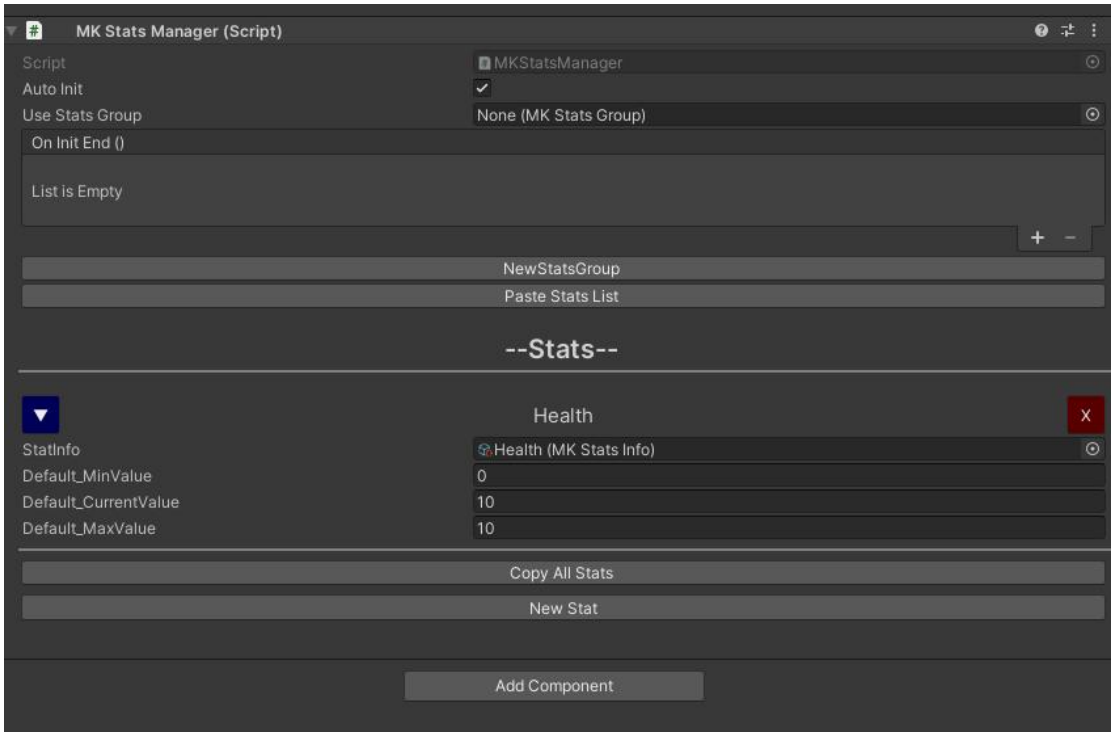
A unique Stats ID will be generated by default. You can customize it to another ID as shown below:





4. Configure Stat

After configuring your StatsInfo, go back to the MKStatsManager component on the object. It will automatically assign the previously created Health attribute information to this new attribute.



You can configure their default initial values:

InitDMinValue: Minimum value

InitDValue: Current value

InitDMaxValue: Maximum value

5. Done!

Everything is ready. You have configured a health attribute for an object. You can run the game and expand the Health attribute on the MKStatsManager to see various information about the attribute, including the current value and all current modifiers.

What's Next?

Next, you can get and dynamically set the value of the attribute in code (for example, taking damage or healing health points). You can also add modifiers to the attribute (such as receiving a blessing buff, increasing maximum health by 50%).

```
public MKStatsManager mKStatsManager; // Get your attribute manager

mKStatsManager.SetInit(); // Manually or automatically initialize the attribute manager

MKStats health = mKStatsManager.FindValidationStat("Health"); // Get the corresponding attribute object by ID

health.SetNowBaseValue(health.GetNowBaseValue() - damage); // Inflict damage

// Another way: mKStatsManager.AddModToStats("Health", new MKStatModifier(0.5f, MKStatModType.PercentAdd, MKStatModTargetType.MaxValue， 0, "UpHeathBuff")); // Add modifier
```

```
health.AddModifier(new MKStatModifier(0.5f, MKStatModType.PercentAdd,
MKStatModTargetType.MaxValue，0, "UpHeathBuff")); // Add modifier
```

Which can set the MKStatModifier Order attribute, the larger the Order, the more after the sort, the more after the calculation,

By default, the Order of MKStatModType from small to large is,

Add=100,PercentAdd=200,PercentMult = 300

You can directly use an MKStats variable anywhere and manage and use it with your own code. It does not necessarily require MKStatsManager as MKStats already includes a complete API.

Example Scene

In the plugin package, we provide an example scene to demonstrate the functionality of the MKStats plugin. You can follow these steps to run and view the example scene:

1. Import the plugin package and find the `MKStats/Example` folder in the project view.

2. Open the `BaseMKStatsDemo` scene file.

3. In the scene, you can see the pre-configured MKStatsManager and related UI elements.

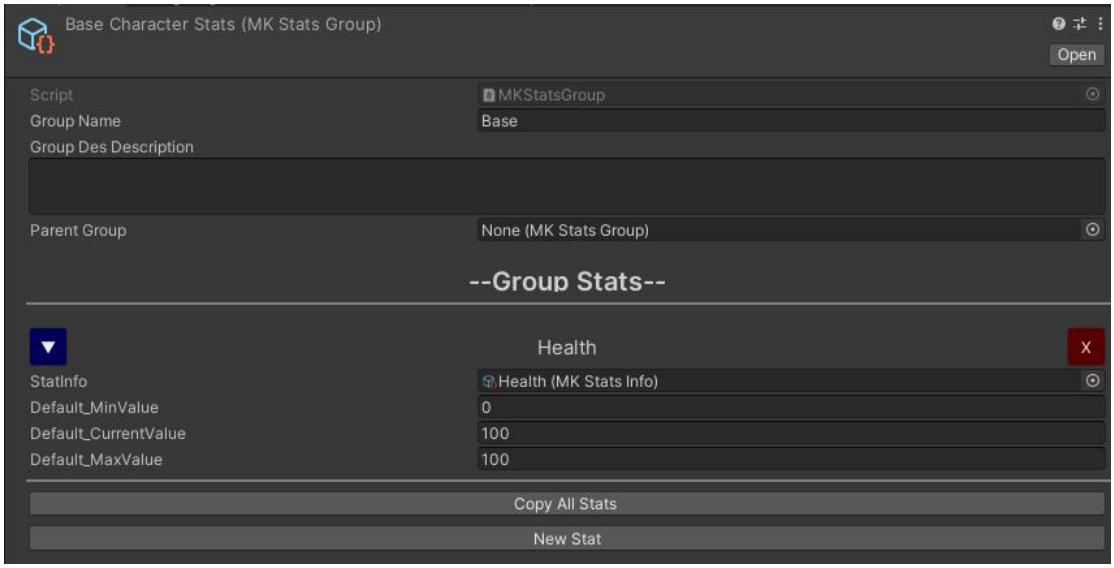
4. Run the scene and experience how to manage and operate attribute values through the plugin.

The `TestPlayerStatsController` object in the scene contains the `TestMKStatsSet` script, which implements most common attribute interface usages. You can open the script for reference.

Advanced Usage

1. Using MKStatsGroup to Pre-configure Attribute Groups

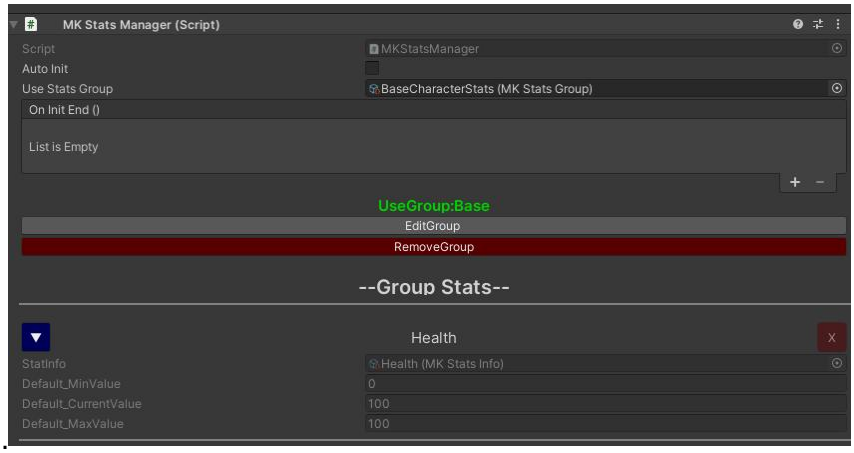
1.1 Create MKStatsGroup: Right-click in the project view and select `Create -> MKGame -> StatsGroup` to create a new attribute group file. Name and save the attribute group.



1.2 Configure Attribute Group: Add attribute information files to the attribute group and configure the initial default value, minimum value, and maximum value for each attribute.

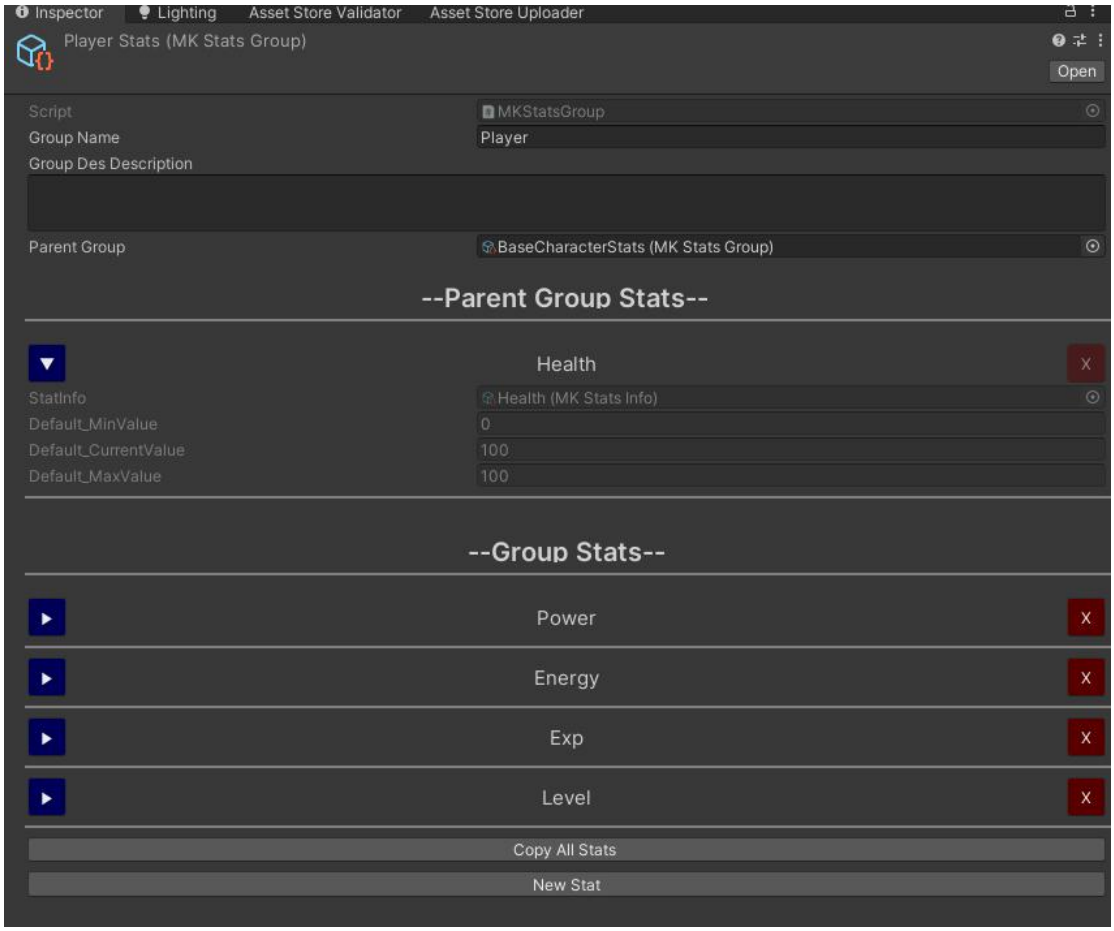
1.3 Set the Attribute Group in MKStatsManager: Set the attribute group to be used (UseStatsGroup) in the Inspector of MKStatsManager.

An MKStatsManager configured with MKStatsGroup can also define its own other Stats. At runtime, MKStatsManager will automatically merge and instantiate all attributes



2. Inheritance and Combination of Attribute Groups

2.1 Attribute Group Inheritance: You can create an attribute group and set its parent attribute group. The child attribute group will automatically inherit all attributes from the parent attribute group and can add attributes on this basis.



2.2 Combination of Attribute Groups and Directly Configured Attributes:
MKStatsManager supports using both pre-configured attribute groups and directly configured attributes simultaneously. At runtime, MKStatsManager will merge these attributes and manage them uniformly.

3. Automatic and Manual Initialization

3.1 By default, MKStatsManager will automatically initialize attributes in the `Awake` method.

3.2 If manual initialization is required, you can disable the `AutoInit` option.

3.3 To manually initialize attributes when needed, you can call the `MKStatsManager.SetInit()` method to complete the initialization.

API Interface

MKStatsManager Class

void SetInit()

// Initializes all stats.

MKStats AddOneNewStat(MKStatsInfo mKStatsInfo)

// Adds a new stat.

// Parameters: MKStatsInfo mKStatsInfo - The stat information to add.

void RemoveOneStat(int id)

// Removes a stat by index.

// Parameters: int id - The index of the stat to remove.

void RemoveOneStat(MKStats stats)

// Removes a stat by instance.

// Parameters: MKStats stats - The stat instance to remove.

MKStats FindValidationStat(MKStatsInfo statsinfo)

// Finds a stat by information.

// Parameters: MKStatsInfo statsinfo - The stat information.

MKStats FindValidationStat(string statId)

// Finds a stat by ID.

// Parameters: string statId - The stat ID.

void GetNowAllModifier(List<MKStatModifier> OutAllModifierList)

// Retrieves all current modifiers.

// Parameters: List<MKStatModifier> OutAllModifierList - List to store all current modifiers.

void AddModToStats(string statId, MKStatModifier mKStatModifier)

// Adds a modifier to a stat.

// Parameters: string statId - The stat ID.

// Parameters: MKStatModifier mKStatModifier - The modifier instance.

void AddModToStats(MKStatsInfo statsinfo, MKStatModifier mKStatModifier)

// Adds a modifier to a stat.

// Parameters: MKStatsInfo statsinfo - The stat information.

// Parameters: MKStatModifier mKStatModifier - The modifier instance.

void RemoveModToStats(string statId, MKStatModifier mKStatModifier)

// Removes a modifier from a stat.

// Parameters: string statId - The stat ID.

// Parameters: MKStatModifier mKStatModifier - The modifier instance.

void RemoveModToStats(MKStatsInfo statsinfo, MKStatModifier mKStatModifier)

// Removes a modifier from a stat.

// Parameters: MKStatsInfo statsinfo - The stat information.

// Parameters: MKStatModifier mKStatModifier - The modifier instance.

void RemoveModToStatsFromSource(MKStatsInfo statsinfo, string sourceKey)

// Removes all modifiers from a specific source from a stat.

// Parameters: MKStatsInfo statsinfo - The stat information.

// Parameters: string sourceKey - The source key of the modifiers.

void RemoveModToStatsFromSource(string statsid, string sourceKey)

// Removes all modifiers from a specific source from a stat.

// Parameters: string statsid - The stat ID.

// Parameters: string sourceKey - The source key of the modifiers.

void RemoveAllModToStatsFromSource(string sourceKey)

// Removes all modifiers from a specific source from all stats.

// Parameters: string sourceKey - The source key of the modifiers.

string GetSaveJsonData()

// Retrieves the JSON representation of current stat data.

void LoadSaveDataFromJson(string jsondata)

// Loads stat data from JSON.

// Parameters: string jsondata - JSON data string.

MKStatsInfo Class

string GetShowName()

// Retrieves the display name.

string GetShowDescription()

// Retrieves the display description.

Sprite GetShowIcon()

// Retrieves the display icon.

MKStatModifier Class

public float Value { get; set; }

// The value of the modifier.

public MKStatModType Type { get; set; }

// The type of the modifier (e.g., addition, percentage addition, percentage multiplication).

public MKStatModTargetType TargetType { get; set; }

// The target type of the modifier (current value, maximum value, minimum value).

public int Order { get; set; }

// The order in which the modifier is applied.

//The larger the Order, the more after the sort, the more after the calculation

public string SourceKey { get; set; }

// The source identifier of the modifier.

public MKStats LastAppendMKStats { get; }

// The most recently appended MKStats instance (read-only).

// Full parameter constructor

// Creates a new MKStatModifier instance with all parameters.

public MKStatModifier(float value, MKStatModType type, MKStatModTargetType ttype, int order, string sourcekey, MKStats TargetStat)

// Copy constructor

// Creates a new MKStatModifier instance by copying another MKStatModifier.

public MKStatModifier(MKStatModifier other)

// Partial parameter constructors

// Creates a new MKStatModifier instance with some default parameters.

public MKStatModifier(float value, MKStatModType type, MKStatModTargetType ttype)

public MKStatModifier(float value, MKStatModType type, MKStatModTargetType ttype, int order)

public MKStatModifier(float value, MKStatModType type, MKStatModTargetType ttype, string sourcekey)

// Copies data from another MKStatModifier and optionally updates LastAppendMKStats.

public void CloneFromOtherData(MKStatModifier mKStatModifier, MKStats NewMKStats)

// Clones the current MKStatModifier with a new MKStats instance.

public MKStatModifier CloneNew(MKStats NewMKStats)

string GetSaveJsonData()

// Retrieves the JSON representation of the modifier data.

static MKStatModifier LoadFromJsonData(string jsondata, MKStats targetstat)

// Loads a modifier from JSON data.

// Parameters: string jsondata - JSON data string.

// Parameters: MKStats targetstat - The target property instance.

MKStats Class

// Available event listeners

UnityEvent<float, float> onBaseMinValueChanged

UnityEvent<float, float> onBaseValueChanged

UnityEvent<float, float> onBaseMaxValueChanged

UnityEvent<float, float> onMinValueChanged

UnityEvent<float, float> onValueChanged

UnityEvent<float, float> onMaxValueChanged

UnityEvent<MKStatModifier> OnAddModifier

UnityEvent<MKStatModifier> OnRemoveModifier

void SetNowBaseMaximum(float value)

// Sets the current maximum value.

// Parameters: float value - The maximum value.

void SetNowBaseMinimum(float value)

// Sets the current minimum value.

// Parameters: float value - The minimum value.

void SetNowBaseValue(float value)

// Sets the current value.

// Parameters: float value - The current value.

void AddModifier(MKStatModifier mod)

// Adds a modifier.

// Parameters: MKStatModifier mod - The modifier instance.

bool RemoveModifier(MKStatModifier mod)

// Removes a modifier.

// Parameters: MKStatModifier mod - The modifier instance.

bool RemoveAllModifiersFromSource(string sourceKey)

// Removes all modifiers from a specific source.

// Parameters: string sourceKey - The source key of the modifiers.

string GetSaveJsonData()

// Retrieves the JSON representation of the property data.

static MKStats LoadFromJsonData(string jsondata, MKStatsInfo mKStatsInfo)

// Loads a property from JSON data.

// Parameters: string jsondata - JSON data string.

// Parameters: MKStatsInfo mKStatsInfo - The property information.

Support and Contact Information

If you encounter any issues or have any suggestions during use, please contact us through the following ways:

Email: 1174283584@qq.com

Thank you for your support, and we will continue to improve and refine the plugin to provide you with a better experience. Please leave a positive review!